
A Textbook of Non-textbook Cryptography

Explanations and Elaborations

by

Wenbo Mao
(wenbo.mao@hp.com)

CONTENTS

LIST OF FIGURES	xvi
LIST OF TABLES	xx
PREFACE	xxi
I INTRODUCTION	1
1 BEGINNING WITH A SIMPLE COMMUNICATION GAME...	3
1.1 A Communication Game	4
1.1.1 Our First Application of Cryptography	4
1.1.2 An Initial Hint on the Foundation of Cryptography	6
1.1.3 Security: More Than a Bless from Mathematical Properties	6
1.1.4 Modern Role of Cryptography: Ensuring Fair Play of Games	7
1.2 Criteria for Desirable Cryptographic Systems/Protocols	8
1.2.1 Stringency of Protection Tuned to Application Needs	9
1.2.2 Confidence in Security Based on Established “Pedigree”	10
1.2.3 Efficiency	11
1.2.4 Use of Practical and Available Primitives and Services	12
1.2.5 Explicitness	14
1.2.6 Openness	17
1.3 Chapter Summary	18
	v

2	WRESTLING BETWEEN SAFEGUARD AND ATTACK	20
2.1	Encryption	21
2.2	Vulnerable Environment (the Dolev-Yao Threat Model)	23
2.3	Authentication Servers	24
2.4	Security Properties for Authenticated Key Establishment	26
2.5	Protocols for Authenticated Key Establishment Using Encryption	27
2.5.1	Protocols Serving Message Confidentiality	27
2.5.2	Attack, Fix, Attack, Fix, ...	29
2.5.3	Protocol With Message Authentication	34
2.5.4	Protocol With Challenge-Response	38
2.5.5	Protocol With Entity Authentication	40
2.5.6	Protocol Using Public-key Cryptosystems	42
2.6	Chapter Summary	48
II	MATHEMATICAL FOUNDATIONS	49
	STANDARD NOTATION	51
3	PROBABILITY	53
3.1	Basic Concept of Probability	54
3.2	Properties	55
3.3	Basic Calculation	55
3.3.1	Addition Rules	55
3.3.2	Multiplication Rules	56
3.3.3	The Law of Total Probability	57
3.4	Random Variables and Their Probability Distributions	58
3.4.1	Uniform Distribution	59
3.4.2	Binomial Distribution	60
3.5	The Birthday Paradox	61
3.6	Information Theory	63
3.6.1	Properties of Entropy	64
3.7	Redundancy in Natural Languages	65
3.8	Chapter Summary	66

4	COMPUTATIONAL COMPLEXITY	68
4.1	Introduction	68
4.2	Turing Machines	69
4.3	Polynomial-Time	71
4.4	Polynomial-Time Computational Problems	73
4.5	Algorithms and Computational Complexity Expressions	75
4.5.1	Greatest Common Divisor	76
4.5.2	Extended Euclid's Algorithm	77
4.5.3	Time Complexity of Euclid's Algorithms	78
4.5.4	Two Expressions for Computational Complexity	80
4.5.5	Modular Arithmetic	81
4.5.6	Modular Exponentiation	84
4.6	Probabilistic Polynomial-Time	86
4.6.1	Subclass "Always Fast and Always Correct"	88
4.6.2	Subclass "Always Fast and Probably Correct"	90
4.6.3	Subclass "Probably Fast and Always Correct"	92
4.6.4	Subclass "Probably Fast and Probably Correct"	95
4.7	Efficient Algorithms	99
4.8	Nondeterministic Polynomial-Time	101
4.8.1	Nondeterministic Polynomial-time Complete	105
4.9	Non-Polynomial Bounds	106
4.10	Polynomial-time Indistinguishability	108
4.11	Theory of Computational Complexity and Modern Cryptography	110
4.11.1	A Necessary Condition	111
4.11.2	Not a Sufficient Condition	112
4.12	Chapter Summary	113
5	ALGEBRAIC FOUNDATIONS	115
5.1	Groups	115
5.1.1	Lagrange's Theorem	118
5.1.2	Order of Group Element	120
5.1.3	Cyclic Groups	121
5.1.4	The Multiplicative Group \mathbb{Z}_N^*	124

5.2	Rings and Fields	125
5.3	Structure of Finite Fields	128
5.3.1	Finite Fields of Prime Numbers of Elements	128
5.3.2	Finite Fields Modulo Irreducible Polynomials	130
5.3.3	Finite Fields Constructed Using Polynomial Basis	135
5.3.4	Primitive Roots	139
5.3.5	Field Construction Using Normal Basis	140
5.4	Chapter Summary	144
6	NUMBER THEORY	145
6.1	Congruences and Residue Classes	145
6.1.1	Congruent Properties for Arithmetic in \mathbb{Z}_n	147
6.1.2	Solving Linear Congruence in \mathbb{Z}_n	147
6.1.3	The Chinese Remainder Theorem	149
6.2	The Euler's Phi Function	154
6.3	The Theorems of Fermat, Euler and Lagrange	155
6.4	Quadratic Residues	156
6.4.1	Quadratic Residuosity	157
6.4.2	Legendre-Jacobi Symbols	159
6.5	Computing Square Roots Modulo an Integer	162
6.6	Blum Integers	168
6.7	Chapter Summary	170
III	BASIC CRYPTOGRAPHIC TECHNIQUES	171
7	ENCRYPTION — SYMMETRIC TECHNIQUES	173
7.1	Introduction	173
7.1.1	Chapter Outline	175
7.2	Substitution Ciphers	176
7.2.1	Simple Substitution Ciphers	176
7.2.2	Polyalphabetic Ciphers	178
7.3	The Vernam Cipher and the One-Time Pad	179
7.4	Classical Ciphers: Usefulness and Security	180

7.5	The Data Encryption Standard (DES)	183
7.5.1	A Description of the DES	184
7.5.2	The Kernel Functionality of the DES: Random and Non-linear Distribution of Message	186
7.5.3	The Security of the DES	187
7.6	The Advanced Encryption Standard (AES)	188
7.6.1	An Overview of the Rijndael Cipher	189
7.6.2	The Internal Functions of the Rijndael Cipher	190
7.6.3	Summary of the Roles of the Rijndael Internal Functions	194
7.6.4	Fast and Secure Implementation	194
7.6.5	Positive Impact of the AES on Applied Cryptography	195
7.7	Confidentiality Modes of Operation	196
7.7.1	The Electronic Codebook Mode (ECB)	197
7.7.2	The Cipher Block Chaining Mode (CBC)	198
7.7.3	The Cipher Feedback Mode (CFB)	199
7.7.4	The Output Feedback Mode (OFB)	200
7.7.5	The Counter Mode (CTR)	201
7.8	Key Channel Establishment for Symmetric Cryptosystems	202
7.9	Chapter Summary	203
8	ENCRYPTION — ASYMMETRIC TECHNIQUES	204
8.1	Introduction	204
8.1.1	Insecurity of Textbook Encryption Algorithms	206
8.1.2	Chapter Outline	207
8.2	The Diffie-Hellman Key Exchange Protocol	207
8.2.1	The Man-in-the-Middle Attack	209
8.2.2	The Diffie-Hellman Problem and the Discrete Logarithm Problem	211
8.2.3	The Importance of Arbitrary Instances in Intractability Assumptions	214
8.3	The RSA Cryptosystem	215
8.3.1	Cryptanalysis Against Public-key Cryptosystems	217
8.3.2	Insecurity of the Textbook RSA	219

8.3.3	The Integer Factorization Problem	223
8.4	The Rabin Cryptosystem	225
8.4.1	Insecurity of the Textbook Rabin	227
8.5	The ElGamal Cryptosystem	229
8.5.1	Insecurity of the Textbook ElGamal	231
8.6	Need for Stronger Security Notions for Public-key Cryptosystems	232
8.7	Combination of Asymmetric and Symmetric Cryptography	233
8.8	Bit Security of the Basic Public-key Cryptographic Functions	235
8.8.1	The RSA Bit	236
8.8.2	The Rabin Bit and the Strength of the Blum-Blum-Shub Pseudo Random Bits Generator	239
8.8.3	The ElGamal Bit	240
8.8.4	The Discrete Logarithm Bit	241
8.9	Key Channel Establishment for Public-key Cryptosystems	244
8.10	Chapter Summary	244
9	DATA INTEGRITY TECHNIQUES	246
9.1	Introduction	246
9.1.1	Chapter Outline	248
9.2	Symmetric Techniques	248
9.2.1	Cryptographic Hash Functions	248
9.2.2	MAC Based on a Keyed Hash Function	252
9.2.3	MAC Based on a Block Cipher Encryption Algorithm	253
9.3	Asymmetric Techniques I: Digital Signatures (Textbook Versions)	253
9.3.1	The RSA Signature	256
9.3.2	The Rabin Signature	258
9.3.3	The ElGamal Signature	259
9.3.4	ElGamal-like Signature Schemes	261
9.3.5	Security of Digital Signature Schemes	264
9.4	Asymmetric Techniques II: Data Integrity Without Source Identification	266
9.5	Chapter Summary	269

IV Authentication	271
10 AUTHENTICATION PROTOCOLS I — PRINCIPLES	273
10.1 Introduction	273
10.1.1 Chapter Outline	274
10.2 Authentication and Refined Notions	275
10.2.1 Data-Origin Authentication	275
10.2.2 Entity Authentication	277
10.2.3 Authenticated Key Establishment	278
10.2.4 Attacks on Authentication Protocols	278
10.3 Convention	279
10.4 Basic Authentication Techniques	281
10.4.1 Message Freshness and Principal Liveness	281
10.4.2 Mutual Authentication	289
10.4.3 Authentication Involving Trusted Third Party	291
10.5 Password-based Authentication Techniques	295
10.5.1 Needham's Password Protocol	295
10.5.2 A One-time Password Scheme (and a Flawed Modification)	297
10.5.3 Add Your Own Salt: Encrypted Key Exchange (EKE)	299
10.6 Authenticated Key Exchange Based on Asymmetric Cryptography	302
10.6.1 The Station-to-station Protocol	303
10.6.2 A Flaw in a Simplified STS Protocol	306
10.6.3 A Minor Flaw of the STS Protocol	308
10.7 Typical Attacks on Authentication Protocols	311
10.7.1 Message Replay Attack	312
10.7.2 Man-in-the-Middle Attack	313
10.7.3 Parallel Session Attack	314
10.7.4 Reflection Attack	317
10.7.5 Interleaving Attack	319
10.7.6 Attack due to Type Flaw	320
10.7.7 Attack due to Name Omission	321
10.7.8 Attack due to Misuse of Cryptographic Services	322
10.8 A Brief Literature Note	326

10.9 Chapter Summary	327
11 AUTHENTICATION PROTOCOLS FOR THE REAL WORLD	328
11.1 Introduction	328
11.1.1 Chapter Outline	329
11.2 Authentication Protocols for Internet Security	330
11.2.1 Communications at the Internet Protocol Layer	330
11.2.2 Internet Protocol Security (IPSec)	331
11.2.3 The Internet Key Exchange (IKE) Protocol	333
11.2.4 A Plausible Deniability Feature in IKE	339
11.2.5 Critiques on IPSec and IKE	341
11.3 The Secure Shell (SSH) Remote Login Protocol	342
11.3.1 The SSH Architecture	342
11.3.2 The SSH Transport Layer Protocol	343
11.3.3 The SSH Strategy	347
11.3.4 A Caveat	347
11.4 The Kerberos Protocol and its Realization in Windows 2000	347
11.4.1 A Single-sign-on Architecture	349
11.4.2 The Kerberos Exchanges	351
11.4.3 Caveats	353
11.5 SSL and TLS	354
11.5.1 TLS Architecture Overview	354
11.5.2 TLS Handshake Protocol	355
11.5.3 A Typical Run of the TLS Handshake Protocol	358
11.6 Chapter Summary	359
12 AUTHENTICATION FRAMEWORK FOR PUBLIC-KEY CRYPTOGRAPHY	360
12.1 Introduction	360
12.1.1 Chapter Outline	361
12.2 Directory-Based Authentication Framework	361
12.2.1 Certificate Issuance	363
12.2.2 Certificate Revocation	363

12.2.3 Examples of Public-key Authentication Framework	364
12.3 Non-Directory Based Authentication Framework	365
12.3.1 Shamir's ID-Based Signature Scheme	367
12.3.2 What Exactly does ID-based Cryptography Offer?	370
12.3.3 ID-Based Encryption from Pairings on Elliptic Curves	371
12.3.4 Non-interaction Property: Authentication Without Key Channel	380
12.3.5 Two Open Questions for Identity-based Public-key Cryptography	380
12.4 Chapter Summary	381

V FORMALISM APPROACHES TO SECURITY ESTABLISHMENT **382**

13 FORMALISM FOR RIGOROUSNESS - SECURITY DEFINITIONS AND FORMALLY PROVABLE SECURITY	384
13.1 Introduction	384
13.1.1 Chapter Outline	386
13.2 A Formal Treatment for Security	386
13.3 Semantic Security — the Debut of Provable Security	390
13.3.1 The SRA Mental Poker Protocol	390
13.3.2 A Security Analysis Based on a Rough Notion of Security	391
13.3.3 Probabilistic Encryption of Goldwasser and Micali	394
13.3.4 The Security of the GM Cryptosystem	396
13.3.5 A Semantically Secure Version of the ElGamal Cryptosystem	398
13.3.6 Semantically Secure Cryptosystems Based on Rabin Bits	401
13.4 Inadequacy of Semantic Security	402
13.5 Beyond Semantic Security	404
13.5.1 Security Against Chosen Ciphertext Attack	405
13.5.2 Security Against Adaptively Chosen Ciphertext Attack	409
13.5.3 Non-Malleable Cryptography	412
13.5.4 Relations Between Indistinguishability and Non-Malleability	415
13.6 Strong Security Notion for Digital Signatures	420

13.7 Chapter Summary	420
14 PROVABLY SECURE AND PRACTICALLY EFFICIENT PUBLIC-KEY CRYPTOSYSTEMS	421
14.1 Introduction	421
14.1.1 Chapter Outline	422
14.2 The Optimal Asymmetric Encryption Padding (OAEP)	423
14.2.1 Random Oracles	425
14.2.2 Random Oracle Model for Security Proof	426
14.2.3 RSA-OAEP	428
14.2.4 A Twist in the Security Proof for RSA-OAEP	428
14.2.5 Tightness of “Reduction to Contradiction” for RSA-OAEP	440
14.2.6 A Critique on the Random Oracle Model	441
14.3 The Cramer-Shoup Public-key Cryptosystem	442
14.3.1 Provable Security Under Standard Intractability Assumptions	442
14.3.2 The Scheme	443
14.3.3 Proof of Security	446
14.4 An Overview of Provably Secure Hybrid Cryptosystems	455
14.5 Literature Notes on Practical and Provably Secure Public-key Cryptosystems	457
14.6 Chapter Summary	459
15 FORMAL METHODS FOR AUTHENTICATION PROTOCOLS ANALYSIS	460
15.1 Introduction	460
15.1.1 Chapter Outline	461
15.2 Toward Formal Specification of Authentication Protocols	462
15.2.1 Imprecision of Encryption-decryption Approach for Authentication	462
15.2.2 A Refined Specification for Authentication Protocols	467
15.2.3 Examples of Refined Specification for Authentication Protocols	468
15.3 A Computational View of Correct Protocols — the Bellare-Rogaway Model	472
15.3.1 Formal Modelling of the Behavior of Principals	474

15.3.2	The Goal of Mutual Authentication: Matching Conversations	477
15.3.3	Protocol <i>MAP1</i> and its Proof of Security	478
15.3.4	Further Work in Computational Model for Protocols Correctness	480
15.3.5	Discussion	481
15.4	A Symbolic Manipulation View of Correct Protocols	481
15.4.1	Theorem Proving	482
15.4.2	A Logic of Authentication	483
15.5	Formal Analysis Techniques: Finite-State System Approach	485
15.5.1	Model Checking	486
15.5.2	The NRL Protocol Analyzer	489
15.5.3	The CSP Approach	491
15.6	Reconciling Two Views of Formal Reasoning about Security	497
15.7	Chapter Summary	498
	BIBLIOGRAPHY	499

List of Figures

1.1	Protocol “Coin Tossing Over Telephone”	5
2.1	A Cryptographic System	22
2.2	Protocol “From Alice To Bob”	28
2.3	Protocol “Session Key From Trent”	30
2.4	An Attack on Protocol “Session Key From Trent”	31
2.5	Protocol “Message Authentication”	35
2.6	Protocol “Challenge Response”	39
2.7	An Attack on the Needham-Schroeder Symmetric-key Authentication Protocol	41
2.8	Needham-Schroeder Public-key Authentication Protocol	44
2.9	An Attack on the Needham-Schroeder Public-key Authentication Protocol	46
4.1	A Turing Machine	70
4.2	Euclid’s Algorithm for Greatest Common Divisor	76
4.3	Extended Euclid’s Algorithm	79
4.4	An Algorithm for Modular Exponentiation	84
4.5	A Zero-sided-error (\mathcal{ZPP}) Algorithm	89
4.6	A One-sided-error (Monte-Carlo) Algorithm	91
4.7	A One-sided-error (Las-Vegas) Algorithm	94
4.8	A Two-sided-error (\mathcal{BPP}) Algorithm: Quantum Key Distribution Protocol	97
4.9	A Probabilistic Algorithm for Prime Number Generation	100

4.10	All Possible Moves of a Nondeterministic Turing Machine	103
5.1	An Algorithm for Finding Random Primitive Root Modulo Prime	141
6.1	An Algorithm for Computing Chinese Remainder	152
6.2	An Algorithm for Computing Legendre/Jacobi Symbol	161
6.3	An Algorithm for Computing Square Root Modulo $p \equiv 3, 5, 7 \pmod{8}$	164
6.4	An Algorithm for Computing Square Root Modulo Prime	165
6.5	An Algorithm for Computing Square Roots Modulo Composite	166
7.1	Cryptographic Systems	174
7.2	A Zero-knowledge Proof Protocol Using Shift Ciphers	182
7.3	Feistel Cipher (One Round)	186
7.4	The Cipher Block Chaining Mode of Operation	198
7.5	The Cipher Feedback Mode of Operation	200
7.6	The Output Feedback Mode of Operation	201
8.1	The Diffie-Hellman Key Exchange Protocol	208
8.2	Man-in-the-Middle Attack on the Diffie-Hellman Key Exchange Protocol	210
8.3	The RSA Cryptosystem	216
8.4	The Rabin Cryptosystem	226
8.5	The ElGamal Cryptosystem	230
8.6	An algorithm for binary searching RSA plaintext using a “Parity Oracle”	238
8.7	An algorithm for extracting discrete logarithm using a “Parity Oracle”	242
8.8	An algorithm for extracting discrete logarithm using a “Half-order Oracle”	243
9.1	Data Integrity Systems	247
9.2	The RSA Signature Scheme	256
9.3	The Rabin Signature Scheme	258
9.4	The ElGamal Signature Scheme	260
9.5	The Schnorr Signature Scheme	263
9.6	The Digital Signature Standard	265

9.7 The RSA-OAEP scheme	268
10.1 ISO Public Key Three-Pass Mutual Authentication Protocol	290
10.2 The Wiener Attack on ISO Public Key Three-Pass Mutual Authentication Protocol	291
10.3 The Woo-Lam Protocol	294
10.4 Needham's Password Authentication Protocol	296
10.5 The S/KEY Protocol	298
10.6 The Encrypted Key Exchange (EKE) Protocol	301
10.7 The Station-to-Station (STS) Protocol	304
10.8 Flawed Authentication-only STS Protocol	306
10.9 An Attack on the Authentication-only STS Protocol	307
10.10Lowe's Attack on the STS Protocol (a Minor Flaw)	309
10.11An Attack on the S/KEY Protocol	314
10.12A Parallel-Session Attack on the Woo-Lam Protocol	316
10.13A Reflection Attack on a "Fixed" Version of the Woo-Lam Protocol	318
10.14A Minor Variation of the Otway-Rees Protocol	323
10.15An Attack on a Variant Otway-Rees Protocol	325
11.1 An IP Packet	331
11.2 Signature Based IKE Phase 1 Main Mode	335
11.3 Authentication Failure in Signature Based IKE Phase 1 Main Mode	337
11.4 Kerberos Exchanges	350
11.5 A Typical Run of the TLS Handshake Protocol	358
12.1 Shamir's Identity-based Signature Scheme	369
12.2 The Identity-based Cryptosystem of Boneh and Franklin	377
13.1 Chosen plaintext attack	387
13.2 The SRA mental poker protocol	392
13.3 The probabilistic cryptosystem of Goldwasser and Micali	395
13.4 "Lunchtime attack" (chosen ciphertext attack)	406
13.5 "Small-hours attack" (adaptively chosen ciphertext attack)	410
13.6 Summary of the indistinguishable attack games	412

13.7 Malleability attack in chosen plaintext mode	413
13.8 Reduction from an NM-attack to an IND-attack	417
13.9 Reduction from IND-CCA2 to NM-CCA2	419
13.10 Relations among security notions for public-key cryptosystems	420
14.1 Optimal asymmetric encryption padding (OAEP)	423
14.2 OAEP as two-round Feistel cipher	424
14.3 Reduction from inversion of a one-way trapdoor function f to an attack on the f -OAEP scheme	430
14.4 The Cramer-Shoup public-key cryptosystem	444
14.5 An algorithm computing product of exponentiations at the cost of single exponentiation	447
14.6 Reduction from the DDH problem to an attack on the Cramer-Shoup cryptosystem	450
15.1 The Needham-Schroeder Symmetric-key Authentication Protocol in Refined Specification	468
15.2 The Woo-Lam Protocol in Refined Specification	470
15.3 The Needham-Schroeder Public-key Authentication Protocol	471
15.4 The Needham-Schroeder Public-key Authentication Protocol in Refined Specification	472
15.5 Another Refined Specification of the Needham-Schroeder Public-key Authentication Protocol	473
15.6 Protocol <i>MAP1</i>	479
15.7 The CSP Language	492
15.8 The Entailment Axioms	496

List of Tables

4.1	The operation of machine DIV3	72
4.2	Bitwise time complexities of the basic modular arithmetic operations	86

PREFACE

Our society has entered an era where business is conducted and services are offered over open computer and communications networks. In addition to email and the World Wide Web, common tools used in many electronic business and services, a few examples of business and services that can or will be conducted and offered electronically are:

Banking, Bill payment, Home shopping, Stock trading, Auctions, Taxation, Gambling, Micropayment (e.g., pay-per-downloading), On-line access of medical records, Networked computing, Secure data archival and retrieval, Digital libraries, Certified delivery of documents or electronic goods, Time-stamping, Notarization, Voting, Advertising, Ticket booking, Interactive games, . . .

and more can be imagined. Electronic form of business and services can be conducted and offered on 24 hour basis and the interactions between people from any corners of the world. Fascinating business and services like these are only possible if communications on open networks can be securely achieved. A common solution for securing communications in open networks is to employ cryptography. Cryptographic systems and protocols will be designed and widely deployed. These systems use cryptography (such as encryption algorithms, digital signatures, hash functions, etc.) to provide various security services that applications require.

With an increasingly large demand for safeguarding numerous kinds of electronic business and fancy services^a we are witnessing an even larger demand for the designers to design and realize cryptographic systems and protocols. As a result, many “geeks” who are engineers oriented to application problems and may have no proper training in cryptography and system security, have and will become such designers. This is in spite of the fact that designing cryptographic systems and protocols is a difficult job even for an expert cryptographer.

^aGartnerGroup fore-casted that a total electronic business revenues for business to business (B2B) and business to consumer (B2C) in the European Union will reach a projected US dollar \$2.6 trillion in 2004 (with probability 0.7) which is a 28-fold increase from the level of 2000 [AD00].

In past few years the author, a consultant on security and cryptographic systems, has witnessed many times so-called “textbook systems”: security systems and cryptographic protocols which have been designed by “geek” engineers as results of applying cryptographic algorithms and schemes in ways they are introduced in textbooks. Direct encryption of a password (a secret number of a small magnitude) under a basic public-key encryption algorithm (e.g., RSA) is a common example of a “textbook system”. Most such “textbook systems” have been proposed for real applications, however, their designers did not realize that the textbook methods of using cryptographic algorithms are not secure. Consequently, their designs are usually not suitable for real applications.

Motivated by the high demand for engineers who will be using cryptography to solve security problems in real world applications, the author has intended this book to be a textbook on non-textbook cryptography. This book endeavours to

- introduce a range of cryptographic algorithms, schemes and protocols with emphasized explanations and elaborations on their non-textbook versions which are more suitable for real world applications;
- introduce formalism approaches and methodologies to the establishment of security for cryptographic systems and protocols;
- warn against pitfalls, summarize lessons learnt, and provide guidelines and principles for design, specification and analysis of cryptographic systems and protocols.

Overview of The Book

The first two chapters form the introductory material for this book.

Chapter 1 begins with a simple example of applying cryptography to solve a small “security problem” which has a little bit of subtlety due to the problem’s communication nature. The purpose of our example is to provide a clear demonstration of the effectiveness and practicality of using cryptography in communication protocols for solving security problems. The rest of the first chapter consists of discussions on the criteria for the quality of cryptographic systems and protocols. The discussions provide an introduction to a culture held by the communities for research and development of cryptography and information security.

Our cultural introduction changes an aspect in Chapter 2. There we shall introduce a series of simple but flawed cryptographic protocols. These protocols will play two roles. First, they should provide us with an initial familiarity with some basic agreements and languages in the areas of study. These include some basic terminologies, meanings behind them, the naming convention for several famous protocol participants, and the behavior of a special role, the attacker, who is always not welcome however without whom we do not need this book. Secondly, as all

these protocols are flawed, we intend to let the reader, in particular who is new to the areas of cryptography and information security, quickly become used to a fact of life: there is a constant wrestling between an endeavour for safeguarding information and communications and a temptation for cracking them.

The second part of this book is a collection of mathematical material which provides the theoretic foundations for the areas. This includes standard notation, probability and the basic information theory (Chapter 3), computational complexity (Chapter 4), algebraic foundations (Chapter 5) and number theory (Chapter 6).

These four chapters can be used as a self-contained mathematical reference. In the rest of the book whenever we meet non-trivial mathematical problems we will be able to refer to the precise places in these four chapters to obtain supporting facts and/or foundations. Therefore our way of including the mathematical material in this book will help the reader to conduct an active and interactive way to learn the mathematical foundations for modern cryptography.

Part III contains three chapters which introduce the most basic cryptographic techniques for confidentiality and data integrity. Chapter 7 introduces symmetric encryption techniques, Chapter 8 introduces asymmetric encryption techniques, Chapter 9 introduces basic techniques for data integrity.

The basic cryptographic algorithms and schemes to be introduced in Part III, especially the encryption algorithms, can be considered as “textbook versions” since they can be found in many textbooks on cryptography. In Part III we shall frequently expose the weakness of these “textbook version” algorithms and schemes, even though we will not, in fact cannot, fix these weakness in this part. However, this book will not stop at the “textbook version” level of introduction to cryptography. “Fit for application versions” of the encryption algorithms and data-integrity mechanisms will be introduced in later chapters, and most of them are results of enhancing the “textbook versions”.

Part IV introduces various authentication protocol techniques. This part has three chapters. In Chapter 10 we study authentication protocols on their basic working principles, examine typical errors in authentication protocols and investigate causes. In Chapter 11 we case-study several important authentication protocols proposed for use in the real world applications (most of them are already in wide use). In Chapter 12 we introduce the authentication framework for public-key infrastructure.

Because of the application importance of the subject, we feature the authentication part of this book with a comprehensive inclusion and elaboration of authentication protocols which are “fit for application versions”: they are industrial standards (IKE, SSH, Kerberos and SSL) and most of them are already in service in our everyday lives. Also in this part we shall witness the extreme error-prone nature of authentication protocols. For this reason, this part is not an end of the story in this book on authentication protocols. We will return to this important

topic in a larger chapter on formal analysis techniques for authentication protocols.

Part V is a formalism treatment of cryptography and security. This part contains three chapters. Chapter 13 introduces the formal definitions of security notions for public-key cryptography. This chapter provides us with necessary foundations, methods and tools to introduce and explain the “non-textbook versions” of the basic public-key encryption algorithms which we have learnt in Chapter 8. Several important “non-textbook versions” of public-key cryptosystems will be introduced and explained in Chapter 14. These cryptosystems are well-known and practical for they are provably secure under the security definitions given in Chapter 13. In Chapter 15 we will return to the topic of authentication protocols: there we shall introduce various formalism analysis techniques for authentication protocols correctness.

In Part VI we study cryptographic protocols. Zero-knowledge protocols, electronic commerce protocols, and protocols for realizing various fancy services such as anonymity, invisibility, deniability, fairness, auditability, verifiability, ...

Part I

INTRODUCTION

The first part of the book consists of two introductory chapters. They introduce us to some most basic concepts in cryptography and information security, to the setting of the environment in which we communicate sensitive information, to several well known figures who act in that environment and to the standard *modus operandi* of some of them, and to the culture of the communities which research and develop technologies for protecting sensitive information.

BEGINNING WITH A SIMPLE COMMUNICATION GAME...

We begin this book with a simple example of applying **cryptography** to solve a simple problem. This example of cryptographic application serves three purposes from which we will unfold the topics of this book:

- to provide an initial demonstration on the effectiveness and practicality of using cryptography for solving subtle problems in applications;
- to suggest an initial hint on the foundation of cryptography; and
- to begin our process of establishing a required mindset for conducting the development of cryptographic systems for information security.

To begin with, we shall pose a trivially simple problem and then solve it with an equally simple solution. The solution is a two-party game which is very familiar to all of us. However, we will realize that our simple game becomes troublesome soon when our game-playing parties are physically remote from each other. The physical separation of the game-playing parties eliminates the bases for the game to be played fairly. The trouble is then, the game-playing parties cannot trust the other side to play the game fairly.

The need for a fair playing of that game for remote players will “inspire” us to strengthen our simple game by protecting it with a shield of armor. Our strengthening method follows the long established idea for protecting communications over open networks: hiding information using cryptography.

After having applied cryptography and reached a quality solution to our first security problem, we shall conduct a series of discussions on the quality criteria for cryptographic systems (§1.2). The discussions will serve a background and cultural introduction to the areas in which we research and develop technologies for protecting sensitive information.

1.1 A Communication Game

Here is a simple problem. Two friends, Alice and Bob^a want to spend an evening out together, but they cannot decide whether to go to the cinema or the opera. Nevertheless, they reach an agreement to let a coin decide: playing a coin tossing game which is very familiar to all of us.

Alice holds a coin and says to Bob: “You pick a side then I will toss the coin.” Bob does so and then Alice tosses the coin in the air. Then they both look which side of the coin landed on top. If Bob’s choice is on top, Bob may decide where they go; if the other side of the coin lands on top, Alice makes the decision.

In the study of communication procedures, a multi-party-played game like this one can be given a “scientific sounding” name: protocol. A protocol is a well specified procedure running among a plural number of participating entities. We should note the importance of the plurality of the game participants; if a procedure is executed entirely by one entity only then it is a procedure and cannot be called a protocol.

1.1.1 Our First Application of Cryptography

Now imagine that the two friends are trying to run this protocol over the telephone. Alice offers Bob: “You pick a side. Then I will toss the coin and tell you whether or not you have won.” Of course Bob will not agree, because he cannot verify the outcome of the coin toss.

However we can add a little bit of cryptography to this protocol and turn it into a version workable over the phone. The result will become a cryptographic protocol, our first cryptographic protocol in this book! For the time being, let us just consider our “cryptography” to be a mathematical function $f(x)$ which maps over the integers and has the following “magic property”:

Property 1.1: Magic Function f

For every integer x , it is easy to compute $f(x)$ from x while given any value $f(x)$ it is impossible to find any non-trivial information about a pre-image x , e.g., whether x is an odd or even number.

Since for now we view this quality a magic one, it is safe for us to use words such as “easy” and “impossible” without precise quantification. We shall provide formal definitions for these terms and see many realizations this “magic function” in this book.

Suppose that the two friends have agreed on the magic function f . Suppose also that they have agreed that, e.g., an even number represents **HEADS** and an

^aThey are the most well-known figures in the area of cryptography, cryptographic protocols and information security; they will appear in most of the cryptographic protocols in this book.

Protocol 1.1: “Coin Tossing Over Telephone”

PREMISE

Alice and Bob have agreed function f with the “magic property” in Property 1.1.

1. Alice picks a random integer x and computes $f(x)$;
she reads $f(x)$ to Bob over the phone;
2. Bob tells Alice his guess of x as even or odd;
3. Alice reads x to Bob;
4. Bob verifies $f(x)$ and sees the correctness/incorrectness of his guess.

Figure 1.1.

odd number represents TAILS. Now they are ready to run our first cryptographic protocol, Protocol 1.1, over the phone.

It is not difficult to argue that Protocol “Coin Tossing Over Telephone” works quite well over the telephone. The following is a rudimentary “security analysis”.

1.1.1.1 A rudimentary security analysis

First, from the “magic property” of f , Alice is unable to find two different numbers: x and y , one is odd and the other even (this can be expressed as $x \neq y \pmod{2}$) such that $f(x) = f(y)$. To find such a pair is called to find a collision (x, y) for f . Clearly, since given any x , it is easy to compute $f(x)$, ease of finding a collision (x, y) means ease of finding a pre-image y from $f(x)$, and this contradicts the “magic property” of f . Thus, once having read the value $f(x)$ to Bob over the phone (Step 1), Alice has committed to her choice of x and cannot change her mind. This is where Alice has completed her coin flipping.

Secondly, again due to the “magic property” of f , given the value $f(x)$, Bob cannot determine whether the pre-image used by Alice is odd or even and so has to place his guess (Step 2). At this point, Alice can convince Bob whether he has guessed right or wrong by revealing her pre-image x (Step 3). Indeed, Bob should be convinced if his own evaluation of $f(x)$ (in Step 4) matches the value told by

Alice in Step 1 and if he believes that the “magic property” of the agreed function do hold. Also, the coin-flipping is fair if x is taken from an adequately large space so Bob could not have a guessing advantage, that is, some strategy that gives him greater than 50-50 chance of winning.

1.1.2 An Initial Hint on the Foundation of Cryptography

Although our first protocol is very simple, it indeed qualifies as a **cryptographic protocol** because the “magic property” the protocol uses is a fundamental one and is widely regarded as the foundation of cryptography: **one-way-ness**.

From our rudimentary security analysis for Protocol 1.1 we can claim that the existence of one-way function implies a possibility for secure selection of recreation venue. The following is a reasonable generalization of this claim:

The existence of one-way function implies the existence of secure cryptographic system.

It is now well understood that the converse of this claim is also true:

The existence of secure cryptographic system implies the existence of one-way function.

It is widely believed that one-way function does exist. Therefore we are optimistic on the security of our information. Our optimism is often confirmed by our everyday experience: many processes in our world, mathematical or otherwise, has a one-way property. Consider the following phenomenon in physics (though not an extremely precise analogy for mathematics): it is an easy process for a glass to fall on floor and break into pieces while dispersing a certain amount of energy (e.g., heat, sound or even some dim light) into the surrounding environment; the reverse process: recollecting the dispersed energy and using it to reintegrate the broken pieces back into a whole glass must be very hard a problem if not impossible.

In Chapter 4 we shall see a class of mathematical functions which provide the one-way property for modern cryptography.

1.1.3 Security: More Than a Bless from Mathematical Properties

We have just claimed that information security requires certain mathematical properties. Moreover, we have further made an optimistic claim in the converse direction: mathematical properties implies (i.e., guarantees) information security.

However, in reality, the latter claim is not unconditionally true! Security in the real world applications depends on many real world issues. Let us explain this by continuing using our first protocol example.

We should point out that many important issues have not been considered in our rudimentary security analysis for Protocol 1.1. In fact, Protocol 1.1 itself is a much simplified specification. It has omitted some details which are important to the security services that the protocol is designed to offer. The omission has prevented us from asking several questions.

For instance, we may ask: has Alice really been forced to stick to her choice of x ? Likewise, has Bob really been forced to stick to his even-odd guess of x ? By “forced”, we mean whether voice over telephone is sufficient for guaranteeing the strong mathematical property to take effect? We may also ask whether Alice has a good random number generator for her to acquire the random number x . This quality can be crucially important in a more serious application which requires to make a fair decision.

All these details have been omitted from this simplified protocol specification and therefore they become hidden assumptions (more on this later). In fact, if this protocol is used for making a more serious decision, it should include some *explicit* instructions. For example, both participants may consider to record the other party’s voice when the value $f(x)$ and the even/odd guess are pronounced over the phone, and replay the records in case of dispute.

Often cryptographic systems and protocols are specified with simplifications similar to the case in Protocol “Coin Tossing Over Telephone”. Simplifications can help to obtain presentation clarity, especially when some agreement may be thought of as obvious. But sometimes a hidden agreement or assumption may be subtle and can be exploited to result in a surprising consequence. This is somewhat ironical to the “presentation clarity” which is originally intended for by omitting some details. A violation of an assumption of a security system may allow an attack to be exploited and the consequence can be the nullification of an intended service. It is particularly difficult to notice a violation of a hidden assumption. In §1.2.5 we shall provide a discussion on the importance of explicit design and specification of cryptographic systems.

An important theme of this book is to explain that security of/for real world applications has many “non-textbook” contents which must be considered seriously.

1.1.4 Modern Role of Cryptography: Ensuring Fair Play of Games

Cryptography was once a preserve of governments. Military and diplomatic organizations used it to keep messages secret. Nowadays however, cryptography has a modernized role in addition to keeping secrecy: ensuring fair play of “games” by a much enlarged population of “game players”. That is why we have chosen to begin this book cryptography with a communication game.

Deciding on a recreation venue may not be seen as a serious business, and so doing it via flipping a coin over the phone is just playing a small communication game for fun. However, there are many communications “games” which need to be

taken much more seriously. With more and more business activities being conducted electronically over open communications networks, many cases of our communications involve various kinds of “game playing” (in the Preface of this book we have listed various business and services examples which can be conducted or offered electronically over open networks; all of them involve some interactive actions of the participants by following a set of rules, which can be viewed as “playing communication games”). These “games” can be very important!

In general, the “players” of such “games” are physically distant from each other and they communicate over open networks which are notorious for lack of security. The physical distance combined with the lack of security may help and/or encourage some of the “game players” (some of them can even be uninvited) to try to defeat the rule of the game in some clever way. The intention for defeating the rule of the game is to try to gain some un-entitled advantage, such as causing disclosure of confidential information, modification of data without detection, forgery of false evidence, repudiation of an obligation, damage of accountability, reduction or nullification of availability, and so on. The importance of our modern communications in business and in the conduct of commerce (and also in various other aspects such as mission of companies, personal privacy, military actions and state affairs) means that no un-entitled advantage should be easily gained by a player who does not follow the rule of the game.

In our simple example (Protocol “Coin Tossing Over Telephone”) we have witnessed the process whereby an easy-to-sabotage communication game evolves to a cryptographic protocol and thereby offers the security service as desired. Our example demonstrates the effectiveness of cryptography in maintaining the order of “game playing”. Indeed, the use of cryptography is an effective and the *only practical* way to ensure secure communications over open computers and communications networks. Cryptographic protocols are just communication procedures armored with the use of cryptography and thereby have protective functions designed to keep communications in good order. The endless need for fascinating communications in business activities coupled with another need for anticipating the ceaseless temptation of “breaking the rules of the games” have resulted in the existence of many cryptographic protocols, which form the subject matter of this book.

1.2 Criteria for Desirable Cryptographic Systems/Protocols

We should start by asking a fundamental question:

What is a good cryptographic system/protocol?

Undoubtedly this question is not easy to answer! One reason is that there are many answers to it depending on various meanings the word *good* may have. It is

a main task for this book to provide comprehensive answers to this fundamental question. However, here in this first chapter we should provide a few initial answers.

1.2.1 Stringency of Protection Tuned to Application Needs

Let us begin with considering our first cryptographic protocol we designed in §1.1.1.

We can say that Protocol “Coin Tossing Over Telephone” is good in the sense that it is conceptually very simple. Some readers who may already be familiar with many practical one-way hash functions, such as SHA-1 (see §9.2.1), might further consider that the function $f(x)$ is also easy to implement even in a pocket calculator. For example, SHA-1 has a 160-bit, or 20-byte (1 byte = 8 bits), output length; using the hexadecimal encoding scheme (see Example 5.14) such an output can be encoded into 40 characters and so it is just not too tedious for Alice (Bob) to read (jot down) 40 characters over the phone. Such an implementation should also be considered sufficiently secure for Alice and Bob to decide their recreation venue: if Alice wants to cheat, she faces a non-trivial difficulty in order to find $x \neq y \pmod{2}$ with $f(x) = f(y)$; likewise, Bob will also have to face a non-trivial difficulty, that is, given $f(x)$, to determine whether x is even or odd.

However, our judgement on the quality of Protocol “Coin Tossing Over Telephone” is made on the basis of the non-seriousness level of the game players’ demanding on the quality of the game. In many more serious applications (e.g., one which we shall discuss in §1.2.4), a fair coin-flipping primitive for cryptographic use will in general require much stronger one-way and commitment-binding properties than a practical one-way hash function can offer. We should point out that a function with the “magic property” (Property 1.1), if we take the words “easy” and “impossible” literally, is a *completely secure* one-way function. In fact, such a function is not easily implementable. Worse, even its very existence remains an open question (we shall discuss the condition for the existence of a one-way function in Chapter 4). Therefore, for a more serious application of fair coin-flipping, much more stringent cryptographic means is necessary and practical one-way hash functions won’t be considered good. On the other hand, for deciding a recreation venue, use of heavy-weight cryptography is clearly unnecessary or overkill.

We should point out that there are applications where a too strong protection will even prevent an intended security service from functioning properly. In a later chapter we will introduce a protocol for realizing micro-payment which works by making use of known deficiencies in some practical cryptographic primitives to its advantage. That payment scheme (called MicroMint [RS96]) exploits a reasonable assumption that only a resourceful service provider (e.g., a large bank) is able to prepare a large number of “collisions” under a practical one-way hash function, and do so economically. This is to say that the service provider can compute in advance k distinct numbers (x_1, x_2, \dots, x_k) satisfying

$$f(x_1) = f(x_2) = \dots = f(x_k).$$

(The numbers x_1, x_2, \dots, x_k , are called collision under the hash function f .) Since a pair of collision can be checked efficiently, they can be considered to have been issued by the resourceful service provider and hence can represent a certified value. The Data Encryption Standard (DES, see §7.5) was suggested to be a suitable algorithm for implementing such a one-way hash function ([RS96]) and so to achieve a relatively small output space (64 binary bits). Thus, unlike in the normal cryptographic use of one-way hash functions where a collision almost certainly constitutes a successful attack on the system (for example, in the case of Protocol “Coin Tossing Over Telephone”), here, collisions are used in order to enable a fascinating micro-payment service! Clearly, a strong one-way hash function with a significantly larger output space ($\gg 64$ bits, such as SHA-1 with 160 bits) will nullify this service even for a resourceful service provider (in §3.5 we will study the computational complexity for finding collisions under a hash function).

Although it is understandable that using heavy-weight cryptographic technologies in the design of security systems (for example, wrapping with layers of encryption, arbitrarily using digital signatures, calling for on-line services from a trusted third party or even from a large number of them) may provide a better feeling that a stronger security may have been achieved (it may also ease the design job), often this feeling only provides a false sense of assurance. Reaching the point of overkill with unnecessary armor is undesirable because in so doing it is more likely to require stronger security assumptions and to result in a more complex system. A complex system can also mean a difficulty for security analysis (hence a more likelihood to be error-some), a poorer performance as well as a higher cost overhead.

It is more interesting and a more challenging task to design cryptographic or security systems which use only necessary technologies while achieving adequate security protection. This is an important element for cryptographic and security systems to qualify *good*.

1.2.2 Confidence in Security Based on Established “Pedigree”

How can we be confident that a cryptographic algorithm or a protocol is secure? Is it valid to say that an algorithm is secure because nobody has broken it? The answer is, unfortunately, *no*. In general, what we can say about an unbroken algorithm is merely that we do not know how to break it yet. Because in cryptography, the meaning of a broken algorithm sometimes has a quantitative measurement, if such a measurement is missing from an unbroken algorithm, then we cannot even assert that the unbroken algorithm is more secure than a known broken one.

Nevertheless, there are a few exceptions.

In most cases, the task of breaking a cryptographic primitive or an algorithm is to solve some mathematical problem, such as to find the solution to an equation or to invert a function, which is considered to be “hard” or “intractable”. A formal definition for “hard” or “intractable” will be given in Chapter 4. Here we can

informally, yet safely, say that a mathematical problem is intractable if it cannot be solved within a reasonable length of time.

There are a number of well-known intractable problems that have been frequently used as standard ingredients in modern cryptography, in particular, in public-key or asymmetrical cryptographic systems (see §8.2 — §8.6). These include the integer factorization problem, the discrete logarithm problem, the Diffie-Hellman problem, and a few associated problems (we will define and discuss these problems in Chapter 8). These problems can be referred to as established “pedigree” problems because they have sustained a long history of study by generations of mathematicians and as a result, they are trusted to be really hard with a high degree of confidence.

Today, a standard technique for establishing a high degree of confidence in the security of a cryptographic algorithm is to conduct a formal proof which demonstrates that an attack on the algorithm can lead to a solution to one of the accepted “pedigree” hard problems. Such a proof is an efficient mathematical transformation, or a sequence of such transformations, leading from an attack on an algorithm to a solution to a hard problem. Such an efficient transformation is called a reduction which “reduces” an attack to a solution to a hard problem. Since we are highly confident that the resultant solution to the hard problem should not exist, we will have a measurable high confidence that the alleged attack should not exist. This way of security proof is therefore named “reduction to contradiction”.

Formally provable security, in particular under a powerful attack model called an *adaptive attack*, forms an important criterion for cryptographic algorithms and protocols to be regarded *good*. We shall study this topic for many cryptographic algorithms and protocols in several chapters after Chapter 13.

1.2.3 Efficiency

When we say that a mathematical problem is efficient or is efficiently computable, we basically assert that the problem is solvable in time which can be measured by a polynomial in the size of the problem. (A formal definition for efficiency will be provided in Chapter 4.) This assertion is very useful in that it divides all the problems into two classes: computable and incomputable. This division plays a fundamental role in the foundation for modern cryptography: a complexity-theoretically based one. Clearly, a cryptographic algorithm must be designed such that it is computable on the one hand and so is usable by a legitimate user, but is incomputable on the other hand and so forms a difficult problem for a non-user or an attacker.

We should however note that this assertion for computability covers a vast span. If a problem’s computing time for a legitimate user is measured by a huge polynomial, then the “efficiency” can be impractical, i.e., can have no value for a practical use. Thus, an important criterion for a cryptographic algorithm to be *good* is that

it should be *practically efficient* for a legitimate user. Specifically, the polynomial that measures the time complexity for the user should be small (i.e., have a small degree, see Chapter 4 for the definition of the degree of a polynomial).

A protocol is not only an algorithm, it is also a communication procedure which involves transmitting a message over computer networks between different protocol participants. So a protocol has a further dimension for efficiency measurement: the number of communication interactions which are often called communication rounds. Usually a step of communication is regarded to be more expensive than a step of computation (typically an execution of a set of computer instructions, e.g. a multiplication of two numbers on a computing device). Therefore it is desirable that a cryptographic protocol should have few communication rounds. The standard efficiency criterion for declaring an algorithm as being efficient if its running time is bounded by a small polynomial in the size of the problem. If we apply this efficiency criterion to a protocol, then an efficient protocol should have its number of communication rounds bounded by a polynomial of an *extremely* small degree: a constant (degree 0) or at most a linear (degree 1) function. A protocol with communication rounds exceeding a linear function should not be regarded as practically efficient, that is, no *good* for any practical use.

In Chapter ?? we will see some zero-knowledge proof protocols which have communication rounds measured by non-linear polynomials. We should note that those protocols were not proposed for real applications; instead, they have importance in the theory of cryptography. In Chapters ?? we will see a tremendous research effort for designing practically efficient zero-knowledge proof protocols.

1.2.4 Use of Practical and Available Primitives and Services

A level of security which is good enough for one application need not be good enough for another. Again, let us use our coin-flipping protocol for example. In §1.2.1 we have agreed that, if implemented with the use of a practical one-way hash function, Protocol “Coin Tossing Over Telephone” is good enough for Alice and Bob to decide their recreation venue over the phone. However, in many cryptographic applications of a fair coin-flipping primitive, security services against cheating and/or for fairness are at much more stringent levels; in some applications the stringency must be in an absolute sense. For example, in Chapter ?? we will see some zero-knowledge proof protocols which need random bit-string input and such random input must be mutually trusted by the proving/verification parties, or else serious damages will occur to one or both parties. In such zero-knowledge proof protocols, if the two communication parties do not have access to, or do not trust, a third-party-based service for supplying random numbers (such a service is usually nicknamed “random numbers from the sky” to imply its impracticality) then they have to generate their mutually trusted random numbers, bit-by-bit via a fair coin-flipping protocol. Notice that here the need for the randomness to be generated in a bit-by-bit (i.e., via fair coin-flipping) manner is in order to satisfy certain requirements,

such as the mutual trustworthiness of the randomness or zero-knowledge-ness. In such a situation, a level of practically good (e.g., in the sense of using a practical hash function in Protocol “Coin Tossing Over Telephone”) is most likely to be inadequate.

A challenging task in applied research on cryptography and cryptographic protocols is to build high quality security services from *practical* and *available* cryptographic primitives. Once more, let us use a coin-tossing protocol to make this point clear. In Chapter ?? we will see another remote coin-flipping protocol proposed by Blum [Blu81]. Blum’s protocol employs a *practically secure* and *easily implementable* “one-way” function but achieves a high-quality security in a *very strong* secure fashion which can be expressed as follows. First, it is almost impossible for the party who tosses the coin (e.g., Alice) to succeed in cheating, i.e., to prepare $x \neq y$ with $f(x) = f(y)$ either in advance or during a protocol run. (It is provable that Alice’s successful cheating will lead to an efficient algorithm to factor a composite integer created by Bob, i.e., solve a “pedigree” hard problem.) Secondly, there is *absolutely no way* for Bob to have a guessing strategy biased away from the 50-50 chance. (This is in terms of a complete security.) Thus, Blum’s coin-flipping protocol is *particularly good* in the sense of having achieved a strong security while using only practical cryptographic primitives.

Several years after the discovery of public-key cryptography [DH76a], [DH76b], [RSA78], it became gradually apparent that several basic and best-known public-key encryption algorithms (now commonly referred to as “textbook” public-key algorithms) generally have two kinds of weakness: (i) leaking partial information about the message encrypted; (ii) extremely vulnerable to active attacks (see Chapter 13). Early approaches to a general fix for these weaknesses in “textbook versions” of cryptographic algorithms invariantly apply bit-by-bit style of encryption [GM84a], [NY90], [RS92] and even apply zero-knowledge proof technique at bit-by-bit level as a means to prevent active attacks [NY90], plus authentication framework [RS92]. These results, while valuable in the development of provably secure public-key encryption algorithms, are not suitable for most encryption applications since the need for zero-knowledge proof or for authentication framework are not practical for the case of encryption algorithms.

Since the successful initial work of [BR95a], provably secure public-key encryption algorithms begin to be constructed by enhancing the “textbook” public-key encryption algorithms with practical and popular primitives such as hash functions and pseudo-random number generators. These enhanced algorithms are practical since their efficiency are similar to the underlying “textbook” schemes. Due to this quality element, these enhanced algorithms become public-key encryption standards. We shall see two typical so-enhanced schemes in Chapter 14.

Designing protocols or security systems using available and popular techniques or primitives is also desirable in the sense that such techniques or algorithms are more likely to be secure as they have attracted a wider interest and have been under

public scrutiny longer.

1.2.5 Explicitness

In the late 1960's, software systems grew very large and complex. Computer programmers began to experience a crisis, the so-called "software crisis". Large and complex software systems were getting more and more error prone, and the cost of debugging a program became far in excess of the cost of the program design and development. Soon computer scientists discovered a few perpetrators who helped to setup the crisis which resulted from bad programming practices. Bad programming practices include:

- arbitrary use of the GOTO statement (jumping up and down seems to be very convenient);
- abundant use of global variables (causing uncontrolled change of their values, e.g., in an unexpected execution of a subroutine);
- the use of variables without declaration of their types (implicit types can be used in Fortran, so, for example, a real value may be truncated to an integer one without being noticed by the programmer);
- unstructured and unorganized large chunk of codes for many tasks (can be thousands of lines a piece);
- few commentary lines (since they don't execute!).

These were a few "convenient" things for a programmer to do, but had proven to be capable of causing great difficulties in program debugging, maintenance and further development. Software codes designed with these "convenient" features can be just too obscure to be comprehensible and maintained. Back then it was not uncommon a phenomenon that a programmer would not be able to understand a piece of code s/he had written merely a couple of months or even weeks ago.

Once the disastrous consequences resulting from the bad programming practices were being gradually understood, *Program Design Methodology* became a subject of study in which *being explicit* became an important principle for programming. Being explicit includes limiting the use of GOTO and global variables (better not to use them at all), explicit (via mandatory) type declaration for any variables which permits a compiler to check type flaws systematically and automatically, modularizing programming (dividing a large program in to many smaller parts, each for one task), and using abundant (as clear as possible) commentary material which are texts inside a program and documentation outside.

A security system (cryptographic algorithm or protocols) includes program parts implemented in software and/or hardware, and in the case of protocol, the program

parts run on a number of separate hosts (or a number of programs concurrently and interactively running on these hosts). The explicitness principle for software engineering applies to a security system's design by default (this is true in particular for protocols). However, because a security system is assumed to run in a hostile environment in which even a legitimate user may be malicious, a designer of such systems must also be explicit about many additional things. Here we list three important aspects to serve as general guidelines for security system designers and implementors. (In the rest of the book we will see many attacks on algorithms and protocols due to being implicit in design or specification of these systems.)

1. Be explicit about all assumptions needed

A security system operates by interacting with an environment and therefore it has a set of requirements which must be satisfied by that environment. These requirements are called assumptions (or premises) for a system to run. A violation of an assumption of a protocol may allow a possibility to exploit an attack on the system and the consequence can be the nullification of some intended services. It is particularly difficult to notice a violation of an assumption which has not been clearly specified (a hidden assumption). Therefore all assumptions of a security system should be made explicit.

For example, it is quite common that a protocol has an implicit assumption or expectation that a computer host upon which the protocol runs can supply good random numbers, but in reality few desktop machines or hand-held devices are capable of satisfying this assumption. A so-called low-entropy attack is applicable to protocols using a poor random source. A widely publicized attack on an early implementation of the Secure Sockets Layer (SSL) Protocol (an authentication protocol for World Wide Web browser and server, see §11.5) is a well-known example of the low-entropy attack [GW96].

Explicit identification and specification of assumptions can also help the analysis of complex systems. DeMillo et al (Chapter 4 of [DDD⁺83]), DeMillo and Merritt [DM83] suggest a two-step approach to cryptographic protocol design and analysis, which are listed below (after a modification by Moore [Moo88], [Moo92]):

- i) Identify **all** assumptions made in the protocol.
- ii) For each assumption in step (i), determine the effect on the security of the protocol if that assumption were violated.

2. Be explicit about exact security services to be offered

A cryptographic algorithm/protocol provide certain security services. Examples of some important security services include: confidentiality (a message cannot be comprehended by a non-recipient), authentication (a message can be recognized to confirm its integrity or its origin), non-repudiation (impossibility for one to deny a connection to a message), proof of knowledge (demonstration of an evidence without disclosing it), and commitment (e.g., a service

offered to our first cryptographic protocol “Coin Tossing Over Telephone” in which Alice is forced to stick to a string without being able to change).

When designing a cryptographic protocol, the designer should be very clear regarding exactly what services the protocol intends to serve and should explicitly specify them as well. The explicit identification and specification will not only help the designer to choose correct cryptographic primitives or algorithms, but also help an implementor to correctly implement the protocol. Often, an identification of services to the refinement level of the general services given in these examples is not adequate, and further refinement of them is necessary. Here are a few possible ways to further refine some of them:

Confidentiality \Rightarrow privacy, anonymity, invisibility, indistinguishability;

Authentication \Rightarrow data-origin, data-integrity, peer-entity;

Non-repudiation \Rightarrow message-issuance, message-receipt;

Proof of knowledge \Rightarrow knowledge possession, knowledge structure.

A mis-identification of services in a protocol design can cause misuse of cryptographic primitives, and the consequence can be a security flaw in the protocol. In Chapter 2 and Chapter 10 we will see disastrous examples of security flaws in authentication protocols due to mis-identification of security services between confidentiality and authentication.

There can be many more kinds of security services with more ad-hoc names (e.g., message freshness, non-malleability, forward secrecy, perfect-zero-knowledge, fairness, binding, deniability, receipt freeness, and so on). These may be considered as derivatives or further refinement from the general services that we have listed earlier (a derivative can be in terms of negation, e.g., deniability is a negative derivative from non-repudiation). Nevertheless, explicit identification of them is often necessary in order to avoid design flaws.

3. Be explicit about special cases in mathematics

As we have discussed in §1.2.2, some hard problems in computational complexity theory can provide a high confidence in the security of a cryptographic algorithm or protocol. However, often a hard problem has some special cases which are not hard at all. For example, we know that the problem of factorization of a large composite integer is in general very hard. However the factorization of a *large* composite integer $N = PQ$ where Q is the next prime number of a *large* prime number P is not a hard problem at all! One can do so efficiently by computing $\lfloor \sqrt{N} \rfloor$ ($\lfloor \cdot \rfloor$ is called the floor function and denotes the integer part of \cdot) and followed by a few trial-divisions around that number to pinpoint P and Q .

Usual algebraic structures upon which cryptographic algorithms work (such as groups or fields, to be defined in Chapter 5) contain special cases which produce easy problems. Elements of small multiplicative orders (also defined in Chapter 5) in a multiplicative group or a finite field provide such an example; an extreme case of this is when the base for the Diffie-Hellman key exchange protocol (see §??) is the unity element in these algebraic structures. Easy cases of elliptic curves, e.g., “supersingular curves” and “anomalous curves”, form another example. The discrete logarithm problem on “supersingular curves” can be reduced to the discrete logarithm problem on a finite field, known as the Menezes-Okamoto-Vanstone attack [MOV83] (see §12.3.3). An “anomalous curve” is one with the number of points on it being equal to the size of the underlying field, which allows a polynomial time solution to the discrete logarithm problem on the curve, known as the attack of Satoh-Araki [SA98], Semaev [Sem98] and Smart [Sma99].

An easy special case, if not understood by an algorithm/protocol designer and/or not clearly specified in an algorithm/protocol specification, may easily go into an implementation and can thus be exploited by an attacker. So an algorithm/protocol designer must be aware of the special cases in mathematics, and should explicitly specify the procedures for the implementor to eliminate such cases.

It is not difficult to list many more items for explicitness (for example, a key-management protocol should stipulate explicitly the key-management rules, such as separation of keys for different usages, and the procedures for proper key disposal, etc.). Due to the specific nature of these items we cannot list all of them here. However, explicitness as a general principle for cryptographic algorithm/protocol design and specification will be frequently raised in the rest of the book. In general, the more explicit that an algorithm/protocol is designed and specified, the easier it is for the algorithm/protocol to be analyzed; therefore the more likely it is for the algorithm/protocol to be correctly implemented, and the less likely it is for the algorithm/protocol to suffer an unexpected attack.

1.2.6 Openness

Cryptography was once a preserve of governments. Military and diplomatic organizations used it to keep messages secret. In those days, most cryptographic research was conducted behind closed doors; algorithms and protocols were secrets. Indeed, governments did, and they still do, have a valid point in keeping their cryptographic research activities secret. Let us imagine that a government’s agency publishes a cipher. We should only consider the case that the cipher published is provably secure; otherwise the publication can be too dangerous and may actually end up causing embarrassment to the government. Then other governments may use the provably secure cipher and consequently undermine the effectiveness of the code-breakers of

the government which published the cipher.

Nowadays, however, cryptographic mechanisms have been incorporated in a wide range of civilian systems (we have provided a non-exhaustive list of applications in the very beginning of this chapter). Cryptographic research for civilian use should take an open approach. Cryptographic algorithms do use secrets, but these secrets should be confined to the cryptographic keys or keying material (such as passwords or PINs); the algorithms themselves should be made public. Let's explore the reasons for this stipulation.

In any area of study, quality research depends on the open exchange of ideas via conference presentations and publications in scholarly journals. However, in the areas of cryptographic algorithms, protocols and security systems, open research is more than just a common means to acquire and advance knowledge. An important function of open research is public expert examination. Cryptographic algorithms, protocols and security systems have been notoriously error prone. Once a cryptographic research result is made public it can be examined by a large number of experts. Then the opportunity for finding errors (in design or maybe in security analysis) which may have been overlooked by the designers will be greatly increased. In contrast, if an algorithm is designed and developed in secret, then in order to keep the secret, only few, if any, experts can have access to and examine the details. As a result the chance for finding errors is decreased. A worse scenario can be that a designer may know an error and may exploit it secretly.

It is now an established principle that cryptographic algorithms, protocols, and security systems for civilian use must be made public, and must go through a lengthy public examination process. Peer review of a security system should be conducted by a hostile expert ([And94]).

1.3 Chapter Summary

In this chapter we began with an easy example of applied cryptography. The three purposes served by the example are:

- i) showing the effectiveness of cryptography in problem solving;
- ii) aiming for a fundamental understanding of cryptography, and
- iii) emphasizing the importance of non-textbook aspects of security.

They form the main topics to be developed in the rest of this book.

We then conducted a series of discussions which served the purpose for an initial background and cultural introduction to the areas of study. Our discussions in these directions are by no means of complete. Several other authors have also conducted extensive study on principles, guidelines and culture for the areas of cryptography

and information security. The following books form good material further reading: [Sch01], [Gol99] and [And01].

WRESTLING BETWEEN SAFEGUARD AND ATTACK

One reason for the existence of many cryptographic protocols is the consequence of a fact: It is very difficult to make cryptographic protocols right. Endless endeavours have been made to design correct protocols. Many new protocols were proposed as a result of fixing existing ones in which security flaws were discovered. A security flaw in a cryptographic protocol can always be described by an attack scenario in which some security services that the protocol purports to provide can be sabotaged by an attacker or by a number of them via their collusion. In the area of cryptographic protocols it is as if there is a permanent wrestling between protocol designers and attackers: A protocol is proposed, an attack is discovered, a fix follows, then another attack, and another fix, ...

In this chapter we shall demonstrate a series of examples of such a wrestling battle. We shall start from an artificial protocol which is made flawed deliberately. From that protocol we will go through a “fix, attack, fix again and attack again” process. Eventually we will reach two real protocols (all of the “fixes” prior to these two final results are artificial protocols). These two final results are not only real protocols, but also well-known ones for two reasons. First, these two protocols have played seminal roles both in applications and in underlying an important study on formal analysis of cryptographic protocols. Secondly, they contain security flaws which were only discovered *long* after their publication. (The two protocols were published in the same article; one flaw in one of them was found three years after the publication, and another flaw in the other protocol was exposed after another fourteen years passed!)

This chapter also serves a role of providing an introduction to material and

ideas that will enable us (in particular, readers who are new to the area of cryptographic protocols) to establish some common and important concepts, definitions and agreements in the area of study. These include some basic terminologies and the meanings behind them (a term appearing in the first time will be in **bold** form), and the naming convention for the protocol participants who we will be frequently meeting throughout the book. Also, the attacks on these flawed protocols will let us be getting familiar with some typical behaviour of a special role in our game play: the enemy, against whom we design cryptographic protocols.

2.1 Encryption

All protocols to be designed in this chapter will use **encryption**.

Encryption (sometimes called **encipherment**) is a process to transform a piece of information into an incomprehensible form. The input to the transformation is called **plaintext** (or **cleartext**) and the output from it is called **ciphertext** (or **cryptogram**). The reverse process of transforming ciphertext into plaintext is called **decryption** (or **decipherment**). The encryption and decryption algorithms are collectively called **cryptographic algorithms** (**cryptographic systems** or **cryptosystems**). Both encryption and decryption processes are controlled by a cryptographic **key**, or keys. In a **symmetric** (or **shared-key**) cryptosystem, encryption and decryption use the same (or essentially the same) key; in an **asymmetric** (or **public-key**) cryptosystem, encryption and decryption use two different keys: an **encryption key** and a (**matching**) **decryption key**, and the encryption key can be made public (and hence is also called **public key**) without causing the matching decryption key being discovered (and thus a decryption key in a public-key cryptosystem is also called a **private key**). Figure 2.1 illustrates a pictorial description of a cryptographic system.

We should point out that, within the scope of this chapter, the terms “plaintext”, “ciphertext”, “encryption”, “decryption”, “encryption key” and “decryption key” are pairs of relative notions. For a message M (whether it is plaintext or ciphertext), a crypto algorithm A (whether it represents encryption or decryption) and a cryptographic key K (whether an encryption key or a decryption key), we may denote by

$$M' = A(K, M),$$

a **cryptographic transformation** which is represented by the functionality of either the upper box or the lower box in Figure 2.1. Thus, we can use A' and K' to denote

$$M = A'(K', M'),$$

namely,

$$M = A'(K', A(K, M))$$

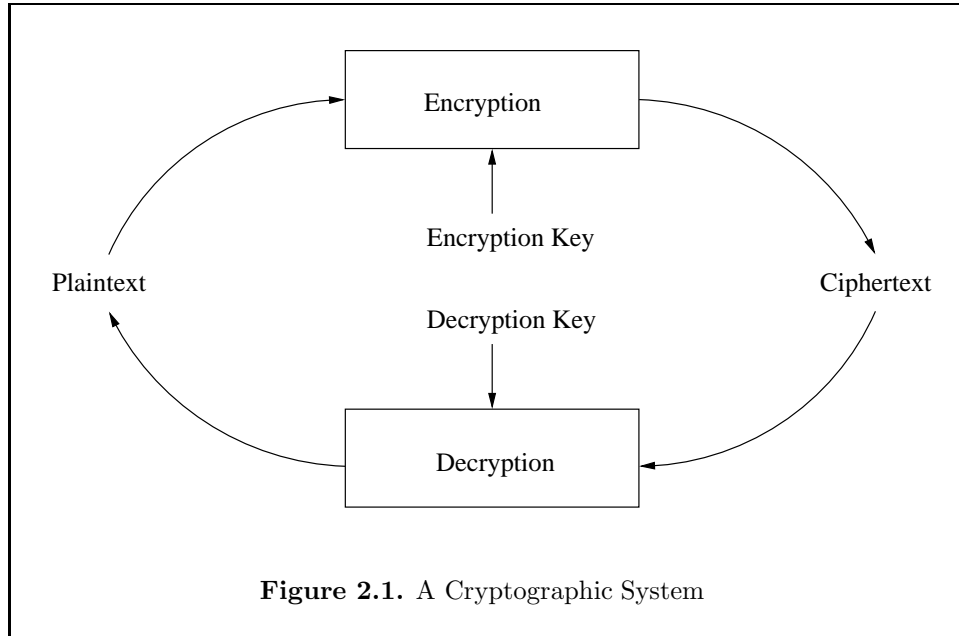


Figure 2.1. A Cryptographic System

completes the circle in Figure 2.1. In the case of symmetric cryptosystem, we may view $K' = K$, and in the case of asymmetric cryptosystem, K' represents the matching public or private component of K . In this chapter ciphertext in a protocol message will be conventionally specified as

$$\{M\}_K.$$

We should notice that the above notation of ciphertext in protocol specifications always means a result of using the perfect cryptographic algorithm in the following two senses:

Property 2.1: Perfect Encryption with Notation $\{M\}_K$

- i) Without the key K (in the case of a symmetric cryptosystem), or the matching private key of K (in the case of an asymmetric cryptosystem), the ciphertext $\{M\}_K$ does not provide any cryptanalytic means for finding the plaintext message M ;
- ii) The ciphertext $\{M\}_K$ and maybe together with some known information about the plaintext message M do not provide any cryptanalytic means for finding the key K (in the case of a symmetric cryptosystem), or the matching private key of K (in the case of an asymmetric cryptosystem).

Perfect encryption with these two properties (there will be an additional property which we shall discuss in §2.5.3) is an idealization from the encryption algorithms that exist in the real world. The idealization is a convenient treatment which allows a segregation of responsibilities of the protocol design and analysis from those of the underlying cryptographic algorithm design and analysis. The segregation eases the job of protocol design and analysis. We shall see in a moment that perfect encryption does not immune a protocol from containing a security flaw. In fact, for every attack on each protocol to be demonstrated in this chapter, none of them depends on any deficiency in the underlying cryptosystems.

We will introduce a number of encryption algorithms In Chapter 7. Nevertheless the above abstract-level description on the functionality of encryption/decryption shall suffice our use in this chapter. It is harmless now for us to think an encryption algorithm as a keyed padlock and a piece of ciphertext as a box of texts with the box being padlocked.

2.2 Vulnerable Environment (the Dolev-Yao Threat Model)

A large network of computers, devices and resources (for example, Internet) is typically open, which means that a **principal** (or **entity**, **agent**, **user**), which can be a computer, a device, a resource, a service provider, a person or an organization of these things, can join such a network and start sending and receiving messages to and from other principals across it, without a need of being authorized by a “super” principal. In such an open environment we must anticipate that there are **bad guys** out there who will do all sorts of bad things, not just passively eavesdropping, but also actively altering (maybe using some unknown calculations or methods), forging, duplicating, re-routining, deleting or injecting messages. The injected messages can be malicious and cause a destructive effect to the principals in the receiving end. In the literature of cryptography such a bad guy is called an (**active**) **attacker** (or **adversary**, **enemy**, **eavesdropper**, **intruder**, etc). In this book we shall name an attacker **Malice** (someone who does harm or mischief, and often does so under the masquerade of a different identity). Malice can be an individual, a coalition of a group of attackers, and, as a special case, a legitimate principal in a protocol (an **insider**).

In general, Malice is assumed to be very clever in manipulating communications over the open network. His manipulation techniques are unpredictable because they are unspecified. Also because Malice can represent a coalition of bad guys, he may simultaneously control a number of network nodes which are geographically far apart. The real reason why Malice can do these things will be discussed in §11.2.

In anticipation of such a powerful adversary over such a vulnerable environment, Dolev and Yao propose a **threat model** which has been widely accepted as the standard threat model for cryptographic protocols [DY81]. In that model, Malice has the following characteristics:

- He can obtain any message passing through the network.
- He is a legitimate user of the network, and thus in particular can initiate a conversation with any other user.
- He will have the opportunity to be a receiver to any principal.
- He can send messages to any principal by impersonating any other principal.

Thus, in the **Dolev-Yao threat model**, any message sent to the network is considered to be sent to Malice for his disposal (according to whatever he is able to compute). Consequently, any message received from the network is treated to have been received from Malice after his disposal. In other words, Malice is considered to have the complete control of the entire network. In fact, it is harmless to just think the open network to be Malice.

However, unless explicitly stated, we do not consider Malice to be *all powerful*. This means that there are certain things that Malice cannot do, even in the case that he represents a coalition of bad guys and thereby may use a large number of computers across the open network in parallel. We list below a few things Malice cannot do without quantifying the meaning of “cannot do”; precise quantification will be made in Chapter 4:

- Malice cannot guess a random number which is chosen from a sufficiently large space.
- Without the correct secret (or private) key, Malice cannot retrieve plaintext from given ciphertext, and cannot create valid ciphertext from given plaintext, with respect to the perfect encryption algorithm.
- Malice cannot find the private component, i.e., the private key, matching a given public key.
- While Malice may have in control of a large public part of our computing and communication environment, in general, he is not in control of many private areas of the computing environment, such as accessing the memory of an principal’s off-line computing device.

The Dolev-Yao threat model will apply to all our protocols.

2.3 Authentication Servers

Suppose that two principals **Alice** and **Bob** (who we have already met in our first cryptographic protocol *Coin-Tossing-Over-Telephone* in Chapter 1) wish to communicate with each other in a secure manner. Suppose also that Alice and Bob have never met before, and therefore they do not already share a secret key between

them and do not already know for sure the other party's public key. Then how can they communicate securely over completely insecure networks?

It is straightforward to see that at least Alice and Bob can make an arrangement to meet each other physically and thereby establish a shared secret key between them, or exchange sure knowledge on the other party's public key. However, in a system with N users who wish to hold private conversations, how many trips these users need make in order to securely establish these keys? The answer is $N(N - 1)/2$. Unfortunately, this means a prohibitive cost for a large system. So this straightforward way for secure key establishment is not practical for use in modern communication systems.

It is nevertheless feasible for each principal who chooses to communicate securely to obtain an **authentication** (and a **directory**) service. Needham and Schroeder suggest that such a service can be provided by an **authentication server** [NS78]. Such a server is like a name registration authority; it maintains a database indexed by names of the principals it serves, and can deliver identifying information computed from a requested principal's cryptographic key that is already shared between the server and the principal.

An authentication server is a special principal who has to be trusted by its users (**client principals**) in that it will always behave honestly, namely, upon a client principal's request (usually a **run** of an authentication protocol using a trusted third party is initiated by a client principal) it will respond exactly according to the protocol's specification, and will not engage in any other activity which will deliberately compromise the security of its clients (so, for instance, it will never disclose any secret key it shares with its clients to any third party). Such a principal is called a **trusted third party** or **TTP** for short. In this book we shall use **Trent** to name a trusted third party.

We suppose that both Alice and Bob use authentication services offered by their respective authentication servers. In an extended network it is inexpedient to have a single central authentication server. Needham and Schroeder proposed to use multiple authentication servers who know each other. Thus, principals served by an authentication server have names of the form "AuthenticationAuthority.SimpleName." The idea of using multiple authentication servers has also been proposed by Diffie and Hellman [DH76a].

However, in order to describe our protocols in this chapter with simplicity and clarity we suppose that Alice and Bob use the same authentication server Trent. In Chapter 11 we will introduce the network authentication basis for Windows 2000 operating system, the Kerberos authentication protocol [DS89], where a general architecture of multiple authentication servers serving in different network realms will be considered.

Being served by the same Trent, we assume that Alice (Bob) shares a cryptographic key with Trent; let the key be denoted by K_{AT} (K_{BT}). Later we shall see

that such a key is called **key-encryption key** because its use is mainly for encryption of other cryptographic keys. Also due to the high cost in the establishment of such a key, it should be used for a prolonged period of time, and hence is also called a **long-term key**.

2.4 Security Properties for Authenticated Key Establishment

All protocols to be described in this chapter are of a kind: they achieve **authenticated key establishment**. The precise meaning of this **security service** can be elaborated by the following three properties.

Let K denote a shared secret key to be established between Alice and Bob, the protocols to be designed in this chapter should achieve a security service with the following three properties:

At the end of the protocol run

1. Only Alice and Bob (or perhaps a principal who is trusted by them) should know K ;
2. Alice and Bob should know that the other principal knows K ;
3. Alice and Bob should know that K is newly generated.

The first property follows the most basic meaning of authentication: identifying the principal who is the intended object of communication. Alice (respectively, Bob) should be assured that the other end of the communication, if “padlocked” by the key K , can only be Bob (respectively, Alice). If the key establishment service is achieved with the help of Trent, then Trent is trust in that, he will not impersonate these two principals.

The second property extends authentication service to an additional dimension, that is, **entity authentication**, or the **liveliness** of a principal who is the intended object of the communication. Alice (respectively, Bob) should be assured that Bob (respectively, Alice) is alive and responsive to the communications in the current protocol run. We shall see later that this property is necessary in order to thwart an attacking scenario based on replaying of old messages.

The need for the third property follows a long established **key management** principle in cryptography. That principle stipulates that a secret cryptographic key should have a short lifetime if it is a shared key and is used for bulk data encryption. Such a key usage is rather different from that of a “key-encryption key” or a “long-term key” which we have described in the end of §2.3). There are two reasons behind this key management principle. First, if a key for data encryption is a shared one, then even if one of the sharing party, say, Alice, is very careful in her key management and disposal, compromise of the shared key by the other sharing party, say, Bob, due to Bob’s carelessness which is totally out

of Alice's control will still result in Alice's security being compromised. Secondly, most data in confidential communications usually contain (possibly large volume of) known or predictable information or structure. For example, a piece of computer program contains large quantity of known texts such as "begin", "end", "class", "int", "if", "then", "else", "++", etc. Such data are said to contain a large quantity of **redundancy** (definition see §??). Encryption of such data makes the key a target for **cryptanalysis** which aims for finding the key or the plaintext. Prolonged such use of a key for encryption of such data may ease the difficulty of cryptanalysis. We should also consider that Malice has unlimited time to spend on finding an old data-encryption key and then to reuse it as thought it were new. The well established and widely accepted principle for key management thus stipulates that a shared data-encryption key should be used for *one* communication session only. Hence, such a key is also referred to as a **session key** and a **short-term key**. The third property of authenticated key establishment service assures Alice and Bob that the session key K established is one that has been newly generated.

2.5 Protocols for Authenticated Key Establishment Using Encryption

Now we are ready to design protocols for authenticated key establishment. The first protocol to be designed merely intends to realize straightforwardly the following simple idea: Alice and Bob, though do not know each other, both know Trent and share respective long-term keys with Trent; so it is possible for Trent to securely pass messages between them.

2.5.1 Protocols Serving Message Confidentiality

Since the environment for our protocols to run is a vulnerable one, our protocols will use encryption to safeguard against any threat. At this initial stage of our step-by-step discussions to follow, we shall restrict our attention on a threat which aims for undermining **message confidentiality**.

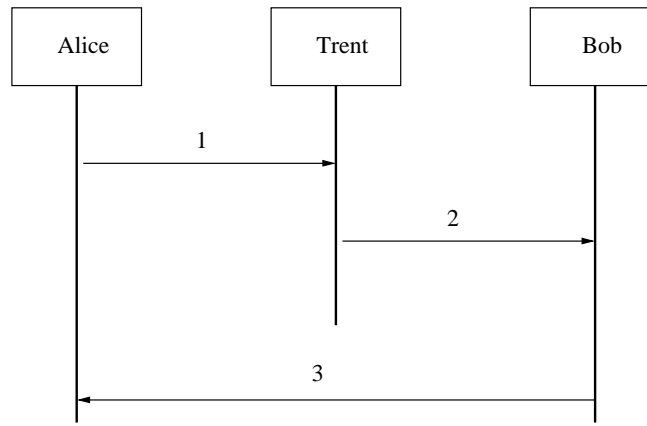
2.5.1.1 Protocol "From Alice to Bob"

Let Alice initiate a run of such a protocol. She starts by generating a session key at random, encrypts it under the key she already shares with Trent, and sends to Trent the resultant ciphertext together with the identities of herself and Bob. Upon receipt of Alice's request for session key delivery, Trent shall first find from his database the shared long-term keys of the two principals mentioned in Alice's request. He shall then decrypt the ciphertext using Alice's key, re-encrypt the result using Bob's key, and then send to Bob the resultant ciphertext. Finally, upon receipt and decryption of the delivered session key material, Bob shall acknowledge the receipt by sending

Protocol 2.1: “From Alice To Bob”

PREMISE Alice and Trent share key K_{AT} ; Bob and Trent share key K_{BT} ;

GOAL Alice and Bob want to establish a new and shared secret key K .



1. Alice generates K at random, creates $\{K\}_{K_{AT}}$, and sends to Trent:
Alice, Bob, $\{K\}_{K_{AT}}$;
2. Trent finds keys K_{AT} , K_{BT} , decrypts $\{K\}_{K_{AT}}$ to reveal K , creates $\{K\}_{K_{BT}}$ and sends to Bob: *Alice, Bob, $\{K\}_{K_{BT}}$* ;
3. Bob decrypts $\{K\}_{K_{BT}}$ to reveal K , forms and sends to Alice:
 $\{Hello\ Alice, I'm\ Bob!\}_K$.

Figure 2.2.

an encrypted message to Alice using the newly received session key. Protocol 2.1 illustrates a protocol description which realizes delivery of a session key from Alice to Bob. In this protocol, Alice is an **initiator**, and Bob, a **responder**.

In this chapter we shall introduce most of our protocols (and attacks on them) in two parts, a pictorial part which illustrates message flows among principals, and an annotative part which provides the details of the actions performed by principals regarding the messages sent or received. Although the annotative part alone should

suffice us to specify a protocol with needed precision (the annotative part alone is what we will use in the specification of protocols in the rest of the book beyond this chapter), by adding pictorial presentation of message flows we intend to allow those readers who are new to the area of cryptographic protocols with an easy start. This is a purpose that this chapter should serve.

Before investigating whether Protocol “From Alice To Bob” contains any security flaw we should comment a design feature of it. The protocol lets Alice generate a session key to be shared with Bob. Will Bob be happy about this? If it turns out that the session key generated by Alice is not sufficiently random (a cryptographic key should be random to make it difficult to be determined by guessing) then Bob’s security can be compromised since the key is a shared one. Maybe Alice does not care whether the session key is strong, or maybe she just wants the key to be easily memorable. So as long as Bob does not trust Alice (may not even know her prior to a protocol run), he should not feel comfortable to accept a session key generated by her and to share with her. We shall modify this protocol by removing this design feature and discuss security issues of the modified protocol.

2.5.1.2 Protocol “Session Key from Trent”

Since Trent is trusted by both client principals, he should be trusted to be able to properly generate the session key. Protocol 2.1 is thus modified to Protocol 2.2. It starts with Alice sending to Trent the identities of herself and Bob, the two principals who intend to share a session key for secure communications between them. Upon receipt of Alice’s request, Trent shall find from his database the respective keys of the two principals, shall generate a new session key to be shared between the two principals and shall encrypt the session key under each of the principals’ keys. Trent should then send the encrypted session key material back to Alice. Alice shall process her own part and shall relay to Bob the part intended for him. Finally, Bob shall process his share of the protocol which ends at sending out an acknowledgement for the receipt of the session key. We shall name the modified Protocol “Session Key From Trent”.

With the session key K being encrypted under the perfect encryption scheme, a passive eavesdropper, upon seeing the communications in a run of Protocol “Session Key From Trent” and without the encryption keys K_{AT} nor K_{BT} , will not gain anything about the session key K since it may only be read by the legitimate recipients via decryption using the respective keys they have.

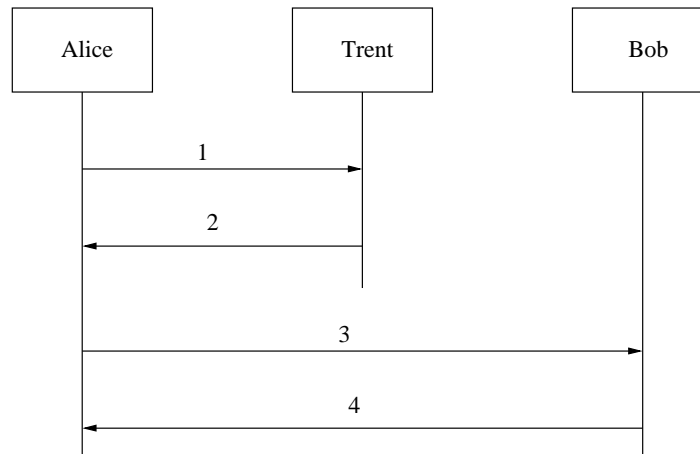
2.5.2 Attack, Fix, Attack, Fix, ...

We now illustrate a standard scene of this book, that is, attack, fix, attack, fix, ...

Protocol 2.2: “Session Key From Trent”

PREMISE Alice and Trent share key K_{AT} ; Bob and Trent share key K_{BT} ;

GOAL Alice and Bob want to establish a new and shared secret key K .



1. Alice sends to Trent: *Alice, Bob*;
2. Trent finds keys K_{AT} , K_{BT} , generates K at random and sends to Alice: $\{K\}_{K_{AT}}$, $\{K\}_{K_{BT}}$;
3. Alice decrypts $\{K\}_{K_{AT}}$, and sends to Bob: *Trent, Alice*, $\{K\}_{K_{BT}}$;
4. Bob decrypts $\{K\}_{K_{BT}}$ to reveal K , forms and sends to Alice: $\{Hello\ Alice, I'm\ Bob!\}_K$.

Figure 2.3.

2.5.2.1 An attack

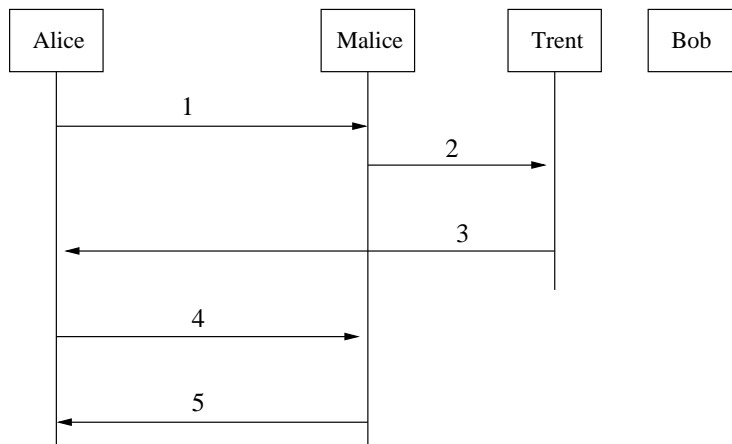
However, Protocol “Session Key From Trent” is flawed. The problem with the protocol is that the information about *who* should get the session key is not protected. An attack is shown in Example 2.1. In the attack, Malice intercepts some messages

Example 2.1: An Attack on Protocol “Session Key From Trent”

PREMISE In addition to that in Protocol “Session Key From Trent”, Malice and Trent share key K_{MT} ;

RESULT OF ATTACK

Alice thinks sharing a key with Bob
while actually sharing it with Malice.



1 Alice sends to Malice(“Trent”): *Alice, Bob*;

1’ Malice(“Alice”) sends to Trent: *Alice, Malice*;

2 Trent finds keys K_{AT} , K_{MT} , generates K_{AM} at random and sends to Alice: $\{K_{AM}\}_{K_{AT}}$, $\{K_{AM}\}_{K_{MT}}$;

3 Alice decrypts $\{K_{AM}\}_{K_{AT}}$, and sends to Malice(“Bob”): *Trent, Alice, $\{K_{AM}\}_{K_{MT}}$* ;

4 Malice(“Bob”) sends to Alice: $\{Hello\ Alice, \ I'm\ Bob!\}_{K_{AM}}$.

Figure 2.4.

transmitted over the network, modifies them and sends them to some principals by impersonating some other principals. In the attack shown in Example 2.1 we write

Alice sends to Malice(“Trent”): ...

to denote Malice’s action of intercepting Alice’s message intended for Trent, and we use

Malice(“Alice”) sends to Trent: ...

to denote Malice’s action of sending message to Trent by impersonating Alice. We should note that according to the Dolev-Yao threat model for our protocol environment that we have agreed in §2.2, Malice is assumed to have the entire control of the vulnerable network. So Malice is capable of performing the above malicious actions. We can imagine the symbol (“principal_name”) to be a mask worn by Malice when he is manipulating protocol messages passing along the network. In §11.2 we shall see technically how Malice could manipulate messages transmitted over the network this way.

Malice begins with intercepting the initial message from Alice to Trent. That message is meant for instructing Trent to generate a session key to share with Alice and Bob. Malice alters it by replacing Bob’s identity with his own and then sends the altered message to Trent. Trent will think that Alice wants to talk to Malice. So he generates a new session key K_{AM} to share between Alice and Malice, and encrypts it with the respective keys that he shares with these two principals. Since Alice cannot distinguish between encrypted messages meant for other principles she will not detect the alteration. Malice then intercepts the message from Alice intended for Bob so that Bob will not know that he is requested to run the protocol. The result of the attack is that Alice will believe that the protocol has been successfully completed with Bob whereas in fact Malice knows K_{AM} and so can masquerade as Bob as well as learn all the information that Alice intends to send to Bob. Notice that this attack will only succeed if Malice is a legitimate user known to Trent. This, again, is a realistic assumption – an insider attacker is often more of a threat than outsiders.

We have seen that the above attack works as a result of Malice’s alteration of Bob’s identity. We should notice the fact that the alteration is possible because Bob’s identity is sent in cleartext. This suggests us to repair the protocol by hiding Bob’s identity.

2.5.2.2 A fix

Having seen the attack in which Malice alters Bob’s identity, it seems to be straightforward to repair Protocol “Session Key From Trent”. For example, we can modify the protocol into one with Bob’s identity in the first message line being treated as a secret and encrypted under the key shared between Alice and Trent, namely, the first message line in Protocol “Session Key From Trent” should be correctly

modified into

1. Alice sends to Trent: $Alice, \{Bob\}_{K_{AT}}$;

Notice that it is necessary for Alice's identity to remain in cleartext and so Trent will be able to know which key he should use to decrypt the ciphertext part.

2.5.2.3 Another attack

However, the above way of "repairing" does not provide a sound fix for Protocol "Session Key From Trent". For example, it is easy to see that Malice can do the following

1. Malice("Alice") sends to Trent: $Alice, \{Malice\}_{K_{AT}}$;

with the rest of the attack runs exactly the same as that in Example 2.1. If initially Malice did not know to whom Alice was intending to run the protocol, he would know that piece of information when he intercepts Alice's message to Bob since that message has to contain Bob's address in order for the network to correctly deliver the message. So Malice can in the end still successfully masquerade as Bob. Notice that in this attack we assume that Malice has the ciphertext $\{Malice\}_{K_{AT}}$; this is possible as it can be the case that Malice has recorded it from a previous protocol run (a correct run) between Alice and Malice.

2.5.2.4 Yet another attack

In fact, another way to attack Protocol "Session Key From Trent" (or its "fix" shown above) does not rely on change of any principal's identity. Instead, Malice can alter the message from Trent to Alice (message line 2 in Protocol "Session Key From Trent") into the following

- Malice("Trent") sends to Alice: $\{K'\}_{K_{AT}}, \dots$;

here K' is a session key transported in a previous protocol run (a correct run) between Alice and Malice such that Malice has recorded the ciphertext part $\{K'\}_{K_{AT}}$. The rest of the attack run is similar to that in the attack in Example 2.1: Malice should intercept the subsequent message from Alice to Bob, and finally acknowledges Alice by masquerading as Bob

- Malice("Bob") sends to Alice: $\{Hello Alice, I'm Bob!\}_{K'}$.

The fact that the "fixed" versions of Protocol "Session Key From Trent" can be attacked with or without altering Bob's identity clearly shows that to have Bob's identity in the first line of Protocol "Session Key From Trent" protected in terms of confidentiality cannot be a correct security service. The attacks demonstrated so far have shown possibilities for Malice to alter some protocol messages without detection. This suggests that the protocol needs a security service which can guard against tampering of messages.

This brings us to the following security service.

2.5.3 Protocol With Message Authentication

We have seen in the attacks shown so far that Malice has always been able to alter some protocol messages without detection. Indeed, none of the protocols designed so far has provided any cryptographic protection against message alteration. Thus, one way to fix these protocols is to provide such protection. The protection should enable legitimate principals who have the right cryptographic keys to detect any unauthorized alteration of any protected protocol messages. Such protection or security service is called **message authentication** (also called **message integrity**).

2.5.3.1 Protocol “Message Authentication”

We observe that Malice’s alteration of the protocol messages has caused the following two effects. Either a session key is shared between wrong principals, or a wrong session key gets established. Therefore we propose that the message authentication protection should provide a cryptographic binding between the session key to be established and its intended users. This leads to a new protocol: Protocol 2.3, where the identities of Alice and Bob are included in the encrypted message parts sent by Trent. We should name the new protocol “Message Authentication”.

We should pay a particular attention to the annotated part of Protocol “Message Authentication” where it instructs

3. Alice (decrypts $\{Bob, K\}_{K_{AT}}$), **checks Bob’s identity**, ...

4. Bob (decrypts $\{Alice, K\}_{K_{BT}}$), **checks Alice’s identity**, ...

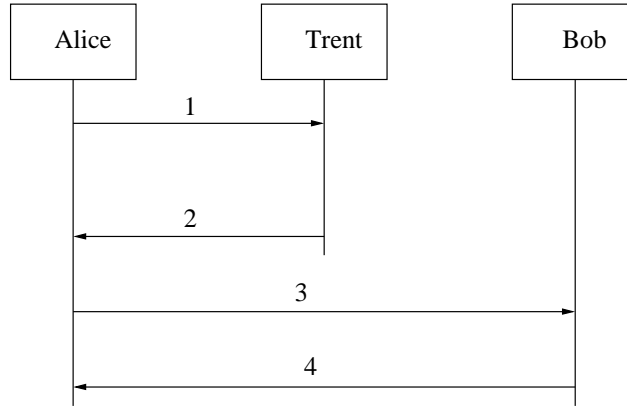
Here in Protocol “Message Authentication”, checking of the intended principals’ identities makes a crucial distinction between this protocol and its predecessors (i.e., Protocol “Session Key From Trent” and its “fixes”). These checkings are possible only after correct decryption of the respective ciphertext blocks using the correct cryptographic keys. Thus, the cryptographic operation “decryption-and-checking” performed by the recipient attempts to achieve a message authentication service which enables the recipient to verify the cryptographic bindings between the session key to be established and its intended users. A correct decryption result should imply that the ciphertext message blocks in question have not been altered in transition. That is how Protocol “Message Authentication” should thwart the attacks shown so far.

We should point out that to achieve message authentication, the operation of “decryption-and-checking” (performed by a recipient) is not a correct **mode of operation**. In Chapter 8 we shall see that the correct mode of operation should be “re-encryption-and-checking” (again performed by a recipient). The reason that we use an incorrect or imprecise mode of operation in this chapter is merely be-

Protocol 2.3: “Message Authentication”

PREMISE Alice and Trent share key K_{AT} ; Bob and Trent share key K_{BT} ;

GOAL Alice and Bob want to establish a new and shared secret key K .



1. Alice sends to Trent: *Alice, Bob*;
2. Trent finds keys K_{AT} , K_{BT} , generates K at random and sends to Alice: $\{Bob, K\}_{K_{AT}}$, $\{Alice, K\}_{K_{BT}}$;
3. Alice decrypts $\{Bob, K\}_{K_{AT}}$, checks Bob's identity, and sends to Bob: *Trent*, $\{Alice, K\}_{K_{BT}}$;
4. Bob decrypts $\{Alice, K\}_{K_{BT}}$, checks Alice's identity, and sends to Alice: *Hello Alice, I'm Bob!* $\}_K$.

Figure 2.5.

cause “encryption-by-sender” and “decryption-by-recipient” are the only available cryptographic operations for us to use at this stage.

Since we will use an incorrect mode of operation to realize the message authentication service, it is necessary for us to explicitly state an additional property requirement that our encryption algorithm must satisfy. The property is given be-

low (its enumeration (iii) follows the enumeration of the other two properties for “The Perfect Encryption with Notation $\{M\}_K$ ” that we have listed in §2.1).

Property 2.2: Perfect Encryption with Notation $\{M\}_K$ (for message authentication service)

- iii) Without the key K , even with the knowledge of the plaintext M , it should be impossible for someone to alter $\{M\}_K$ without being detected by the recipient during the time of decryption.*

In order to show the importance of this property, below we demonstrate an attack on Protocol “Message Authentication” supposing that our perfect encryption algorithm does not possess the above message authentication property (namely, we assume that the encryption algorithm only possesses the perfect confidentiality properties listed in §2.1). For ease of exposure, we modify the presentation of the ciphertext blocks

$$\{Bob, K\}_{K_{AT}}, \quad \{Alice, K\}_{K_{BT}},$$

in the protocol into the following presentation

$$\{Bob\}_{K_{AT}}, \quad \{K\}_{K_{AT}}, \quad \{Alice\}_{K_{BT}}, \quad \{K\}_{K_{BT}}.$$

With this presentation of ciphertext blocks, we imply that the cryptographic binding between principals’ identities and the session key has been destroyed while the encryption retains the perfect confidentiality service for any plaintext message being encrypted. Protocol “Message Authentication” using this “perfect” encryption scheme should have its message lines 2, 3 and 4 look like the following

2. Trent ..., sends to Alice: $\{Bob\}_{K_{AT}}, \{K\}_{K_{AT}}, \{Alice\}_{K_{BT}}, \{K\}_{K_{BT}}$;

3. Alice decrypts $\{Bob\}_{K_{AT}}$ and $\{K\}_{K_{AT}}$, checks Bob’s identity, ...

4. Bob decrypts $\{Alice\}_{K_{BT}}$ and $\{K\}_{K_{BT}}$, checks Alice’s identity, ...

Obviously, the confidentiality protection provided on the principals identities does not make a point; by simply observing the protocol messages flowing over the network (from senders and to recipients) Malice should be able to determine exactly the plaintext content inside the ciphertext blocks $\{Bob\}_{K_{AT}}$ and $\{Alice\}_{K_{BT}}$. Thus, the modified protocol is essentially the same as Protocol “Session Key From Trent”, and thus can be attacked by essentially the same attacks demonstrated in §2.5.2. The reader can apply these attacks as an exercise.

2.5.3.2 Attack on Protocol “Message Authentication”

Even considering that the encryption algorithm used possesses the message authentication property, Protocol “Message Authentication” can still be attacked. The problem stems from the difference in quality between the long-term key-encrypting keys shared initially between Trent and its clients, and the session keys generated for each protocol run.

First, we note that the relationship between Trent and each of his clients is a long-term based one. This means that a shared key between him and his client is a long-term key. In general, to establish a key between an authentication server and a client is more difficult and more costly than to establish a session key between two client principals (it should require thorough security checking routines, even maybe based on a face-to-face contact). Fortunately, such a key is mainly used in authentication protocols, with infrequent use for encrypting few messages with little redundancy, and hence such use of a key provides little information available for cryptanalysis. Therefore, secret keys shared between an authentication server and its clients can be used for a long period of time. Often they are called **long-term keys**.

On the other hand, we should recall a key management principle we have discussed in §2.4, which stipulates that a session key should only be used for one session only. Consequently, no run of a session-key establishment protocol should establish a session key which is identical to one which was established in a previous run of the protocol. However, this is not the case for Protocol “Message Authentication”. An attack run of the protocol will breach the session key management principle. In this attack, all Malice needs to do is first to intercept Alice’s request (see Protocol 2.3)

1. Alice sends to Malice(“Trent”): ...

and then injects a message line 2 as follows

2. Malice(“Trent”) sends to Alice: $\{Bob, K'\}_{K_{AT}}, \{Alice, K'\}_{K_{BT}}$

Here, the two ciphertext blocks containing K' are a **replay** of old messages which Malice has recorded from a previous run of the protocol (a normal run between Alice and Bob), and therefore this attack will cause Alice and Bob to reuse the old session key K' which they should not use. Notice that, since K' is old, it may be possible for Malice to have discovered its value (maybe because it has been discarded by a careless principal, or maybe due to other vulnerabilities of a session key that we have discussed in §2.4). Then he can either eavesdrop the confidential session communications between Alice and Bob, or impersonate Bob to talk to Alice.

An attack in the above fashion is called a **message replay attack**.

2.5.4 Protocol With Challenge-Response

There are several mechanisms that may be employed to allow users to check that a message in a protocol is not a replay of an old message. These mechanisms will be considered in detail in Chapter 10. However for now we will improve our protocol using a well known method called **challenge-response** (also called **handshake**). Using this method Alice will generate a new random number N_A at the start of the protocol and send this to Trent with the request for a new session key. If this same value (N_A) is returned with a session key such that the two pieces are bound together cryptographically and the cryptographic binding provides a message authentication service (i.e., Alice can verify the message integrity regarding the ciphertext containing N_A), then Alice can deduce that the cryptographic binding has been created by Trent after having received her random number N_A . Moreover, recall our stipulation on the trustworthiness of Trent (see §2.3); Alice knows that Trent will always follow the protocol honestly. So Trent has indeed created a new session key *after* receiving Alice's random challenge. Consequently, the session key should be new (or **fresh, current**), namely, is not a replay of an old key. The random number N_A created by Alice for enabling the challenge-response mechanism is called a **nonce** which stands for a **number used for once** [BAN89].

2.5.4.1 Protocol “Challenge Response” (Needham-Schroeder)

Protocol 2.4 specifies a new protocol which utilizes the challenge-response mechanism for Alice to check the freshness of the session key. We shall temporarily name “Challenge Response” (we will soon to change its name).

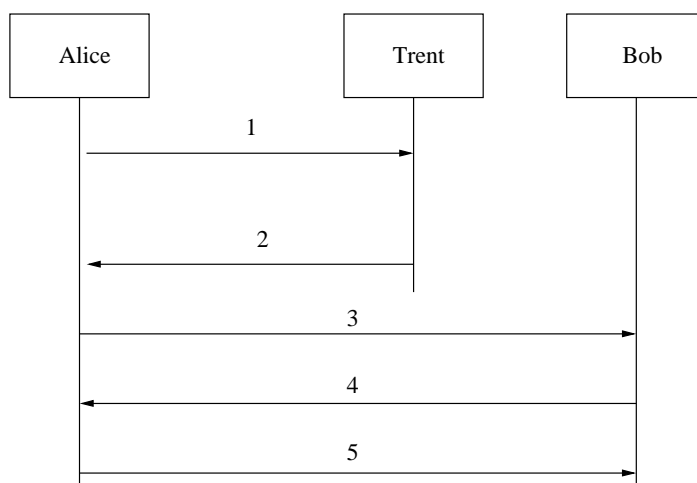
In Protocol “Challenge Response”, Bob also creates a nonce (N_B), but this nonce is not sent to Trent since in this protocol Bob does not directly contact Trent. Instead, Bob's nonce is sent to Alice and then is replied from her after her slight modification (subtracting 1). So if Alice is satisfied that the session key K is fresh and uses it in her response to Bob's freshly created nonce, then Bob should deduce the freshness of the session key. Thus, the mutual confidence on the session key is established.

Protocol “Challenge Response”, which we have reached by a series of steps, is probably the most celebrated in the subject of authentication and key establishment protocols. It is exactly the protocol of Needham and Schroeder which they published in 1978 [NS78]. Below we rename the protocol the Needham-Schroeder Symmetric-key Authentication Protocol. This protocol has also been the basis for a whole class of related protocols.

Protocol 2.4: “Challenge Response”

PREMISE Alice and Trent share key K_{AT} ; Bob and Trent share key K_{BT} ;

GOAL Alice and Bob want to establish a new and shared secret key K .



1. Alice creates N_A at random and sends to Trent: $Alice, Bob, N_A$;
2. Trent generates K at random and sends to Alice: $\{N_A, K, Bob, \{K, Alice\}_{K_{BT}}\}_{K_{AT}}$;
3. Alice decrypts, checks her nonce N_A , checks Bob's ID and sends to Bob: $Trent, \{K, Alice\}_{K_{BT}}$;
4. Bob decrypts, checks Alice's ID, creates random N_B and sends to Alice: $\{I'm Bob! N_B\}_K$;
5. Alice sends to Bob: $\{I'm Alice! N_B - 1\}_K$.

Figure 2.6.

2.5.4.2 Attack on the Needham-Schroeder symmetric-key authentication protocol

Unfortunately the Needham-Schroeder Protocol is vulnerable to an attack discovered by Denning and Sacco in 1981 [DS81]. In the attack of Denning and Sacco, Malice intercepts the messages sent by and to Alice in the message lines 3, 4 and 5, and replaces them with his own version. The attack is given in Example 2.2.

In the attack, Malice becomes active in message line 3 and intercepts Alice's message sent to Bob. He then completely blockades Alice's communication channel and replays old session key material $\{K', Alice\}_{K_{BT}}$ which he recorded from a previous run of the protocol between Alice and Bob. By our assumption on the vulnerability on an old session key, Malice may know the value K' and therefore he can launch this attack to talk to Bob by masquerading as Alice.

We should point out that, the vulnerability of an old session key is only one aspect of the danger of this attack. Another danger of this attack is Malice's successful defeat of an important goal of authentication. We shall specify that goal in §10.2.2 and see how the goal is easily defeated by Malice in §10.7.1.

2.5.5 Protocol With Entity Authentication

The challenge-response mechanism used in the Needham-Schroeder Protocol (the interaction part between Alice and Trent) provides a security service called **entity authentication**. Like message authentication, the service of entity authentication is also obtained via verifying a cryptographic operation (by a verification principal). The difference between the two services is that in the latter case, an evidence of **liveliness** of a principal (proving principal) is shown. The liveliness evidence is shown if the proving principal has performed a cryptographic operation *after* an event which is known to be *recent* to the verification principal. In the case of the Needham-Schroeder Protocol, when Alice receives the message line 2, her decryption operation revealing her nonce N_A shows her that Trent has only operated the encryption *after* the event of her sending out the nonce N_A (since the key used is shared between she and Trent). So Alice knows that Trent is alive after that event. This accomplishes an entity authentication from Trent to Alice.

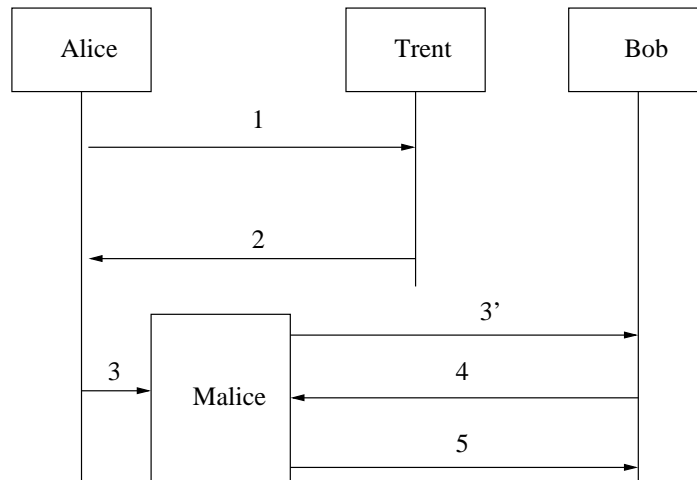
However, in Bob's position in the Needham-Schroeder Protocol, he has no evidence of entity authentication regarding Trent's liveliness.

As usual, once a problem has been spotted, it becomes relatively easy to suggest ways of fixing: Trent should have himself authenticated in entity authentication to *both* of the client principals. This can be done by, for instance, Bob sending a nonce to Trent too, which is to be included by Trent in the session key message returned from Trent. This way of fixing will add more message flows to the protocol (an additional handshake between Bob and Trent). Denning and Sacco suggest to use **timestamps** to avoid adding message flows [DS81].

Example 2.2: An Attack on the Needham-Schroeder Symmetric-key Authentication Protocol

RESULT OF ATTACK

Bob thinks sharing a new session key with Alice while actually the key is an old one and may be known to Malice.



1 and 2. (same as in a normal run)

3. Alice sends to Malice("Bob"): ...

3'. Malice("Alice") sends to Bob: $\{K', Alice\}_{K_{BT}}$;

4. Bob decrypts, checks Alice's ID and sends to Malice("Alice"):
 $\{I'm Bob! N_B\}_{K'}$;

5. Malice("Alice") sends to Bob: $\{I'm Alice! N_B - 1\}_{K'}$.

Figure 2.7.

2.5.5.1 Timestamps

Let T denote a timestamp. The following fix was suggested by Denning and Sacco:

1. Alice sends to Trent: $Alice, Bob$;
2. Trent sends to Alice: $\{Bob, K, T, \{Alice, K, T\}_{K_{BT}}\}_{K_{AT}}$;
3. Alice sends to Bob: $\{Alice, K, T\}_{K_{BT}}$;
4. ... } Same as in the Needham-Schroeder Protocol.
5. ... }

When Alice and Bob receive their protocol messages from Trent, they can verify that their messages are not replays by checking that

$$|Clock - T| < \Delta t_1 + \Delta t_2$$

where $Clock$ gives the recipient's local time, Δt_1 is an interval representing the normal discrepancy between Trent's clock and the local clock, and Δt_2 is an interval representing the expected network delay time. If each client principal sets its clock manually by reference to a standard source, a value of about one or two minutes for Δt_1 would suffice. as long as $\Delta t_1 + \Delta t_2$ is less than the interval since the last use of the protocol, this method will protect against the replay attack in Example 2.2. Since timestamp T is encrypted under the secret keys K_{AT} and K_{BT} , impersonation of Trent is impossible given the perfectness of the encryption scheme.

Needham and Schroeder have considered the use of timestamps, but they reject it on the grounds that it requires a good-quality time value to be universally available [NS87].

2.5.6 Protocol Using Public-key Cryptosystems

The final protocol to be introduced in this chapter is called the Needham-Schroeder Public-key Authentication Protocol [NS78]. We introduce this protocol here with two reasons both of which fall inside the agenda of this chapter. First, the protocol lets us obtain an initial familiarity to the use of public-key cryptosystems. Secondly, we shall show a subtle attack on this protocol. Even though the protocol looks simple, the attack was found seventeen years after the publication of the protocol.

2.5.6.1 Public-key cryptosystems

We use key labels such as K_A for Alice's public key and K_A^{-1} for the matching private key (Alice's private key). It is supposed that Alice is the only person who

has in possession of her private key. The following ciphertext block

$$\{\dots\}_{K_A}$$

denotes the perfect encryption of the plaintext “...” using Alice’s public key K_A . It is supposed that to decrypt the above ciphertext one must use the matching private key K_A^{-1} . Since Alice is supposed to be the only person to possess the private key, only she is supposed to be able to perform decryption to retrieve the plaintext “...”. Analogously, the following ciphertext block

$$\{\dots\}_{K_A^{-1}}$$

denotes the perfect encryption of the plaintext “...” using Alice’s private key K_A^{-1} , and decryption is only possible with the use of Alice’s public key K_A . With the knowledge of K_A being Alice’s public key, an action of decryption using K_A provides one with further knowledge that the ciphertext $\{\dots\}_{K_A^{-1}}$ is created by Alice since the creation requires to use a key that only she has in possession. For this reason, the ciphertext $\{\dots\}_{K_A^{-1}}$ is also called Alice’s (**digital**) **signature** on message “...”, and an action of decryption using K_A is called verification of Alice’s signature on message “...”.

2.5.6.2 Needham-Schroeder public-key authentication protocol

Suppose that Trent has in possession of authenticated copies of the public keys of all his client principals. Also, every client principle has an authenticated copy of Trent’s public key. Protocol 2.5 specifies the Needham-Schroeder Public-key Authentication Protocol.

Here Alice is an initiator who seeks to establish a session with responder Bob, with the help of Trent. In step 1, Alice sends a message to Trent, requesting Bob’s public key. Trent responds in step 2 by returning the key K_B , along with Bob’s identity (to prevent the sort of attacks in §2.5.2), encrypted using Trent’s private key K_T^{-1} . This forms Trent’s digital signature on the protocol message which assures Alice that the message in step 2 is originated from Trent (Alice should verify the signature using Trent’s public key). Alice then seeks to establish a connection with Bob by selecting a nonce N_A at random, and sending it along with her identity to Bob (step 3), encrypted using Bob’s public key. When Bob receives this message, he decrypts the message to obtain the nonce N_A . He requests (step 4) and receives (step 5) the authentic copy of Alice’s public key. He then returns the nonce N_A , along with his own new nonce N_B , to Alice, encrypted with Alice’s public key (step 6). When Alice receives this message she should be assured that she is talking to Bob, since only Bob should be able to decrypt message 3 to obtain N_A and this must have been done *after* her action of sending the nonce out (a recent action). Alice then returns the nonce N_B to Bob, encrypted with Bob’s public key. When Bob receives this message he should, too, be assured that he is talking to Alice, since

Protocol 2.5: Needham-Schroeder Public-key Authentication Protocol

PREMISE Alice's public key is K_A ;
 Bob's public key is K_B ;
 Trent's public key is K_T ;

GOAL Alice and Bob establish a new and shared secret.

1. Alice sends to Trent: *Alice, Bob*;
2. Trent sends to Alice: $\{K_B, Bob\}_{K_T^{-1}}$;
3. Alice verifies Trent's signature on "*K_B, Bob*", creates her nonce N_A at random, and sends to Bob: $\{N_A, Alice\}_{K_B}$;
4. Bob decrypts, checks Alice's ID and sends to Trent: *Bob, Alice*;
5. Trent sends to Bob: $\{K_A, Alice\}_{K_T^{-1}}$;
6. Bob verifies Trent's signature on "*K_A, Alice*", creates his nonce N_B at random, and sends to Alice: $\{N_A, N_B\}_{K_A}$;
7. Alice decrypts, and sends to Bob: $\{N_B\}_{K_B}$.

Figure 2.8.

only Alice should be able to decrypt message 6 to obtain N_B (also a recent action). Thus, a successful run of this protocol does achieve the establishment of the shared nonces N_A and N_B and they are shared secrets exclusively between Alice and Bob. Further notice that both principals contribute to these shared secrets recently, they have the freshness property. Also, each principal should trust the randomness of the secrets as long as her/his part of the contribution is sufficiently random.

Needham and Schroeder suggest that N_A and N_B , which are from a large space, can be used to initialize a shared secret key (“as the base for seriation of encryption blocks”) [NS78] for subsequent secure communications between Alice and Bob.

Denning and Sacco have pointed out that this protocol provides no guarantee that the public keys obtained by the client principals are current, rather than replays of old, possibly compromised keys [DS81]. This problem can be overcome in various ways, for example by including timestamps in the key deliveries. Below we assume that the clients’ public keys are obtained from Trent are current and good.

2.5.6.3 Attack on the Needham-Schroeder public-key authentication protocol

Lowe discovers an attack on the Needham-Schroeder Public-key Authentication Protocol [Low95].

Lowe observes that this protocol can be considered as the interleaving of two logically disjoint protocols; steps 1, 2, 4 and 5 are concerned with obtaining public keys, whereas steps 3, 6 and 7 are concerned with the authentication of Alice and Bob. Therefore, we can assume that each principal initially has the authentic copies of each other’s public key, and restrict our attention to just the following steps (we only list message flows; the reader may refer to Protocol 2.5 for annotative details):

3. Alice sends to Bob: $\{N_A, Alice\}_{K_B}$;
6. Bob sends to Alice: $\{N_A, N_B\}_{K_A}$;
7. Alice sends to Bob: $\{N_B\}_{K_B}$.

We shall consider how Malice can interact with this protocol. We assume that Malice is a legitimate principal in the system, and so other principals may try to set up standard sessions with Malice. Indeed, the attack below starts with Alice trying to establish a session with Malice. Example 2.3 describes the attack.

The attack involves two simultaneous runs of the protocol; in the first run (steps 1-3, 1-6 and 1-7), Alice establishes a valid session with Malice; in the second run (steps 2-3, 2-6 and 2-7), Malice impersonates Alice to establish a bogus session with Bob. In step 1-3, Alice starts to establish a normal session with Malice, sending him a nonce N_A . In step 2-3, Malice impersonates Alice to try to establish a bogus session with Bob, sending to Bob the nonce N_A from Alice. Bob responds in step 2-6 by selecting a new nonce N_B , and trying to return it, along with N_A , to Alice. Malice intercepts this message, but cannot decrypt it because it is encrypted with Alice’s public key. Malice therefore seeks to use Alice to do the decryption for him, by forwarding the message to Alice in step 1-6; note that this message is of the form expected by Alice in the first run of the protocol. Alice decrypts the message to obtain N_B , and returns this to Malice in step 1-7 (encrypted with Malice’s public key). Malice can then decrypt this message to obtain N_B , and returns this to Bob in step 2.7, thus completing the second run of the protocol. Hence Bob believes that Alice has correctly established a session with him and they share exclusively

Example 2.3: Lowe's Attack on the Needham-Schroeder Public-key Authentication Protocol

PREMISE Alice's public key is K_A , Bob's public key is K_B ,
Malice's public key is K_M ;

RESULT OF ATTACK

Bob thinks sharing secrets N_A, N_B with Alice
while actually sharing them with Malice.

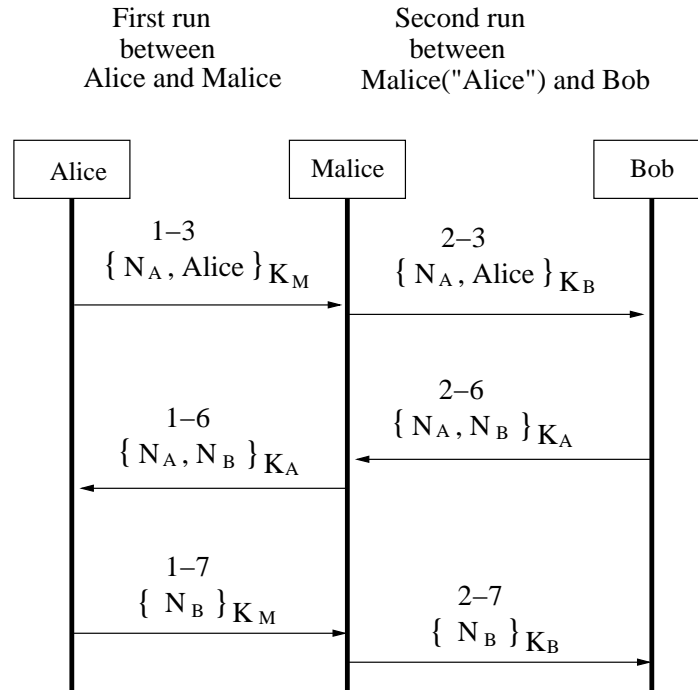


Figure 2.9.

the secret nonces N_A and N_B .

A crucial step for Malice to succeed the attack is Alice's decryption of Bob's nonce N_B for Malice unwittingly. We say that a principal is used as a **oracle** when

the principal performs a cryptographic operation unwittingly for an attacker.

We can imagine the following consequences of this attack. Malice may include the shared nonces within a subsequent message suggesting a session key, and Bob will believe that this message originated from Alice. Similarly, if Bob is a bank, then Malice could impersonate Alice to send a message such as:

Malice(“Alice”) sends to Bob:

$\{N_A, N_B, \text{ Transfer } \pounds 1,000,000 \text{ from my account to Malice's}\}_{K_B}$.

2.5.6.4 A fix

It is fairly easy to change the protocol so as to prevent the attack. If we include the responder’s identity in message 6 of the protocol

6. Bob sends to Alice: $\{Bob, N_A, N_B\}_{K_A}$;

then step 2-6 of the attack would become

2-6. Bob sends to Malice(“Alice”): $\{Bob, N_A, N_B\}_{K_A}$;

now because Alice is expecting a message with Malice’s identity, Malice cannot successfully replay this message in step 1-6 with an intention to use Alice as a decryption oracle.

This fix represents an instance of a principle for cryptographic protocols design suggested by Abadi and Needham [AN95]:

If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal’s name explicitly in the message.

However, we should refrain ourselves from claiming that this way of “fixing” should result in a secure protocol. In §15.2.1 we will reveal several additional problems in this protocol due to an undesirable design feature which can be referred to as “message authentication via decryption-and-checking” (we have labelled it a wrong mode of operation, see §2.5.3.1). That design feature appears generally in authentication protocols using secret-key or public-key cryptographic techniques and has appeared in all protocols in this chapter (the design feature has been retained in our “fix” of the Needham-Schroeder Public-key Authentication Protocol, and hence our “fix” is still not a correct one). Methodical fixes for the Needham-Schroeder Authentication Protocols (both symmetric-key and public-key) will be given in §15.2.3.

The error-prone nature of authentication protocols has inspired the considerations of systematic approaches to the development of correct protocols. That topic will be introduced in Chapter 15.

2.6 Chapter Summary

Some design protection mechanisms, others want to crack them. This is a fact of life and there is nothing special about it. However, in this chapter we have witnessed a rather sad part of this fact of life in authentication protocols: they, as protection mechanisms, are very easily erred.

Actually, all complex systems are easily contain design errors. However, unlike in the case of systems which provide security services, users and the environment of other complex system are generally non-hostile or even friendly. For example, a careful user of a buggy software may quickly become learnt to avoid certain usages to avoid system crash. However, the environment and some users of security system are always hostile: the whole reason for their existence is to attack the system. Exploiting system design errors is of course an irresistible source of tricks for them.

We have used authentication protocols as a means to manifest the error-prone nature of security systems. Although it seems that protocols are more notorious in error-prone-ness due to their communication nature, the real reason for us to have used authentication protocols is that they require relatively simpler cryptographic techniques and therefore are more suitable for serving our introductory purpose at this early stage of the book. We should remember that it is the hostility of the environment for all security systems that should always alert us to be careful when we develop security systems.

We will return to studying authentication protocols in several later chapters. The further study will include a study on the principles and structures of authentication protocols and a taxonomy of attacks on authentication protocols (Chapter 10), case studies of several protocols for real world applications (Chapter 11), and formalism approaches to the development of correct authentication protocols (Chapter 15).

Part II

MATHEMATICAL FOUNDATIONS

This part is a collection of mathematical material which provides the basic notations, methods, basis of algebraic operations, building blocks of algorithmic procedures and references for modelling, specifying, analyzing, transforming and solving various problems to appear in the rest parts of the book.

This part has four chapters: probability and information theory (Chapter 3), computational complexity (Chapter 4), algebraic foundations (Chapter 5) and number theory (Chapter 6). This part serves as a self-contained mathematical reference. In the rest of the book whenever we meet non-trivial mathematical problems we will be able to refer to the precise places in these four chapters for to obtain supporting facts and/or foundations. Therefore our way of including the mathematical material in this book will help the reader to conduct an active and interactive way to learn the mathematical foundations for modern cryptography.

We will provide a proof for a theorem if the proof may help the reader to develop skills which are relevant to the study of cryptography. For the same reason, we will provide sufficiently detailed elaborations for a number of algorithms which are important to the development of cryptographic topics in this book. Sometimes, our mathematical presentations have to make use of facts from other branches of mathematics (e.g., linear algebra) which do not have a direct relevance to the cryptographic topics to be developed; in such cases we will simply use the needed facts without proof.

STANDARD NOTATION

The following standard notation is used throughout the rest of the book. Some notation will be defined locally near its first use, other notation will be used without further definition.

\emptyset	empty set
$S \cup T$	union of sets S and T
$S \cap T$	intersection of sets S and T
$S \setminus T$	difference of sets S and T
$S \subseteq T$	S is a subset of T
$\#S$	number of elements in set S (e.g., $\#\emptyset = 0$)
$x \in S, x \notin S$	element x in (not in) set S
$x \in_U S$	sampling element x in set S at uniformly random
$x \in (a, b), x \in [a, b],$	x in open interval (a, b) (x in closed interval $[a, b]$)
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$	sets of natural numbers, integers, rationals, reals and complex numbers
\mathbb{Z}_n	integers modulo n
\mathbb{Z}_n^*	multiplicative group of integers modulo n
\mathbb{F}_q	finite field with q elements
$\text{desc}(A)$	the description of algebraic structure A
$x \leftarrow D$	value assignment according to distribution D
$x \leftarrow_U S$	value assignment according to the uniform distribution in set S
$a \pmod{b}$	modulo operation, the remainder of a divided by b
$x \mid y, x \nmid y$	integer y is divisible (not divisible) by integer x
$\stackrel{\text{def}}{=}$	defined to be
\forall	for all
\exists	there exists
$\text{gcd}(x, y)$	greatest common divisor of x and y
$\text{lcm}(x, y)$	least common multiple of x and y
$\lfloor x \rfloor$	the maximum integer less than or equal to x
$\lceil x \rceil$	the least integer greater than or equal to x
$ x $	absolute value of x

$ x $	the length of integer x in the binary representation, i.e., the number of bits (binary digits) in x
$\log_b x$	logarithm to base b of x ; natural log if b is omitted
$\phi(n)$	Euler's function of n
$\lambda(n)$	Carmichael's function of n
$\pi(n)$	number of primes $\leq n$
$\text{ord}(x)$	order of a group element
$\text{ord}_n(x)$	order of $x \pmod n$
$\langle g \rangle$	the cyclic group generated by g
$\left(\frac{x}{y}\right)$	the Legendre-Jacobi symbol of integer x modulo integer y
$\mathbb{J}_n(1)$	$\{ x \mid x \in \mathbb{Z}_n^*, \left(\frac{x}{n}\right) = 1 \}$
QR_n	the set of quadratic residues modulo integer n ;
QNR_n	the set of quadratic non-residues modulo integer n ;
$\deg(P)$	degree of a polynomial P
\overline{E}	complement of event E
$E \cup F$	sum of events E, F
$E \cap F$	product of events E, F
$E \setminus F$	difference of events E, F
$E \subseteq F$	event F contains event E , occurrence of E implies occurrence F
$\text{Prob}[E]$	probability of event E occurring
$\text{Prob}[E \mid F]$	conditional probability of event E occurring given that event F has occurred
$n!$	factorial of n ($= n(n-1)(n-2) \cdots 1$ with $0! = 1$)
$\binom{n}{k}$	Ways of picking k out of n ($= \frac{n!}{k!(n-k)!}$)
$O(f(n))$	function $g(n)$ such that $ g(n) \leq c f(n) $ for some constant $c > 0$ and all sufficiently large n
$O_B()$	$O()$ in the bitwise computation mode
$\neg x$	logical operation NOT (x is a binary variable)
$x \wedge y$	logical operation AND (x and y are binary variables)
$x \vee y$	logical operation OR (x and y are binary variables)
$x \oplus y$	logical operation XOR (x and y are binary variables)
Box	end of proof, remark, example or algorithm
$(* \dots *)$	non-executable comment parts in algorithms or protocols

PROBABILITY AND INFORMATION THEORY

Probability Probability plays an important role in cryptography, and is a basic tool for the analysis of security. We often need to estimate “how possible” an insecure event may occur under certain condition. For example, considering Protocol “Coin Tossing Over Telephone” in Chapter 1, we need to estimate the probability for Alice to succeed in finding a collision for a given one-way function f (which should desirably be bounded by a very small quantity), and that for Bob to succeed in finding the parity of x when given $f(x)$ (which should desirably be very close to $\frac{1}{2}$).

Information Theory An important aspect of security for an encryption algorithm can be referred to as “uncertainty of ciphers”: an encryption algorithm should desirably output ciphertext which has a random distribution in the entire space of its ciphertext message space. Shannon quantifies the uncertainty of information by a notion which he names **entropy**. Historically, the desire for achieving a high entropy in ciphers comes from the need for thwarting a cryptanalysis technique which makes use of the fact that natural languages contain **redundancy** which is related to frequent appearance of some known patterns of in natural languages.

Recently, the need for modern cryptographic systems, in particular, public-key cryptosystems, to have probabilistic behavior has reached a rather stringent degree: **semantic security**. This can be described as the following property: if Alice encrypts either 0 or 1 with equal probability under a semantically secure encryption algorithm, sends the resultant ciphertext c to Bob and asks him to answer which is the case, then Bob, without the correct decryption key, should not have an algorithmic strategy to enable him to discern between the two cases with any “advantage” better than a random guessing. We notice that many “textbook version” of encryption algorithms do not have this desirable property.

Probability and information theory are essential tools for the development of modern cryptographic techniques. In this chapter we introduce the basic notions of

probability (§3.1—§3.5) and information theory (§3.6—§3.7).

3.1 Basic Concept of Probability

Let \mathbb{S} be an arbitrary, but fixed, set of points called **probability space** (or **sample space**). Any element $x \in \mathbb{S}$ is called a **sample point** (also called **outcome**, **simple event** or **indecomposable event**, we shall just use **point** for short). An **event** (also called **compound event** or **decomposable event**) is a subset of \mathbb{S} and is usually denoted by a capital letter (e.g., E). An **experiment** or **observation** is an action of yielding (taking) a point from \mathbb{S} . An **occurrence** of an event E (or event E **occurring**) is when an experiment yields $x \in E$ for some point $x \in \mathbb{S}$.

Example 3.1: Consider an experiment of drawing one playing card from a fair deck (here “fair” means drawing a card at random). Here are some examples of probability spaces, points, events and occurrences of events.

1. \mathbb{S}_1 : The space consists of 52 points, 1 for each card in the deck. Let event E_1 be “aces” (i.e., $E_1 = \{A\spadesuit, A\heartsuit, A\diamondsuit, A\clubsuit\}$). It occurs if the card drawn is an ace of any suit.
2. $\mathbb{S}_2 = \{\text{red, black}\}$. Let event $E_2 = \{\text{red}\}$. It occurs if the card drawn is of red colour.
3. \mathbb{S}_3 : This space consists of 13 points, namely, 2, 3, 4, ..., 10, J, Q, K, A. Let event E_3 be “numbers”. It occurs if the card drawn is 2, or 3, or ..., or 10. \square

Definition 3.1: (Classical Definition of Probability) Suppose that an experiment can yield one of $n = \#\mathbb{S}$ equally possible points and that every experiment must yield a point. Let m be the number of points which form event E . Then value $\frac{m}{n}$ is called the **probability** of the event E occurring and is denoted by

$$\text{Prob}[E] = \frac{m}{n}.$$

Example 3.2: In Example 3.1:

1. $\text{Prob}[E_1] = \frac{4}{52} = \frac{1}{13}$;
2. $\text{Prob}[E_2] = \frac{1}{2}$;
3. $\text{Prob}[E_3] = \frac{9}{13}$. \square

Definition 3.2: (Statistical Definition of Probability) Suppose that n experiments are carried out under the same condition, in which event E has occurred μ times. If value $\frac{\mu}{n}$ becomes and remains stable for all sufficiently large n , then the event E is said to have probability which is denoted by

$$\text{Prob}[E] \approx \frac{\mu}{n}.$$

3.2 Properties

1. A probability space itself is an event called **sure event** and satisfies

$$\text{Prob}[\mathbb{S}] = 1.$$

2. Denoting by \mathbb{O} the event that contains no point (i.e., the event that never occurs), it is called **impossible event** and satisfies

$$\text{Prob}[\mathbb{O}] = 0.$$

3. Any event E satisfies

$$0 \leq \text{Prob}[E] \leq 1.$$

4. If $E \subseteq F$, we say that event E **implies** event F , and

$$\text{Prob}[E] \leq \text{Prob}[F].$$

5. Denote by $\overline{E} = \mathbb{S} \setminus E$ the **complementary event** or **opposite event** of E . Then

$$\text{Prob}[E] + \text{Prob}[\overline{E}] = 1.$$

3.3 Basic Calculation

Denote by $E \cup F$ the sum of events E, F to represent an occurrence of at least one of the two events, and by $E \cap F$ the product of events E, F to represent the occurrence of both of the two events.

3.3.1 Addition Rules

1. $\text{Prob}[E \cup F] = \text{Prob}[E] + \text{Prob}[F] - \text{Prob}[E \cap F]$.
2. If $E \cap F = \mathbb{O}$, we say that the two events are **mutually exclusive** or **disjoint**, and

$$\text{Prob}[E \cup F] = \text{Prob}[E] + \text{Prob}[F].$$

3. If $\bigcup_{i=1}^n E_i = \mathbb{S}$ with $E_i \cap E_j = \emptyset$ ($i \neq j$) then

$$\sum_{i=1}^n \text{Prob}[E_i] = 1.$$

Example 3.3: Show

$$\text{Prob}[E \cup F] = \text{Prob}[E] + \text{Prob}[F \cap \overline{E}]. \quad (3.3.1)$$

Because $E \cup F = E \cup (F \cap \overline{E})$ where E and $F \cap \overline{E}$ are mutually exclusive, (3.3.1) holds as a result of Addition Rule 2. \square

Definition 3.3: (Conditional Probability) Let E, F be two events with E having non-zero probability. The probability of occurring F given that E has occurred is called the conditional probability of F given E and is denoted by

$$\text{Prob}[F | E] = \frac{\text{Prob}[E \cap F]}{\text{Prob}[E]}.$$

Example 3.4: Consider families with two children. Let g and b stand for girl and boy, respectively, and the first letter for the older child. We have four possibilities gg, gb, bg, bb and these are the four points in \mathbb{S} . We associate probability $\frac{1}{4}$ with each point. Let event E be that a family has a girl. Let event F be that both children in the family are girls. What is the probability of F given E (i.e., $\text{Prob}[F | E]$)?

The event $E \cap F$ means gg , and so $\text{Prob}[E \cap F] = \frac{1}{4}$. Since the event E means gg , or gb , or bg , and hence $\text{Prob}[E] = \frac{3}{4}$. Therefore by Definition 3.3, $\text{Prob}[F | E] = \frac{1}{3}$. Indeed, in one-third of the families with the characteristic E we can expect that F will occur. \square

Definition 3.4: (Independent Events) Events E, F are said to be **independent** if and only if

$$\text{Prob}[F | E] = \text{Prob}[F].$$

3.3.2 Multiplication Rules

1. $\text{Prob}[E \cap F] = \text{Prob}[F | E] \cdot \text{Prob}[E] = \text{Prob}[E | F] \cdot \text{Prob}[F]$.
2. If events E, F are independent, then

$$\text{Prob}[E \cap F] = \text{Prob}[E] \cdot \text{Prob}[F].$$

Example 3.5: Consider Example 3.1. We expect the events E_1 and E_2 to be independent. Since their probabilities are $\frac{1}{13}$ and $\frac{1}{2}$, respectively (Example 3.2), applying “Multiplication Rule 2”, the probability of their simultaneous realization (a red ace is drawn) is $\frac{1}{26}$. \square

3.3.3 The Law of Total Probability

The law of total probability is a useful theorem.

Theorem 3.1: If $\bigcup_{i=1}^n E_i = \mathbb{S}$ and $E_i \cap E_j = \emptyset$ ($i \neq j$), then for any event A

$$\text{Prob}[A] = \sum_{i=1}^n \text{Prob}[A | E_i] \cdot \text{Prob}[E_i].$$

Proof Since

$$A = A \cap \mathbb{S} = \bigcup_{i=1}^n (A \cap E_i)$$

where $A \cap E_i$ and $A \cap E_j$ ($i \neq j$) are mutually exclusive, the probabilities of the right-hand-side sum of events can be added up using Addition Rule 2, in which each term follows from an application of “Multiplication Rule 1”. \square

The law of total probability is very useful. We will frequently use it when we evaluate (or estimate a bound of) the probability of an event A which is conditional given some other mutually exclusive events (e.g. and typically, E and \bar{E}). The usefulness of this formula is because often an evaluation of conditional probabilities $\text{Prob}[A | E_i]$ is easier than a direct calculation of $\text{Prob}[A]$.

Example 3.6: (This example uses some elementary facts of number theory. The reader who finds this example to be difficult may return to review it after having studied Chapter 6.)

Let $p = 2q + 1$ such that both p and q are prime numbers. Consider choosing two numbers g and h at random from the set $\mathcal{S} = \{1, 2, \dots, p-1\}$ (with replacement). Let event A be “ h is generated by g ”, that is, $h \equiv g^x \pmod{p}$ for some $x < p$ (equivalently, this means “ $\log_g h \pmod{p-1}$ exists”). What is the probability of A for random g and h ?

It is not very straightforward to evaluate $\text{Prob}[A]$ directly. However, the evaluation can be made easy by first evaluating a few conditional probabilities followed by applying the theorem of total probability.

Denote by $\text{ord}_p(g)$ the (multiplicative) order of $g \pmod{p}$, which is the least natural number i such that $g^i \equiv 1 \pmod{p}$. The value $\text{Prob}[A]$ depends on the following four mutually exclusive events.

- i) $E_1 : \text{ord}_p(g) = p - 1 = 2q$ and we know $\text{Prob}[E_1] = \frac{\phi(2q)}{p-1} = \frac{q-1}{p-1}$ (here $\phi()$ is Euler’s phi function; in \mathcal{S} there are exactly $\phi(2q) = q - 1$ elements of order $2q$). In this case, any $h < p$ must be generated by g (g is a generator of the set \mathcal{S}), and so we have $\text{Prob}[A | E_1] = 1$.

- ii) $E_2 : \text{ord}_p(g) = q$ and similar to case (i) we know $\text{Prob}[E_2] = \frac{q-1}{p-1}$. In this case, h can be generated by g if and only if $\text{ord}_p(h) \mid q$. Since in the set \mathcal{S} there are exactly q elements of orders dividing q , we have $\text{Prob}[A \mid E_2] = \frac{q}{p-1} = \frac{1}{2}$.
- iii) $E_3 : \text{ord}_p(g) = 2$. Because there is only one element, $p-1$, of order 2, so $\text{Prob}[E_3] = \frac{1}{p-1}$. Only 1 and $p-1$ can be generated by $p-1$, so we have $\text{Prob}[A \mid E_3] = \frac{2}{p-1}$.
- iv) $E_4 : \text{ord}_p(g) = 1$. Only element 1 is of order 1, and so $\text{Prob}[E_4] = \frac{1}{p-1}$. Also only 1 can be generated by 1, and we have $\text{Prob}[A \mid E_4] = \frac{1}{p-1}$.

The above four events not only are mutually exclusive, but also form all possible cases for the orders of g . Therefore we can apply the theorem of total probability to obtain $\text{Prob}[A]$:

$$\text{Prob}[A] = \frac{q-1}{p-1} + \frac{q-1}{2(p-1)} + \frac{2}{(p-1)^2} + \frac{1}{(p-1)^2}. \quad \square$$

3.4 Random Variables and Their Probability Distributions

Definition 3.5: (Random Variable) Let \mathbb{S} be a probability space. If each experiment yields a point $\xi \in \mathbb{S}$ with certain probability, then ξ is called a random variable.

In cryptography, we mainly consider random variables from **discrete** probability spaces (such as an interval of integers used as a cryptographic **key-space**). Let \mathbb{S} be a discrete probability space: It has a finite or countable number of isolated points $x_1, x_2, \dots, x_n, \dots, x_{\#\mathbb{S}}$. Notice that we have considered the general case that \mathbb{S} may contain a countable number of points, and in that case, $\#\mathbb{S} = \infty$. This will allow us to conduct computational complexity analysis of our algorithms and protocols in an **asymptotic** manner (see §4.9).

Definition 3.6: (Discrete Distribution Function) Let \mathbb{S} be a discrete probability space and $\xi \in \mathbb{S}$ be a random variable. A discrete distribution function for ξ is a function of type $\mathbb{S} \mapsto \mathbb{R}$ provided by a list of probability values

$$\text{Prob}[\xi = x_i] = p_i \quad (i = 1, 2, \dots, \#\mathbb{S})$$

such that the following conditions are satisfied:

- 1) $p_i \geq 0$;
- 2) $\sum_{i=1}^{\#\mathbb{S}} p_i = 1$.

In this definition, the probability values p_i ($i = 1, 2, \dots, \#\mathbb{S}$) are called (discrete) **probability distribution** of random variable ξ .

Now let us look at two discrete probability distributions which are frequently used in cryptography. From now on we shall always drop the word “discrete” from “discrete probability space”, “discrete probability distribution”, etc. All situations in our considerations will always be discrete.

3.4.1 Uniform Distribution

The most frequently used random variables in cryptography follows **uniform distribution**:

$$\text{Prob}[\xi = x_i] = \frac{1}{\#\mathbb{S}} \quad (i = 1, 2, \dots, \#\mathbb{S})$$

Example 3.7: Let S be the set of non-negative numbers up to k bits (binary digits). Sample a point in S at random by following the uniform distribution. Show that the probability that the sampled point is a k -bit number is $\frac{1}{2}$.

$S = \{0, 1, 2, \dots, 2^k - 1\}$ can be partitioned into two disjoint subsets $S_1 = \{0, 1, 2, \dots, 2^{k-1} - 1\}$ and $S_2 = \{2^{k-1}, 2^{k-1} + 1, \dots, 2^k - 1\}$ where S_2 contains all k -bit numbers, $\#S_1 = \#S_2 = \frac{\#S}{2}$. Applying “Addition 2”, we have

$$\begin{aligned} \text{Prob}[\text{sampled point} \in S_2] &= \text{Prob}\left[\bigcup_{i=2^{k-1}}^{2^k-1} \text{sampled point} = i\right] \\ &= \sum_{i=2^{k-1}}^{2^k-1} \text{Prob}[\text{sampled point} = i] \\ &= \sum_{i=2^{k-1}}^{2^k-1} \frac{1}{\#S} \\ &= \frac{\#S_2}{\#S} \\ &= \frac{1}{2}. \end{aligned}$$

□

In this example, the instruction “sample (a point) p in (a set) S at random by following the uniform distribution” is quite long while it is also a frequent instruction

in cryptography. For this reason, we shall shorten this long instruction into “picking p in S at uniformly random”, or into an even shorter notation: $p \in_U S$.

3.4.2 Binomial Distribution

If random variable ξ takes values $0, 1, \dots, n$, and for value p with $0 < p < 1$

$$\text{Prob}[\xi = k] = \binom{n}{k} p^k (1-p)^{n-k} \quad (k = 0, 1, \dots, n)$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ (ways of picking k out of n), then we say ξ follows **binomial distribution**.

Suppose an experiment has two results, titled “Success” and “Failure” (e.g., tossing a coin results in HEADS or TAILS). Independent repetitions of the experiment are called **Bernoulli trials**. Suppose that in any one trial

$$\text{Prob}[\text{Success}] = p, \quad \text{Prob}[\text{Failure}] = 1 - p$$

Then

$$\text{Prob}[k \text{ Successes in } n \text{ trials}] = \binom{n}{k} p^k (1-p)^{n-k}.$$

Therefore, Bernoulli trial follows the binomial distribution.

Example 3.8: If a fair coin is tossed 10 times, then

$$\text{Prob}[\text{HEADS appears 3 times}] = \binom{10}{3} \left(\frac{1}{2}\right)^3 \left(\frac{1}{2}\right)^7 = \binom{10}{3} \left(\frac{1}{2}\right)^{10}.$$

Also, for all possible numbers of “HEADS appearance”, applying “Addition Rule 2”, we have

$$\begin{aligned} \text{Prob} \left[\bigcup_{i=0}^{10} \text{HEADS appears } i \text{ times} \right] &= \sum_{i=0}^{10} \text{Prob}[\text{HEADS appears } i \text{ times}] \\ &= \sum_{i=0}^{10} \binom{10}{i} \left(\frac{1}{2}\right)^{10} \\ &= (1+1)^{10} \left(\frac{1}{2}\right)^{10} \\ &= 1. \end{aligned}$$

□

3.5 The Birthday Paradox

For any function $f : X \mapsto Y$ where Y is a set of n elements, let us solve the following problem:

For a probability bound ϵ (i.e., $0 < \epsilon < 1$), find a value k such that for k pairwise distinct values $x_1, x_2, \dots, x_k \in_U X$, the k evaluations $f(x_1), f(x_2), \dots, f(x_k)$ satisfy

$$\text{Prob}[f(x_i) = f(x_j)] \geq \epsilon \text{ for some } i \neq j.$$

That is, in k evaluations of the function, a collision has occurred with the probability no less than ϵ .

This problem asks for a value k to satisfy the given probability bound from below for *any* function. We only need to consider functions which have a so-called random property: such a function maps uniform input values in X to uniform output values in Y . Clearly, only a function with such a random property can enlarge the value k for the given probability bound, which can then be able to satisfy other functions for the same probability bound. Consequently, it is necessary that $\#X > \#Y$; otherwise it is possible that for some functions there will be no collision to occur at all.

Thus, we can assume that the function evaluation in our problem has n distinct and equally possible outcomes. We can model such a function evaluation as drawing a ball from a bag of n differently colored balls, recording the color and then replacing the ball. Then the problem is to find the value k such that at least one matching color is met with probability ϵ .

There is no color restriction on the first ball. Let y_i be the color for the i th instance of ball drawing. The second ball should not have the same color as the first one, and so the probability for $y_2 \neq y_1$ is $1 - 1/n$; the probability for $y_3 \neq y_1$ and $y_3 \neq y_2$ is $1 - 2/n$, and so on. Upon drawing of the k th ball, the probability for no collision so far is

$$\left(1 - \frac{1}{n}\right) \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right).$$

For sufficiently large n and relatively small x , we know

$$\left(1 + \frac{x}{n}\right)^n \approx e^x$$

or

$$1 + \frac{x}{n} \approx e^{x/n}.$$

So

$$\left(1 - \frac{1}{n}\right) \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) = \prod_{i=1}^{k-1} \left(1 + \frac{-i}{n}\right) \approx \prod_{i=1}^{k-1} e^{-i/n} = e^{-\frac{k(k-1)}{2n}}.$$

This is the probability for drawing k balls without collision. Therefore the probability for at least one collision will be

$$1 - e^{-\frac{k(k-1)}{2n}}.$$

Equalizing this value to ϵ , we have

$$e^{-\frac{k(k-1)}{2n}} \approx 1 - \epsilon$$

or

$$k^2 - k \approx 2n \log \frac{1}{1 - \epsilon},$$

that is,

$$k \approx \sqrt{2n \log \frac{1}{1 - \epsilon}}. \quad (3.5.1)$$

Thus, for a random function mapping onto Y , we only need to perform this amount of evaluations in order to meet a collision with the given probability ϵ . If we consider $\epsilon = 1/2$, then

$$k \approx 1.1774\sqrt{n}. \quad (3.5.2)$$

The square-root relationship between k and n shown in (3.5.1) and in (3.5.2) suggests that for a random function with the cardinality of the output space being n , we need only to make roughly \sqrt{n} evaluations of the function and find a collision with a non-negligible probability.

This fact has a profound impact on the design of cryptosystems and cryptographic protocols. For example, for a piece of data (e.g., a cryptographic key or a message) hidden as a pre-image of a cryptographic function (which is typically a random function), if the square root of this data is not a sufficiently large quantity, then the data may be discovered by random evaluation of the function. Such an attack is often called a **square-root attack**, or a **birthday attack**. The latter name is due to the following seemingly “paradoxical phenomenon”: taking $n = 365$ in (3.5.2), we find $k \approx 22.49$; that is, in order for two people in a room of random people to have the same birthday with more than 50% chance, we only need 23 people in the room. This seems to be a little bit of counter-intuition at a first glance.

3.6 Information Theory

Shannon's definition for entropy [Sha48a], [Sha48b] of a message source is a measure of the amount of information the source has. The measure is in the form of a function of the probability distribution over the set of all possible messages the source may output.

Let $L = \{a_1, a_2, \dots, a_n\}$ be a language of n different symbols. Suppose a source S may output these symbols with independent probabilities

$$\text{Prob}[a_1], \text{Prob}[a_2], \dots, \text{Prob}[a_n],$$

respectively, and these probabilities satisfy

$$\sum_{i=1}^n \text{Prob}[a_i] = 1. \quad (3.6.1)$$

The entropy of the source S is

$$H(S) = \sum_{i=1}^n \text{Prob}[a_i] \log_2 \left(\frac{1}{\text{Prob}[a_i]} \right). \quad (3.6.2)$$

The entropy function $H(S)$ defined in (3.6.2) captures a quantity which we can name “*number of bits per source output*”.

Let us explain the entropy function by assigning ourselves a simple job: considering that the source S is memoryless, we must record the output from S . A straightforward way to do the job is to record whatever S outputs. However, from (3.6.1) we know that each output from S will be one of the n symbols a_1, a_2, \dots, a_n which are already known to us. It can be quite uninteresting and inefficient to record known things. Thus, the question for us is, how can we *efficiently* record something *interesting* in the output from S ?

Let S output these symbols in a k consecutive sequence, i.e., S outputs a word of k symbols

$$a_{i_1} a_{i_2} \cdots a_{i_k} \quad \text{for } 1 \leq i_k \leq n.$$

Let L_k denote the minimum expected number of bits we have to use in order to record a k -symbol word output from S . We have the following theorem for measuring the quantity L_k .

Theorem 3.2: (Shannon) [Sha48a], [Sha48b]

$$\lim_{k \rightarrow \infty} \frac{L_k}{k} = H(S). \quad \square$$

In fact, it is true that

$$kH(S) \leq L_k \leq kH(S) + 1.$$

In other words, the minimum average number of bits needed for recording per output from S is $H(S)$.

3.6.1 Properties of Entropy

The function $H(S)$ has the minimum value 0 if S outputs some symbol, say a_1 , with probability 1, since then

$$H(S) = \text{Prob}[a_1] \log_2\left(\frac{1}{\text{Prob}[a_1]}\right) = \log_2 1 = 0.$$

This case captures the fact that when we are sure that S will only and definitely output a_1 , then why should we waste any bit to record it?

The function $H(S)$ reaches the maximum value of $\log_2 n$ if S outputs each of these n symbols with equal probability $1/n$, since then

$$H(S) = \frac{1}{n} \sum_{i=1}^n \log_2 n = \log_2 n.$$

This case captures the following fact: since S can output any one of these n symbols with equal probability, we have to prepare $\log_2 n$ bits in order to mark any possible one of the n numbers.

To this end we can think of $H(S)$ as the amount of *uncertainty*, or *information*, contained in each output from S .

Example 3.9: Consider Protocol 1.1 (“Coin Tossing Over Telephone”). Whether running over telephones or on connected computers, that protocol is for Alice and Bob to agree on a random bit. In the protocol, Alice picks a large random integer $x \in_U \mathbb{N}$, then sends $f(x)$ to Bob under the one-way function $f()$, and finally reveals x to Bob after his random guess. Viewed by Bob, x as a whole number should not be regarded as a piece of new information since he knows already that x is one element in \mathbb{N} before even receiving $f(x)$. Bob only uses an interesting part of Alice’s output: the parity of x is used to compute a random bit agreed with Alice. Thus, we have

$$\begin{aligned} H(\text{Alice}) &= \text{Prob}[x \text{ is odd}] \log_2\left(\frac{1}{\text{Prob}[x \text{ is odd}]}\right) + \\ &\quad \text{Prob}[x \text{ is even}] \log_2\left(\frac{1}{\text{Prob}[x \text{ is even}]}\right) \\ &= \frac{1}{2} \log_2 2 + \frac{1}{2} \log_2 2 = 1. \end{aligned}$$

That is, Alice is a source of 1 bit per output, even though her output is a large integer.

If Alice and Bob repeat running Protocol 1.1 n times, they can agree on a string of n bits which they mutually trust to be random (because each party has her/his own random input). In this usage of the protocol, Alice is a source of n large integers

(i.e., n long messages), while Bob has the simple job of recording information from Alice's output. It is now clear that it is the information which is quantified by $H(\text{Alice})$ that Bob should record, not the large integers sent by Alice. \square

3.7 Redundancy in Natural Languages

Consider a source $S(L)$ outputs words in a natural language L . Suppose that, on average, each word in L has k characters. Since by Shannon's Theorem (Theorem 3.2), $H(S(L))$ is the minimum average number of bits per output from $S(L)$ (remember that per output from $S(L)$ is a word of k characters), the value

$$r(L) = \frac{H(S(L))}{k}$$

should be the minimum average number of bits per character in language L . The value $r(L)$ is called the **rate of language** L . Let L be English. Shannon calculated that $r(\text{English})$ is in the range of 1.0 to 1.5 bits/letter [Sha51].

Let $\Sigma = \{a, b, \dots, z\}$. Then we know $r(\Sigma) = \log_2 26 \approx 4.7$ bits/letter. $r(\Sigma)$ is called **absolute rate** of language with alphabet set Σ . Comparing $r(\text{English})$ with $r(\Sigma)$, we see that the actual rate of English is considerably less than its absolute rate.

The **redundancy** of a language L with alphabet set Σ is

$$r(\Sigma) - r(L) \text{ (bits per character).}$$

Thus for a conservative consideration of $r(\text{English}) = 1.5$, the redundancy of English is $4.7 - 1.5 = 3.2$ bits per letter. In terms of percentage, the redundancy ratio is $3.2/4.7 \approx 68\%$. In other words, about 68% of the letters in an English word are redundant. This means a possibility to compress an English article down to 32% of its original volume without loss of information.

Redundancy in a natural language arises from some known and frequent appearing patterns in the language. For example, in English, letter q is almost always followed by u , "the", "ing" and "ed" are a few other known examples of patterns. Redundancy in natural languages provides an important means for **cryptanalysis** which aims for recovering plaintext messages or a cryptographic key from a ciphertext.

Example 3.10: We have mentioned in Chapter 1 that in this book we will study many kinds of attacks on cryptographic algorithms and protocols. In a later chapter (Chapter 13) we will be focusing on four kinds of attacks with some of them having rather long names. They are

- passive attack;

- active attack in chosen plaintext mode;
- active attack in non-adaptive chosen ciphertext mode;
- active attack in adaptive chosen ciphertext mode.

Full meanings of these attacks will be explained in a couple of later chapters. Here we should only point out the following two facts about these attacks:

1. The use of long names is very appropriate because of a non-trivial amount of information behind each long-named attack;
2. In Chapter 13 we will only deal with these four kinds of attacks.

Since only four names of attacks are involved in Chapter 13, the actual entropy of these names can be as low as 2 bits per name. However, in order to uniquely identify these attacks, we need to use more bits than 2. Since in Chapter 13, we do not use a0, a1, a2, a3, we can actually shorten the long names of these four attacks into these short names, respectively. Consequently, within Chapter 13, the entropy for naming these four attacks should be $4.7 + 2 = 6.7$ (bits per name).

On the other hand, we notice that the average length of the four long names is 33.5 (letters). Therefore, the average number of bits per letter in these long names is $6.7/33.5 = 0.2$. From this result, we can further calculate the redundancy of these long names as (within the scope of Chapter 13)

$$\frac{6.7 - 0.2}{6.7} \approx 97\%. \quad \square$$

In the area of study for cryptographic systems with provable security, which is an environment larger than Chapter 13, the extremely shortened names a0, a1, a2, a3 used in Example 3.10 are in fact too short for expressing these attacks without causing unambiguity in understanding. As a matter of fact, the latter three attacks listed in Example 3.10 are shortened to CPA, CCA and CCA2, respectively. (We will adopt these names in Chapter 13 too.)

Finally we point out that the reason why only the latter three long names are shortened is because in the area of study the latter three attacks are discussed more frequently. For “passive attack”, we are comfortable enough to use the long version of the name since the attack is a less frequently discussed topic.

3.8 Chapter Summary

In this chapter we have conducted a very rudimentary study of probability and information theory. However, the material is sufficient for the use in our book.

In probability, it is very important to understand and be familiar with the basic notions, the properties and the rules for the basic calculations. We should emphasize that a good understanding of the very basics, which is not a difficult task, will help the most. As we have seen that the derivation of useful theorems, e.g., the law of total probability and the birthday paradox, totally rests on the applications of a few basic properties and rules.

Later we will frequently meet applications of conditional probability and the law of total probability. In these applications we will become more and more familiar with these useful tools.

In information theory, we now understand that entropy is a measurement for the amount of information contained in a message, or for the degree of randomness, or for the unpredictability, the message is.

COMPUTATIONAL COMPLEXITY

4.1 Introduction

If a random variable follows the uniform distribution, then it is independent from *any* given information. This means that there is *no way* to relate a uniformly random variable to any other information by any means of “computation”. This is exactly the security basis behind the *only* proven *unconditionally* (or *information-theoretically*) secure encryption scheme: one-time pad, that is, mixing a uniformly random string (called key string) with a message string in a bit by bit fashion (see §??). The need for statistical independence between the key string and the message string requires the two strings to have the same length. Unfortunately, this poses an almost un-passable limitation for a practical use of the one-time-pad encryption scheme.

Nevertheless (and somewhat ironical), we are still in a “fortunate” position. At the time of writing, the computational devices and methods which are widely available to us are based on a notion of computation which is not very powerful: To date we have not been very successful in relating, via computation, between two pieces of information if one of them merely “looks random” while in fact they are completely dependent one another (for example, a plaintext-ciphertext pair in many cryptosystems). As a result, modern cryptography has its security based on a so-called *complexity-theoretic* model: Security of such cryptosystems is *conditional* on various assumptions that certain problems are *intractable*. Here, “intractable” means that the widely available computational methods cannot effectively handle these problems.

We should point out that our “fortunate” position may be only temporary. A new and much more powerful model of computation, *quantum information process* (QIP), has emerged. Under the new model of computation, an exponentially many computation steps can be parallelized by manipulating so-called “super-position” of

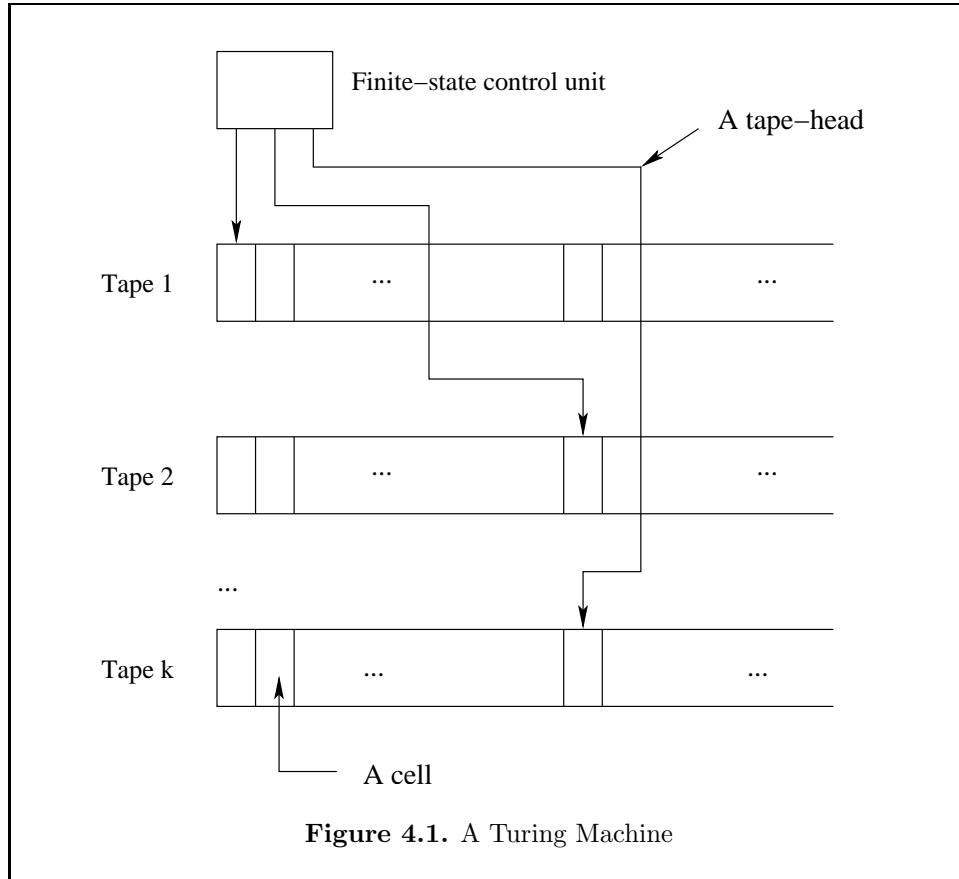
quantum states. The consequence: many nowadays very useful hard problems for the complexity-theoretic based cryptography will collapse (though not all of them), that is, will become useless. For example, using a quantum computer, factorization and multiplication of integers will take similar time if the integers processed have similar sizes, and hence, e.g., the famous public-key cryptosystems of Rivest, Shamir and Adleman (RSA) [RSA78] (see §??) will be thrown out of stage. However, at the time of writing, the QIP technique is still quite distant from practical applications. The current record for factoring a composite number: 15 (see e.g., [WC00]), which is the least size, odd and non-square composite integer.

Therefore, let us not worry too much about the QIP for the time being. The rest of this chapter provides an introduction to our “less-powerful” conventional computational model and to the complexity-theoretic based approach to modern cryptography.

4.2 Turing Machines

In order to make precise the notion of an effective procedure (i.e., an algorithm), Turing proposed an imaginary computing device, called **Turing machine**, to provide a primitive yet sufficiently general model of computation. The computational complexity material to be introduced here follows the computation model of Turing machines. Below we introduce a variant version of Turing machines which are sufficient for our purpose of computational complexity study. A general description of Turing machines can be studied in, e.g., §1.6 of [AHU74].

In our variant, a Turing machine (see picture in Figure 4.1) consists of a *finite-state control unit*, some number k (≥ 1) of *tapes* and the same number of *tape-heads*. The finite-state control unit controls the operations of the tape-heads which read or write some information from or to the tapes; each tape-head does so by accessing one tape, called *its* tape, and by moving along its tape either to left or to right. Each of these tapes is partitioned into an infinite number of *cells*. The machine solves a problem by having a tape-head scanning a string of a finite number of symbols which are placed sequentially in the leftmost cells of one tape; each symbol occupies one cell and the remaining cells to the right on that tape are *blank*. This string is called an *input* of a problem. The scanning starts from the leftmost cell of the tape that contains the input while the machine is in a designated *initial state*. At any time only one tape-head of the machine is accessing its tape. A step of access made by a tape-head on its tape is called a (*legal*) *move*. If the machine starts from the initial state, makes legal moves one after another, completes scanning the input string, eventually causes the satisfaction of a *terminating condition* and thereby terminates, then the machine is said to *recognize* the input. Otherwise, the machine will at some point have no legal move to make; then it will halt without recognizing the input. An input which is recognized by a Turing machine is called an *instance* in a recognizable *language*.



For a given problem, a Turing machine can be fully specified by a function of its finite-state control unit. Such a function can be given in the form of a table which lists the machine's *next-step move* for each state. We shall provide a problem example and a specification of a Turing machine in a moment (see Example 4.1 below).

Upon termination, the number of moves that a Turing machine M has taken to recognize an input is said to be the running time or the *time complexity* of M and is denoted by T_M . Clearly, T_M can be expressed as a function $T_M(n) : \mathbb{N} \mapsto \mathbb{N}$ where n is the *length* or *size* of the input instance, i.e., the number of symbols that consists of the input string when M is in the initial state. Obviously, $T_M(n) \geq n$. In addition to the time requirement, M has also a space requirement S_M which is the number of tape cells that the tape-heads of M have visited in writing access. The quantity S_M can also be expressed as a function $S_M(n) : \mathbb{N} \mapsto \mathbb{N}$ and is said to be the *space complexity* of M .

4.3 Polynomial-Time

We begin with considering the class of languages that are recognizable by Turing machines in **polynomial time**. A function $p(n)$ is a polynomial in n over the integers if it is of the form

$$p(n) = c_k n^k + c_{k-1} n^{k-1} + \cdots + c_1 n + c_0 \quad (4.3.1)$$

where k and c_i ($i = 0, 1, 2, \dots, k$) are constant integers with $c_k \neq 0$ when $k > 0$, the former is called the **degree**, denoted by $\deg(p(n))$, and the latter, the **coefficients**, of the polynomial $p(n)$.

Definition 4.1: Class \mathcal{P} We write \mathcal{P} to denote the class of languages with the following characteristics. A language L is in \mathcal{P} if there exists a Turing machine M and a polynomial $p(n)$ such that M recognizes any instance $I \in L$ in time $T_M(n)$ with $T_M(n) \leq p(n)$ for all non-negative integers n , where n is an integer parameter representing the size of the instance I . We say that L is recognizable in polynomial time.

Roughly speaking, languages which are recognizable in polynomial-time are considered to be *always* “easy”. In other words, polynomial-time Turing machines are considered to be *always* “efficient” (we shall define the notion of “easy” or “efficient” in §4.7). Here let us explain the meaning for *always*. Turing machines which recognize languages in \mathcal{P} are all **deterministic**. A deterministic Turing machine outputs an effect which is entirely determined by the input to, and the initial state of, the machine. In other words, running a deterministic Turing machine twice with the same input and the same initial state, the two output effects will be identical. While this meaning for determinism is quite straightforward, we shall notice that there are two different meanings for the opposite situation: nondeterminism. We shall deal with these two meanings in §4.6 and in §4.8, separately.

We should notice that in Definition 4.1, the universal-style restriction “any instance $I \in L$ ” and “for all non-negative integers n ” are very important. In the study of computational complexity, a problem is considered to be solved only if *any* instance of the problem can be solved by the same Turing machine (i.e., the same method). Only so, the method is sufficiently general and thereby can indeed be considered as a method. Let us look at the following example for an illustration.

Example 4.1: (Language Div3) Let Div3 be the set of non-negative integers divisible by 3. Show $\text{Div3} \in \mathcal{P}$.

We do so by constructing a single-tape Turing machine to recognize Div3 in polynomial time.

We first notice that if we encode the input as integers in the base-3 (i.e., ternary) representation, that is, an input is a string of symbols in $\{0, 1, 2\}$, then the recognition problem becomes trivially easy: an input x is in Div3 if and only if the last

Current state	Symbol on tape	Next move	New state
q_0 (initial state)	0 1 blank	right right “Ding” & Stop	q_0 q_1
q_1	0 1	right right	q_2 q_0
q_2	0 1	right right	q_1 q_2

Table 4.1. The operation of machine DIV3

digit of x is 0. Consequently, the machine to be constructed should simply make consecutive moves to right until reaching a blank symbol, and then it stops with YES answer if and only if the final non-blank symbol is 0. Clearly, this machine can recognize any instance in number of moves which is the size of the instance. Hence $\text{Div3} \in \mathcal{P}$.

However, we want to show that the fact $\text{Div3} \in \mathcal{P}$ should be independent from how we encode the input. It suffices for us to show the case when the input is encoded in the base-2 (i.e., binary) representation. Let this machine be named DIV3. The finite-state control of DIV3 follows a “next move” function specified in Table 4.1.

We now argue that the machine DIV3 defined by the function in Table 4.1 is sufficiently general for recognizing all instances in Div3 .

First, we notice that for recognizing whether or not a binary string $x \in \text{Div3}$, it is sufficient for DIV3 to have three states, corresponding to the cases when it (its tape-head) completes scanning strings $3k$, $3k+1$ and $3k+2$ (for $k \geq 0$), respectively. The least input instance 0 stipulates that DIV3 must be in an initial state (without loss of generality, let the initial state be q_0) upon its completion of scanning input string 0. Without loss of generality, we can assign DIV3 to state q_1 upon its completion of scanning input string 1, and to state q_2 upon its completion of scanning input string 2 ($= (10)_2$)^a.

^aWe use $(a_1 a_2 \dots a_n)_b$, with $a_i < b$ and $i = 1, 2, \dots, n$, to denote a number encoded in the base- b representation; the cases of $b = 10$ and $b = 2$ are often omitted if no confusion arises.

For any non-negative integer a in the binary representation, postfixing a with symbol 0 (respectively, symbol 1) yields value $2a$ (respectively, value $2a + 1$). Thus, after completion of scanning $a = 3k$ (when DIV3 is in state q_0), DIV3 must remain in q_0 upon further scanning symbol 0, since at that point it completes scanning $2a = 6k = 3k'$, and must evolve to q_1 upon further scanning symbol 1, since at that point it completes scanning $2a + 1 = 6k + 1 = 3k' + 1$. Similarly, after completion of scanning $a = 3k + 1$ (when DIV3 is in state q_1), DIV3 must evolve to q_2 upon completion of scanning $2a = 6k + 2 = 3k' + 2$, and must evolve to q_0 upon completion of scanning $2a + 1 = 6k + 3 = 3k'$. The remaining two case for $a = 3k + 2$ are: $2a = 6k + 4 = 3k' + 1$ (DIV3 evolves from q_2 to q_1), and $2a + 1 = 6k + 5 = 3k' + 2$ (DIV3 stays in q_2).

So, the three states q_0 , q_1 and q_2 correspond to DIV3's completion of scanning strings $3k$, $3k + 1$ and $3k + 2$, respectively, for any $k \geq 0$. Now upon the head meeting the special symbol "blank", only in state q_0 DIV3 is configured to ring the bell and stop (meaning to terminate with YES answer) and hence to recognize the input $3k$; in the other two states, DIV3 will have no legal move to make and therefore halt with no recognition.

Finally, it is easy to see $T_{\text{Div3}}(n) = n$. Thus, DIV3 does recognize language Div3 in polynomial time. \square

Example 4.2:

- i) The bit string $10101 (= (21)_{10})$ is recognizable; DIV3 recognizes the string in $T_{\text{Div3}}(|10101|) = |10101| = 5$ moves;
- ii) The bit string $11100001 (= (225)_{10})$ is another recognizable instance; DIV3 recognizes it in $T_{\text{Div3}}(|11100001|) = |11100001| = 8$ moves;
- iii) The bit string $10 (= (2)_{10})$ is not recognizable; DIV3 decides that it is unrecognizable in two moves. \square

4.4 Polynomial-Time Computational Problems

By definition, \mathcal{P} is the class of polynomial-time language recognition problems. A language recognition problem is a **decisional problem**. For every possible input, a decisional problem requires YES or NO to be output. However, class \mathcal{P} is sufficiently general to enclose polynomial-time **computational problems**. For every possible input, a computational problem requires an output to be more general than a YES/NO answer. Since a Turing machine can write symbols to a tape, it can of course output information more general than a YES/NO answer.

For instance, we can design another Turing machine which will not only recognize any instance $x \in \text{Div3}$, but will also output $\frac{x}{3}$ upon recognition of x . Let this new machine be named DIV3-COMP. One very simple way to realize DIV3-COMP is

to have its input to be encoded in the base-3 representation. Then the input is an instance in **Div3** if and only if its final digit is 0, and the output from the machine, upon recognition of the input, should be the content on the input-tape after having erased the last 0 unless 0 is the only symbol on the tape. If one insists that **DIV3-COMP** must only input and output binary numbers, then **DIV3-COMP** can be realized as follows. It first translates an input x from the base-2 representation into the base-3 representation, and upon obtaining $\frac{x}{3}$ in the base-3 representation it translates the number back to the base-2 representation as the final output. It is evident that these translations can be done digit-by-digit *mechanically* in $c|x|$ moves where c is a constant. To this end we know

$$T_{\text{DIV3-COMP}}(|x|) \leq C|x|$$

where C is a constant. From this example we see evidently that the class \mathcal{P} includes the problem which can be solved by **DIV3-COMP**.

A different argument for \mathcal{P} to enclose polynomial-time computational problems can be given as follows. A computing device in the so-called von Neumann architecture (that is, the modern computer architecture we are familiar with, [Oxf91]) has a counter, a memory, and a central processor unit (CPU) which can perform one of the following basic instructions, called micro-instructions, at a time:

Load:	Loading the content in a memory location to a register (in CPU);
Store:	Storing the content of a register to a memory location;
Add:	Adding contents of two registers;
Comp:	Complementing the content of a register (for subtraction via “Add”);
Jump:	Setting the counter to a new value;
JumpZ:	“Jump” upon zero content of a register (for conditional branching);
Stop:	Terminating.

It is well known (see e.g., §1.4 of [AHU74]) that the above small set of micro-instructions is sufficient for constructing algorithms for solving arbitrary arithmetic problems on a von Neumann computer (however notice that by “arbitrary arithmetic problems” we do not mean to consider instances of arbitrary sizes, we will further discuss this in a moment). It can be shown (e.g., Theorem 1.3 in [AHU74]) that each micro-instruction in the above set can be simulated by a Turing machine in polynomial time. Consequently, a problem that can be solved in polynomial time on a von Neumann computer (which implies that the number of micro-instructions used in the algorithm must be a polynomial in the size of the input to the algorithm) can also be solved by a Turing machine in polynomial time. This is because for any polynomials $p(n)$ and $q(n)$, any ways of arithmetic combining $p(n)$, $q(n)$, $p(q(n))$ and $q(p(n))$ will result in a polynomial in n . Notice that we have deliberately excluded multiplication and division from our (simplified) set of micro-instructions. A multiplication between numbers of size n can be done via n additions and hence has its total cost should be measured by $n \times \text{cost}(\text{Add})$. Division has the same cost as

multiplication since it is repeated subtraction which is addition of a complementary number.

We should mention an unimportant difference between the computation model based on Turing machines and that based on von Neumann computers. By Definition 4.1, we regard a problem to be solvable on a Turing machine only if *any* instance is solvable on the *same* machine (“one machine to solve them all!”). The cost for solving a problem on a Turing machine is measured by the size of the problem in an *uniform* manner across the whole spectrum of the size of the problem. Machine DIV3 shows this evidently. Due to this property in cost measurement we say that the Turing-machine-based computation model is *uniform*. In contrast, registers and logical circuits which are the basic building blocks of a von Neumann computer have finite sizes. As a result, problems solvable on a von Neumann computer must also have a finite size: for a same problem, the bigger an instance is, the bigger a machine is needed for solving it. In general, machines of different sizes do not agree on an uniform measurement on the cost for solving the same problem. We therefore say that a circuit-based computation model (upon which a von Neumann computer bases) is *non-uniform*. However, so far, the difference between the uniform and non-uniform measurements on the cost of computations has not created any new complexity class, or caused any known classes to collapse. That is why we say this difference is not significant.

In the rest of this chapter we shall often neglect the difference between a decisional problem and a computational problem, and the difference among a Turing machine, a modern computer, a procedure, or an algorithm. Decisional computational problems will be generally called problems, while machines, computers, procedures or algorithms will be generally referred to as methods or algorithms. Occasionally, we will return to describing a language recognition problem, and only then we will return to using Turing machines as our basic instrument of computation.

4.5 Algorithms and Computational Complexity Expressions

Let us now study three very useful polynomial-time algorithms. Through the study of these algorithms, we shall (i) get familiar with a programming language which we shall use to write algorithms and protocols in this book, (ii) agree on some notation and convention for expressing computational complexity for algorithms and protocols, and (iii) establish the time complexities for a number of arithmetic operations which will be most frequently used in cryptography.

Above we have explained that Turing machines provide us with a general model of computation and with a precise notion for measuring the computational complexity for procedures. However, we do not generally wish to describe algorithms in terms of such a primitive machine, not even in terms of the micro-instructions of a modern computer (i.e., the set of instructions we described in § 4.4). In order to

describe algorithms and mathematical statements effectively and clearly, we shall use a high-level programming language called **Pseudo Programming Language (PPL)** which is very close to a number of popular high-level programming languages such as Pascal or C and can be understood without any difficulty due to its plainly self-explanatory feature.

4.5.1 Greatest Common Divisor

The first algorithm we shall study is the famous algorithm of Euclid for computing greatest common divisor (Algorithm 4.1). Denoting by $\text{gcd}(a, b)$ the greatest common divisor of integers a and b , $\text{gcd}(a, b)$ is defined to be the largest integer that divides both a and b .

Algorithm 4.1: Euclid's Algorithm for Greatest Common Divisor

INPUT Integers $a > b \geq 0$;

OUTPUT $\text{gcd}(a, b)$.

1. if ($b == 0$) return(a);
2. return($\text{gcd}(b, a \bmod b)$).

Figure 4.2.

In Algorithm 4.1, “ $a \bmod b$ ” denotes the remainder of a divided by b . (In §4.5.5 we will formally define the modular operation and provide some useful facts on modular arithmetic.) The condition $a > b \geq 0$ is merely for the purpose of ease of exposition. In the implementation, this condition can be satisfied by replacing a, b with their absolute values, and by invoking $\text{gcd}(|b|, |a|)$ in case $|a| < |b|$.

Now let us examine how Algorithm 4.1 works. For positive integers $a \geq b$, we can always write

$$a = bq + r \tag{4.5.1}$$

for some integer $q \neq 0$ (the quotient of a divided by b) and $0 \leq r < b$ (the remainder of a divided by b). Since by definition, $\text{gcd}(a, b)$ divides both a and b , equation (4.5.1) shows that it must also divide r too. Consequently, $\text{gcd}(a, b)$ equals $\text{gcd}(b, r)$. Since the remainder r (of a divided by b) is denoted by $a \bmod b$, we have derived

$$\text{gcd}(a, b) = \text{gcd}(b, a \bmod b).$$

This is the fact we have used in Algorithm 4.1, namely, $\text{gcd}(a, b)$ is defined by $\text{gcd}(b, a \bmod b)$ recursively. The series of recursive calls of gcd compute the following

series of equations, each is in the form of (4.5.1) and is formed by a division between the two input values:

$$\begin{aligned}
 a &= bq_1 + r_1 \\
 b &= r_1q_2 + r_2 \\
 r_1 &= r_2q_3 + r_3 \\
 &\vdots \\
 &\vdots \\
 &\vdots \\
 r_{k-3} &= r_{k-2}q_{k-1} + r_{k-1} \\
 r_{k-2} &= r_{k-1}q_k + r_k
 \end{aligned} \tag{4.5.2}$$

where $r_k = 0$ (which causes the terminating condition in step 1 being met) and $q_1, q_2, \dots, q_k, r_1, r_2, \dots, r_{k-1}$ are non-zero integers. With $r_k = 0$, the last equation in (4.5.2) means r_{k-1} divides r_{k-2} , and in the last-but-one equation, it must also divide r_{k-3}, \dots , eventually, as shown in the first equation in (4.5.2), r_{k-1} must divide both a and b . None of other remainders in other equations has this property (that's why they are called remainders, not a divisor, only r_{k-1} is a divisor in the last equation in (4.5.2)). Therefore, r_{k-1} is indeed the greatest common divisor of a and b , i.e., $r_{k-1} = \gcd(a, b)$.

For example, $\gcd(108, 42)$ will invoke the following sequence of recursive calls (where \leftarrow denotes “value assignment”):

$$\gcd(108, 42) \leftarrow \gcd(42, 24) \leftarrow \gcd(24, 18) \leftarrow \gcd(18, 6) \leftarrow \gcd(6, 0) \leftarrow 6.$$

4.5.2 Extended Euclid's Algorithm

Algorithm 4.1 has thrown away all the intermediate quotients. If we accumulate them during the computation of $\gcd(a, b)$, we can obtain something more than just $\gcd(a, b)$. Let us see what we can obtain.

The first equation in (4.5.2) can be written as

$$a + b(-q_1) = r_1.$$

Multiplying both sides of this equation with q_2 , we can obtain

$$aq_2 + b(-q_1q_2) = r_1q_2.$$

Using this equation and the second equation in (4.5.2), we can derive

$$a(-q_2) + b(1 + q_1q_2) = r_2. \tag{4.5.3}$$

The same way of calculation can be carried out. In general, for $i = 1, 2, \dots, k$, we can derive

$$a\lambda_i + b\mu_i = r_i \quad (4.5.4)$$

where λ_i, μ_i are some integers which are, as indicated in (4.5.3), certain form of accumulations of the intermediate quotients. We have seen in §4.5.1 that following this way of calculation we will eventually reach $r_k = 0$, and upon then we have

$$a\lambda_{k-1} + b\mu_{k-1} = r_{k-1} = \gcd(a, b). \quad (4.5.5)$$

An algorithm that inputs a, b and outputs the integers λ_{i-1}, μ_{k-1} satisfying (4.5.5) is called **Extended Euclid Algorithm**. Extended Euclid Algorithm will have an extensive use in the rest of the book for computing division modulo integers. Let us now specify this algorithm, that is, find a general method for accumulating the intermediate quotients.

Observe the equations in (4.5.2) and denote $r_{-1} = a, r_0 = b, \lambda_{-1} = 1, \mu_{-1} = 0, \lambda_0 = 0, \mu_0 = 1$. Then for $i = 1, 2, \dots, k-1$, the i th equation in (4.5.2) relates r_{i-1}, r_i and r_{i+1} by

$$r_{i+1} = r_{i-1} - r_i q_{i+1}. \quad (4.5.6)$$

Replacing r_{i-1} and r_i in the right-hand side of (4.5.6) using equation (4.5.4), we derive

$$r_{i+1} = a(\lambda_{i-1} - q_{i+1}\lambda_i) + b(\mu_{i-1} - q_{i+1}\mu_i). \quad (4.5.7)$$

Comparing between (4.5.7) and (4.5.4), we obtain (for $i = 0, 1, \dots, k-1$)

$$\begin{aligned} \lambda_{i+1} &= \lambda_{i-1} - q_{i+1}\lambda_i \\ \mu_{i+1} &= \mu_{i-1} - q_{i+1}\mu_i \end{aligned} \quad (4.5.8)$$

These two equations provide us with a general method for accumulating the intermediate quotients while computing greatest common divisor (see Algorithm 4.2).

Remark 4.1:

In order to present Algorithms 4.1 and 4.2 with easily understandable working principles, we have chosen to sacrifice the efficiency. In the next two sections we will analyse their time complexities and contrast our result with the best known time complexity result for computing greatest common divisor. \square

4.5.3 Time Complexity of Euclid's Algorithms

Let us now measure the time complexities for the two Euclid's algorithms. It is clear that the number of recursive calls in Algorithm 4.1 is equal to the number of loops in Algorithm 4.2 which is in turn equal to k in (4.5.2).

Algorithm 4.2: Extended Euclid's Algorithm

INPUT a, b : integers with $a > b \geq 0$;
 OUTPUT integers λ, μ satisfying $a\lambda + b\mu = \gcd(a, b)$.

1. $i \leftarrow 0$; $r_{-1} \leftarrow a$; $r_0 \leftarrow b$;
 $\lambda_{-1} \leftarrow 1$; $\mu_{-1} \leftarrow 0$; $\lambda_0 \leftarrow 0$; $\mu_0 \leftarrow 1$; (* initialize *)
2. while ($r_i = a\lambda_i + b\mu_i \neq 0$) do (* it always holds $a\lambda_i + b\mu_i = r_i$ *)
 - (a) $q \leftarrow r_{i-1} \div r_i$; (* \div denotes division in integers *)
 - (b) $\lambda_{i+1} \leftarrow \lambda_{i-1} - q\lambda_i$; $\mu_{i+1} \leftarrow \mu_{i-1} - q\mu_i$; (* sum up quotients *)
 - (c) $i \leftarrow i + 1$;
3. return($(\lambda_{i-1}, \mu_{i-1})$).

Figure 4.3.

Consider the case $a > b$ and observe (4.5.6) for $i = 0, 1, \dots, k-1$. We have either of the following two cases:

$$|r_i| < |r_{i-1}|, \quad (4.5.9)$$

or

$$|r_{i+1}| < |r_{i-1}|. \quad (4.5.10)$$

Further noticing $r_{i+1} < r_i$, so case (4.5.9) also implies case (4.5.10), that is, case (4.5.10) holds invariantly. This means that the maximum value for k is bounded by $2 \cdot |a|$. If we consider the modulo operation to be a basic operation which takes one unit of time, then the time complexity of gcd realized in Algorithm 4.1 is bounded by $2 \cdot |a|$. This is a linear function in the size of a .

Theorem 4.1: *Greatest common divisor $\gcd(a, b)$ can be computed by performing no more than $2 \max(|a|, |b|)$ modulo operations. Therefore, Algorithm 4.1 and Algorithm 4.2 terminate within $2 \max(|a|, |b|)$ loops.* \square

G. Lamé (1795–1870) was the first person who proved the first sentence in the statements of Theorem 4.1. It is considered to be the first theorem ever proved about the theory of computational complexity (page 35 of [LeV77]).

The series of equations in (4.5.2) formed by a series of divisions suggest an inherent *sequentiality* characteristic in the computation of greatest common divisor.

Since Euclid discovered his algorithm (i.e., Algorithm 4.1), no significant improvement has been found to cut short this seemingly necessary sequential process.

4.5.4 Two Expressions for Computational Complexity

When we measure the computational complexity for an algorithm, it is often difficult or unnecessarily to pinpoint exactly the constant coefficient in an expression that bounds the complexity measurement. **Order notation** allows us to ease the task of complexity measurement.

Definition 4.2: Order Notation *We write $O(f(n))$ to denote a function $g(n)$ such that there exists a constant $c > 0$ and a natural number N with $|g(n)| \leq c|f(n)|$ for all $n \geq N$.*

Using the notation $O()$ we can express the time complexities of Algorithm 4.1 and Algorithm 4.2 as $O(\log a)$. Notice that in this expression we have replaced $|a|$ with $\log a$ without explicitly giving the base of the logarithm (though we conventionally agree that the omitted base is natural base e). In fact, any base $b > 1$ will provide a correct measurement expression under the order notation (an exercise).

So far we have considered that computing one modulo operation costs one unit of time, that is, it has the time complexity $O(1)$. As a matter of fact, modulo operation “ $a \pmod b$ ” in the general case involves division $a \div b$, which is actually done in Algorithm 4.2 in order to keep the quotient. Therefore the time complexity of modulo operation, same as that of division, should depend on the sizes of the two operands. In practical terms (for the meaning of “practical”, see the end of §4.7), using $O(1)$ to represent the time for a division is too coarse for a sensible resource management.

A simple modification of the order notation is to measure an arithmetic in terms of **bitwise computation**. In bitwise computation, all variables have the values 0 or 1, and the operations used are logical rather than arithmetic: they are \wedge (for AND), \vee (for OR), \oplus (for XOR, i.e., “exclusive or”), and \neg (for NOT).

Definition 4.3: Bitwise Order Notation *We write $O_B()$ to denote $O()$ under the bitwise computation model.*

Under the bitwise model, addition and subtraction between two integers i and j take $\max(|i|, |j|)$ bitwise operations, i.e., $O_B(\max(|i|, |j|))$ time. Intuitively, multiplication and division between i and j take $|i| \cdot |j|$ bitwise operations, i.e., $O_B(|i| \cdot |j|)$ time. We should point out that a lower time complexity of $O_B(\log(i+j) \log \log(i+j))$ can be obtained for multiplication and division if the fast Fourier Transformation (FFT) method is used (see, e.g., Chapter 12 of [PTVF88]). However, this lower complexity is an asymptotic one which is associated with a much larger constant

coefficient (related to the cost of FFT) and may actually cause a higher complexity for operands having relatively small sizes (e.g., sizes for modern cryptographic use). Therefore in this book we shall not consider the FFT implemented multiplication and division. Consequently we shall only use the intuitive complexity measurement for multiplication and division.

Let us now express the time complexities of Algorithms 4.1 and 4.2 using the more precise bitwise order notation $O_B()$. In Theorem 4.1 we have obtained that for $a > b$, $\gcd(a, b)$ can be computed in $O(\log a)$ time. Given that the both input values are bounded by a , and that modulo operation or division cost $O_B((\log a)^2)$, the time complexities of Algorithms 4.1 and 4.2 are both $O_B((\log a)^3)$.

Now we should recall Remark 4.1: we have chosen to present these algorithms with easily understandable working principles by sacrificing the efficiency. As a matter of fact, our sacrifice on efficiency is rather large!

Careful realizations of these two algorithms should make use of the following two facts:

- i) modulo operation or division for creating $a = bq + r$ cost $O_B((\log a)(\log q))$;
- ii) quotients q_1, q_2, \dots, q_k in (4.5.2) satisfy

$$\sum_{i=1}^k \log q_i = \log \prod_{i=1}^k q_i \leq \log a. \quad (4.5.11)$$

Hence the total time for computing greatest common divisor, via a careful realization, can be bounded by

$$\sum_{i=1}^k O_B((\log a)(\log q_i)) \leq O_B((\log a)^2).$$

Careful realizations of the counterparts for Algorithm 4.1 and Algorithm 4.2 can be found in Chapter 1 of [Coh96].

In the rest of this book, we shall use the best known result $O_B((\log a)^2)$ for expressing the time complexity for computing greatest common divisor, either using Euclid's algorithm or the Extended Euclid's algorithm.

4.5.5 Modular Arithmetic

The second algorithm we shall study is one for computing **modular exponentiation**. Modular exponentiation is widely used in public-key cryptography. Let us first take a short course on modular arithmetic (readers who are familiar with the modular arithmetic can skip this section).

Definition 4.4: Modular Operation *Given integers x and $n > 1$, the operation “ $x \pmod{n}$ ” is the remainder of x divided by n , that is, a non-negative integer $r \in [0, n - 1]$ satisfying*

$$x = kn + r$$

for some integer k .

Theorem 4.2: (Properties of Modular Operation) *Let $x, y, n \neq 0$ be integers with $\gcd(y, n) = 1$. The modular operation has the following properties.*

1. $(x + y) \pmod{n} = [(x \pmod{n}) + (y \pmod{n})] \pmod{n}$;
2. $(-x) \pmod{n} = (n - x) \pmod{n} = n - (x \pmod{n})$;
3. $(x \cdot y) \pmod{n} = [(x \pmod{n}) \cdot (y \pmod{n})] \pmod{n}$;
4. Denote by $y^{-1} \pmod{n}$ the **multiplicative inverse** of y modulo n . It is a unique integer in $[1, n - 1]$ satisfying
 $(y \cdot y^{-1}) \pmod{n} = 1$.

Proof We shall only show 1 and 4 while leaving 2 and 3 as exercises.

We can write $x = kn + r$, $y = \ell n + s$ for $0 \leq r, s \leq n - 1$.

For 1, we have

$$\begin{aligned} (x + y) \pmod{n} &= [(kn + r) + (\ell n + s)] \pmod{n} \\ &= [(k + \ell)n + (r + s)] \pmod{n} \\ &= (r + s) \pmod{n} \\ &= [(x \pmod{n}) + (y \pmod{n})] \pmod{n} \end{aligned}$$

For 4, because $\gcd(y, n) = 1$, applying Extended Euclid Algorithm (Algorithm 4.2) on input y, n , we obtain integers λ and μ satisfying

$$y\lambda + n\mu = 1. \tag{4.5.12}$$

Without loss of generality, we have $\lambda < n$ because otherwise we can replace λ with $\lambda \pmod{n}$ and replace μ with $y\lambda + \mu$ for some k while keeping equation (4.5.12). By Definition 4.4, $y\lambda \pmod{n} = 1$. Therefore we have found $y^{-1} = \lambda < n$ as the multiplicative inverse of y modulo n . Below we show the uniqueness of y^{-1} in $[1, n - 1]$. Suppose there exists another multiplicative inverse of $y \pmod{n}$; denote it by $\lambda' \in [1, n - 1]$, $\lambda' \neq \lambda$. We have

$$y(\lambda - \lambda') \pmod{n} = 0,$$

i.e.,

$$y(\lambda - \lambda') = an, \quad (4.5.13)$$

for some integer a . We know $y = \ell n + 1$ for some integer ℓ . Therefore equation (4.5.13) is

$$(\ell n + 1)(\lambda - \lambda') = an,$$

or

$$\lambda - \lambda' = bn,$$

for some integer b . This contradicts to our assumption $\lambda, \lambda' \in [1, n-1]$, $\lambda \neq \lambda'$. \square

Same as in the case of division in rationals \mathbb{Q} , division by a number modulo n is defined to be multiplication with the inverse of the divisor, of course, this requires the existence of the inverse, just as in the case in \mathbb{Q} . Thus, for any y with $\gcd(y, n) = 1$, we write $x/y \bmod n$ for $xy^{-1} \bmod n$.

Since computing y^{-1} involves applying Extended Euclid Algorithm, it needs time $O_B((\log n)^2)$. Therefore the time complexity for division modulo n is $O_B((\log n)^2)$.

Theorem 4.2 shows that modular arithmetic is very similar to the integer arithmetic. It is easy to see that addition and multiplication obey the following laws of commutativity and associativity (where “ \circ ” denotes either addition or multiplication):

$$a \circ b \bmod n = b \circ a \bmod n \quad (\text{Commutativity})$$

$$a \circ (b \circ c) \bmod n = (a \circ b) \circ c \bmod n \quad (\text{Associativity})$$

Finally we should point out that, in the definition for the modular operation $x \bmod n$ (see Definition 4.4), the value of k (the quotient of x divided by n) is not an important element. Therefore in equation

$$x \bmod n = y \bmod n \quad (4.5.14)$$

we should not care whether x and y may differ by a multiple of n . In the sequel, the above equation will always be written as either

$$x \equiv y \pmod{n},$$

or

$$x \pmod{n} \equiv y.$$

We shall call this way of denoting equation (4.5.14) a **congruence** modulo n , or we say: x is congruent to y modulo n .

4.5.6 Modular Exponentiation

For $x, y < n$, modular exponentiation $x^y \pmod n$ follows the usual definition of exponentiation in integers as repeated multiplications of x to itself y times, but in terms of modulo n :

$$x^y \stackrel{\text{def}}{=} \underbrace{xx \cdots x}_y \pmod n.$$

Let $y \div 2$ denote y divided by 2 with truncation to integers, that is,

$$y \div 2 = \begin{cases} y/2 & \text{if } y \text{ is even} \\ (y-1)/2 & \text{if } y \text{ is odd} \end{cases}$$

Then applying the “Associativity Law” of modular multiplication, we have

$$x^y = \begin{cases} (x^2)^{y \div 2} & \text{if } y \text{ is even} \\ (x^2)^{y \div 2} x & \text{if } y \text{ is odd} \end{cases}$$

The above computation provides the well-known algorithm for realizing modular exponentiation called “repeated square-and-multiply”. The algorithm repeats the following process: Divide the exponent into 2, perform a squaring, and if the division yields an odd number then further perform a multiplication.

Algorithm 4.3: Modular Exponentiation

INPUT x, y, n : integers with $x > 0, y \geq 0, n > 1$;
 OUTPUT $x^y \pmod n$.

`mod_exp(x, y, n)`

1. if $(y == 0)$ return(1);
2. if $(y \pmod 2 == 0)$ return(`mod_exp(x2 (mod n), y ÷ 2, n)`);
3. return($x \cdot \text{mod_exp}(x^2 \pmod n, y \div 2, n) \pmod n$).

Figure 4.4.

We should point out the following feature in Algorithm 4.3 due to its recursive definition: The execution of a “return” statement implies that the subsequent step(s) following the “return” statement will never be executed. This is because the statement `return(“value”)` causes the program to go back, with “value”, to the point where the current call of `mod_exp` was made.

For example, starting from $\text{mod_exp}(2, 21, 23)$, Algorithm 4.3 will invoke the following five recursive calls (where \leftarrow denotes “return value to”)

$$\text{mod_exp}(2, 21, 23) \leftarrow 2 \cdot \text{mod_exp}(4(\equiv 2^2 \pmod{23}), 10, 23) \quad (\text{in step 3})$$

$$\text{mod_exp}(4, 10, 23) \leftarrow \text{mod_exp}(16(\equiv 4^2 \pmod{23}), 5, 23) \quad (\text{in step 2})$$

$$\text{mod_exp}(16, 5, 23) \leftarrow 16 \cdot \text{mod_exp}(3(\equiv 16^2 \pmod{23}), 2, 23) \quad (\text{in step 3})$$

$$\text{mod_exp}(3, 2, 23) \leftarrow \text{mod_exp}(9(\equiv 3^2 \pmod{23}), 1, 23) \quad (\text{in step 2})$$

$$\text{mod_exp}(9, 1, 23) \leftarrow 9 \cdot \text{mod_exp}(12(\equiv 9^2 \pmod{23}), 0, 23) \quad (\text{in step 3})$$

$$\text{mod_exp}(12, 0, 23) \leftarrow 1 \quad (\text{in step 1})$$

(Notice that the above six lines contain five recursive calls of mod_exp ; the final line “ $\text{mod_exp}(12, 0, 23) \leftarrow 1$ ” merely represents “return value 1” and is not a recursive call of the function.) The final value returned to $\text{mod_exp}(2, 21, 23)$ is 12 which is constructed from several multiplications made in step 3:

$$12 \equiv 2 \cdot 16 \cdot 9 \equiv 2^1 \cdot (2^2)^2 \cdot (((2^2)^2)^2) \pmod{23}.$$

Below we examine the time complexity of mod_exp realized in Algorithm 4.3. Since $y (> 0)$ can be divided into 2 exactly $\lfloor \log_2 y \rfloor + 1$ times to reach 0 as the quotient, a run of $\text{mod_exp}(x, y, n)$ will invoke exactly $\lfloor \log_2 y \rfloor + 1$ recursive calls of the function itself to reach the terminating condition in step 1 (zero exponent). Each recursive call consists of a squaring or a squaring plus a multiplication which costs $O_B((\log x)^2)$. Thus, considering x, y as numbers less than n , the time complexity for mod_exp realized in Algorithm 4.3 is bounded by $O_B((\log n)^3)$.

Similar to a seemingly unavoidable sequentiality in the computation of gcd , there is also an inherent sequentiality in the computation of mod_exp . This is seen as the following simple requirement in the repeated squaring: x^4 can only be computed after x^2 has been computed. Over the years, no significant progress has been made to improve the complexity from $O_B((\log n)^3)$ (without considering using FFT, review our discussion in 4.5.4).

Table 4.2 summarizes our examination on the time complexities for the basic modular arithmetic operations. We should notice that in the case of addition and subtraction, the modulo operation should not be considered to involve division; this

Operation for $a, b \in_U [1, n)$	Complexity
$a \pm b \pmod n$	$O_B(\log n)$
$a \cdot b \pmod n$	$O_B((\log n)^2)$
$b^{-1} \pmod n$	$O_B((\log n)^2)$
$a/b \pmod n$	$O_B((\log n)^2)$
$a^b \pmod n$	$O_B((\log n)^3)$

Table 4.2. Bitwise time complexities of the basic modular arithmetic operations

is because for $0 \leq a, b < n$, we have $-n < a \pm b < 2n$, and therefore

$$a \pm b \pmod n = \begin{cases} a \pm b & \text{if } 0 \leq a \pm b < n \\ a \pm b - n & \text{if } a \pm b \geq n \\ n + (a \pm b) & \text{if } a \pm b < 0 \end{cases}$$

4.6 Probabilistic Polynomial-Time

It is generally accepted that if a language is not in \mathcal{P} then there is no Turing machine that recognizes it *and* is *always* efficient^b. However, there is a class of languages with the following property: their membership in \mathcal{P} has not been proven, but they can *always* be recognized *efficiently* by a kind of Turing machines which may *sometimes* make mistakes.

The reason why such a machine may sometimes make a mistake is that in some step of its operation the machine will make a **random move**. While some random moves lead to a correct result, others may lead to an incorrect one. Such a Turing machine is called a **nondeterministic Turing machine**. However, the sub-class of decisional problems we are now introducing share the following **bounded error property**:

The probability for a random move made by a nondeterministic Turing

^bThe precise meaning for an “efficient machine” will be defined in §4.7; here we can roughly say that an efficient machine is a fast one.

machine, when answering a YES/NO question, to be correct is a non-negligible^c constant.

We conventionally call a nondeterministic Turing machine with a bounded error a **probabilistic Turing machine**. Because of this reason, the name “nondeterministic Turing machine” becomes reserved for a class of decisional problems which we will introduce in §4.8.

A probabilistic Turing machine also has a plural number of tapes. One of these tapes is called a **random-tape** which contains some uniformly distributed random symbols. During the scanning of an input instance I , the machine will also interact with the random-tape, pick up a random symbol and then proceed like a deterministic Turing machine. The random string is called the **random input** to a probabilistic Turing machine. With the involvement of the random input, the recognition of an input instance I by a probabilistic Turing machine is no longer a deterministic function of I , but is associated with a random variable, that is, a function of the machine’s random input. This random variable assigns certain **error probability** to the event of recognizing I .

The class of languages that are recognizable by probabilistic Turing machines is called **probabilistic polynomial-time (PPT)** languages, which we denote by \mathcal{PP} .

Definition 4.5: Class \mathcal{PP} *We write \mathcal{PP} to denote the class of languages with the following characteristics. A language L is in \mathcal{PP} if there exists a probabilistic Turing machine PM and a polynomial $p(n)$, such that PM recognizes any instance $I \in L$ with certain error probability which is a random variable of PM ’s random input, in time $T_{PM}(n)$ with $T_{PM}(n) \leq p(n)$ for all nonnegative integers n , where n is an integer parameter representing the size of the instance I .*

In Definition 4.5 we have left one element to be particularly vague, which is: “ PM recognizes $I \in L$, with certain error probability”. The “certain error probability” should be formulated into the following two expressions of conditional probability bounds:

$$\text{Prob}[PM \text{ recognizes } I \mid I \in L] \geq \epsilon, \quad (4.6.1)$$

and

$$\text{Prob}[PM \text{ recognizes } I \mid I \notin L] \leq \delta, \quad (4.6.2)$$

where ϵ and δ are any constants in the interval $[0, 1]$, and the probability space is the all possible random input values of PM .

The probability bound in (4.6.1) expresses the probability for a correct recognition of an instance, and that in (4.6.2) expresses the probability for a mistaken

^cThe exact meaning of non-negligible will be defined in §4.9.

recognition of a non-instance. The need for bounding the first probability from below is in order to limit the possibility for a mistaken “no recognition of an instance” which is obviously an error. To manifest this *error* probability more meaningfully, we can re-express (4.6.1) into the following equivalent expression:

$$\text{Prob}[PM \text{ is not decided whether or not } I \in L \mid I \in L] < 1 - \epsilon.$$

Notice that for probabilistic Turing machines we must not equate “no recognition” and “rejection”. Here “no recognition” may be due to an error caused by a random move of the machine and therefore it precisely means that the machine has failed to recognize the instance and *does not know* the answer. In contrast, “rejection” means knowing the answer $I \notin L$ definitely. For a language recognition problem, “rejection” can be answered by a deterministic Turing machine to mean “no recognition” precisely: a deterministic machine is definite about its answer since there is no possibility for any error to occur.

The class \mathcal{PP} has several subclasses which are defined by different ways to characterize the error-probability bound expressions in (4.6.1) and in (4.6.2), using different values of ϵ and δ , respectively. Let us now introduce these subclasses. We will exemplify each subclass with an algorithm. Similar to the case where a deterministic Turing machine simulates a polynomial-time algorithm, a probabilistic Turing machine simulates a **randomized (polynomial-time) algorithm**. Therefore, the algorithm examples shown in our introduction will not be limited to those for language recognition.

4.6.1 Subclass “Always Fast and Always Correct”

A subclass of \mathcal{PP} is named \mathcal{ZPP} (which stands for “Zero-sided-error Probabilistic Polynomial time”) if the error probability bounds in (4.6.1) and (4.6.2) have the following characterization: for any $L \in \mathcal{ZPP}$ there exists a randomized algorithm A such that for any instance I

$$\text{Prob}[A \text{ recognizes } I \mid I \in L] = 1$$

and

$$\text{Prob}[A \text{ recognizes } I \mid I \notin L] = 0.$$

This error-probability characterization means that a random operation in a randomized algorithm makes no error at all. So, at a first glance, \mathcal{ZPP} should have no difference from \mathcal{P} . However, there are a class of problems which can be solved by deterministic algorithms as well as by randomized algorithms, *both in polynomial time*; while the randomized algorithms can yield no error whatsoever, they are much quicker than their deterministic counterparts. We will provide an example for contrasting the time complexity in a moment.

\mathcal{ZPP} algorithms can find good applications in cryptography. In a later chapter we will provide a cryptosystem which has a \mathcal{ZPP} decryption algorithm, which solves a problem which is not clear at all to be solvable by a deterministic algorithm.

4.6.1.1 An example of “zero-sided-error” algorithm

Some randomized algorithms are so natural that all of us have already chosen to use them instead of using their deterministic counterparts. One of such algorithms we are all familiar with is a randomized process for looking-up someone’s phone number from a phone book. This algorithm is specified in Algorithm 4.4.

Algorithm 4.4: Searching Through Phone Book

```

INPUT    Name: a person’s name;
          Book: a phone book;
OUTPUT   The person’s phone number.

1. Repeat the following until Book has one page
   {
     (a) Open Book at a random page;
     (b) If Name occurs before the page, Book  $\leftarrow$  Earlier_pages(Book);
     (c) Else Book  $\leftarrow$  Later_pages(Book);
   }

2. Return( Phone number beside Name );
```

Figure 4.5. A Zero-sided-error (\mathcal{ZPP}) Algorithm

Clearly, the random operation in Algorithm 4.4 will not introduce any error to the output result. Therefore this is indeed a “zero-sided-error” randomized algorithm. For a phone book of N pages, Algorithm 4.4 will only need to execute $O(\log N)$ steps and find the page containing the name and the number. We should notice that a deterministic algorithm for “searching through phone book” will execute average $O(N)$ steps.

The reason why Algorithm 4.4 works so fast is that names in a phone book have been sorted alphabetically. We should notice that sorting is itself a \mathcal{ZPP} problem: “quick-sort” (see, e.g., pages 92–97 of [AHU74]) is a randomized sorting algorithm, can sort N elements in $(N \log N)$ steps, and its random operations will not introduce any error to the outcome result. In contrast, “bubble-sort” is a deterministic sorting

algorithm; it sorts N elements in (N^2) steps (see e.g., pages 77 of [AHU74]).

We can say that \mathcal{ZPP} is a subclass of languages which can be recognized by randomized algorithms in an “always fast and always correct” fashion.

4.6.2 Subclass “Always Fast and Probably Correct”

A subclass of \mathcal{PP} which we name \mathcal{RP} -(Monte Carlo) (where “ \mathcal{RP} ” stands for “Randomized Polynomial time” and “Monte Carlo” is typically used as a generic term for “randomized”) if the error probability bounds in (4.6.1) and (4.6.2) have the following characterization: for any $L \in \mathcal{RP}$ -(Monte Carlo) there exists a randomized algorithm A such that for any instance I

$$\text{Prob}[A \text{ recognizes } I \mid I \in L] = 1,$$

and

$$\text{Prob}[A \text{ recognizes } I \mid I \notin L] \leq \delta.$$

Randomized algorithms with this error-probability characterization are called **Monte-Carlo algorithms**. Recall that δ is any constant in the interval $(0, 1)$.

Let us first argue that the imprecision of the probability bound will not cause any trouble in terms of characterizing this subclass.

Theorem 4.3: *Let x be recognizable as an instance of a language $L \in \mathcal{PP}$ by a Monte-Carlo algorithm A such that $A(x)$ has a running time bounded by a polynomial $p(|x|)$ and*

$$\text{Prob}[A \text{ recognizes } x \mid x \notin L] \leq \delta$$

with $0 < \delta < 1$. Show that there exists a randomized polynomial-time algorithm B such that

$$\text{Prob}[B \text{ recognizes } x \mid x \notin L] \leq 2^{-|x|}. \quad (4.6.3)$$

Proof We construct a randomized algorithm B from A .

The uniformity of A ’s random input means that the conditional event

$$“A \text{ recognizes } x \mid x \notin L”$$

occurred in any run of $A(x)$ is independent from the event occurred in a different run of $A(x)$. Running $A(x)$ k times and apply “Multiplication Rule 2” (in page 56), we have

$$\text{Prob} \left[\bigcap_{i=1}^k A \text{ recognizes } x \mid x \notin L \right] \leq \delta^k.$$

Let $k = \lfloor \frac{-|x|}{\log_2 \delta} \rfloor + 1$ and let $B(x)$ be k runnings of $A(x)$. Then the inequality (4.6.3) is satisfied and the running time of $B(x)$ is bounded by $(\lfloor \frac{-|x|}{\log_2 \delta} \rfloor + 1)p(|x|)$ which is clearly a polynomial in $|x|$. \square

From Theorem 4.3 we know that the error probability of a Monte-Carlo algorithm can be reduced to arbitrarily small by iterating the algorithm while the iterated algorithm remains in polynomial time. We therefore say that a Monte-Carlo algorithm is always fast and is probably correct.

We now show that PRIMES (the set of all prime numbers) is in the subclass \mathcal{RP} -(Monte Carlo).

4.6.2.1 An example of Monte-Carlo algorithm

Since Fermat, it has been known that if p is a prime number and x is relatively prime to p , then $x^{p-1} \equiv 1 \pmod{p}$. This forms a basis for the following Monte-Carlo method for primality test ([SS77]), that is, picking $x \in_U (1, p-1]$ with $\gcd(x, p) = 1$ and checking

$$x^{(p-1)/2} \stackrel{?}{\equiv} \pm 1 \pmod{p}. \quad (4.6.4)$$

The test is repeated $k = \log_2 p$ times with the -1 case occurring at least once. Below is the specification of this test algorithm.

Algorithm 4.5: Probabilistic Primality Test

INPUT p : a positive integer;
 OUTPUT YES if p is prime, NO otherwise.

Prime_Test(p)

1. repeat $\log_2 p$ times:
 - (a) $x \in_U (1, p-1]$;
 - (b) if ($\gcd(x, p) > 1$ or $x^{(p-1)/2} \not\equiv \pm 1 \pmod{p}$) return(NO);
 end_of_repeat;
2. if (test in 1.(b) never shows -1 case) return(NO);
3. return(YES).

Figure 4.6. A One-sided-error (Monte-Carlo) Algorithm

First of all, we know from **Fermat's Little Theorem** (Theorem 6.10) that if p is prime then for all $x < p$:

$$x^{p-1} \equiv 1 \pmod{p}. \quad (4.6.5)$$

So if p is prime then $\text{Prime_Test}(p)$ will always return YES, that is, we always have

$$\text{Prob} \left[x^{(p-1)/2} \equiv \pm 1 \pmod{p} \mid p \text{ is prime} \right] = 1.$$

On the other hand, if p is a composite number then congruence (4.6.4) will not hold in general. In fact (a fact in Group Theory, see Example 5.2.3 and Theorem 5.1) if the inequality against congruence (4.6.4) shows for one $x < p$ with $\gcd(x, p) = 1$ then the inequality must show for at least half the numbers of this kind. Thus we conclude that for $x \in_U (1, p-1]$ with $\gcd(x, p) = 1$:

$$\text{Prob} \left[x^{(p-1)/2} \equiv \pm 1 \pmod{p} \mid p \text{ is composite} \right] \leq 1/2. \quad (4.6.6)$$

Therefore, if the test passes k times for x chosen at uniformly random (remember that the -1 case is seen to hold at least once), then the probability that p is not prime is less than 2^{-k} . We have set $k = \log_2 p$, and so any input instance p :

$$\text{Prob}[\text{Prime_Test}(p) = \text{YES} \mid p \text{ is not prime}] \leq 2^{-\log_2 p}.$$

In §4.3 we have seen that computing modulo exponentiation and computing the greatest common divisor with $\log_2 p$ -bit long input value have their time complexities bounded by $O_B((\log_2 p)^3)$. Therefore the time complexity of $\text{Prime_Test}(p)$ is bounded by $O_B((\log p)^4)$.

To this end we know that PRIMES – the language of all prime numbers – is in \mathcal{RP} -(Monte Carlo).

However, without invalidating the above declaration, in August 2002, three Indian computer scientists, Agrawal, Kayal and Saena, find a *deterministic* polynomial-time primality test algorithm [AKS02]; consequently, PRIMES is in fact in \mathcal{P} .

4.6.3 Subclass “Probably Fast and Always Correct”

A subclass of \mathcal{PP} which we name \mathcal{RP} -(Las Vegas) if the error probability bounds in (4.6.1) and (4.6.2) have the following characterization: for any $L \in \mathcal{RP}$ -(Las Vegas) there exists a randomized algorithm A such that for any instance I

$$\text{Prob}[A \text{ recognizes } I \mid I \in L] \geq \epsilon,$$

and

$$\text{Prob}[A \text{ recognizes } I \mid I \notin L] = 0.$$

Randomized algorithms with this error-probability characterization are called **Las-Vegas algorithms**. This term, introduced in [Bab79], refers to randomized algorithms which either give the correct answer or no answer at all. Recall that ϵ is any constant in the interval $(0, 1)$.

Let us first argue that the imprecision of the probability bound will not cause any trouble in terms of characterizing this subclass.

Theorem 4.4: *Let x be recognizable as an instance of a language $L \in \mathcal{PP}$ by a Las-Vegas algorithm A such that $A(x)$ has a running time bounded by a polynomial $p(|x|)$ and recognizes x with probability*

$$\text{Prob}[A \text{ recognizes } x \mid x \in L] \geq \epsilon$$

with $0 < \epsilon < 1$. Show that there exists a randomized polynomial-time algorithm B such that

$$\text{Prob}[B \text{ recognizes } x \mid x \in L] \geq 1 - 2^{-|x|}. \quad (4.6.7)$$

Proof We construct a randomized algorithm B from A .

Suppose that when running $A(x)$, the following event occurs:

$$A \text{ is not decided whether or not } x \in L \mid x \in L.$$

From the given error probability bound of A we know

$$\text{Prob}[A \text{ is not decided whether or not } x \in L \mid x \in L] < 1 - \epsilon.$$

The uniformity of A 's random input means that if these conditional event occurs in different runs of A independently from each other. Running $A(x)$ k times and apply "Multiplication Rule 2", we have

$$\text{Prob} \left[\bigcap_{i=1}^k A \text{ is not decided whether or not } x \in L \mid x \in L \right] < (1 - \epsilon)^k.$$

Let $k = \lfloor \frac{-|x|}{\log_2(1-\epsilon)} \rfloor + 1$ and let $B(x)$ be k runnings of $A(x)$. Then we have

$$\text{Prob}[B \text{ is not decided whether or not } x \in L \mid x \in L] < 2^{-|x|},$$

that is, inequality (4.6.7) is satisfied. The running time of $B(x)$ is bounded by $(\lfloor \frac{-|x|}{\log_2(1-\epsilon)} \rfloor + 1)p(|x|)$ which is clearly a polynomial in $|x|$. \square

From Theorem 4.4 we know that the probability for a Las-Vegas algorithm to give no answer to a true instance can be reduced to arbitrarily small by iterating the algorithm while the iterated algorithm remains in polynomial time. If we say that a Monte-Carlo algorithm is always fast and probably correct, then a Las-Vegas algorithm is always correct and probably fast.

4.6.3.1 An example of Las-Vegas algorithm

Let p be an odd positive integer and let $p - 1 = q_1 q_2 \cdots q_k$ as the complete prime factorization of $p - 1$ (some of the prime factors may repeat). In Chapter 5 we will establish a fact (Theorem 5.12) that for p to be prime if and only if there exists a positive integer $g \in [2, p - 1]$ such that

$$\begin{aligned} g^{p-1} &\equiv 1 \pmod{p} \\ g^{(p-1)/r_i} &\not\equiv 1 \pmod{p} \text{ for } i = 1, 2, \dots, k. \end{aligned} \quad (4.6.8)$$

This fact provides us with an algorithm for proving primality. Inputting an odd number p and the complete prime factorization of $p - 1$, the algorithm tries to find a number g satisfying (4.6.8). If such a number is found, the algorithm outputs YES and terminates successfully, and p must be prime. Otherwise, the algorithm will be in an undecided state; this means, it does not know if p is prime or not. The algorithm is specified in Algorithm 4.6.

Algorithm 4.6: Proof of Primality

INPUT p : an odd positive number;
 q_1, q_2, \dots, q_k : all prime factors of $p - 1$;

OUTPUT YES if p is prime, NO otherwise;
 NO_DECISION with certain probability of error.

1. pick $g \in_U [2, p - 1]$;
2. for ($i = 1, i++, k$) do
 if ($g^{(p-1)/q_i} \equiv 1 \pmod{p}$) output NO_DECISION and terminate;
3. if ($g^{p-1} \not\equiv 1 \pmod{p}$) output NO and terminate;
4. output YES and terminate.

Figure 4.7. A One-sided-error (Las-Vegas) Algorithm

First we notice that because $k \leq \log_2 p - 1$, it is trivial to see that Algorithm 4.6 terminates in time polynomial in the size of p .

From the fact to be established in Theorem 5.12, it is clear that if Algorithm 4.6 outputs YES, then the input integer p is definitely prime; no error is possible. Also, if the algorithm outputs NO, the answer is also correct since otherwise the Fermat's Little Theorem (4.6.5) will be violated. These two cases reflect the algorithm's "always correct" nature. The error-free property of the algorithm entitles it to be named Proof of Primality.

However, when Algorithm 4.6 outputs NO_DECISION, it does not know if the input integer p is prime or not. It is possible that p not prime, but it is also possible that an error has occurred. In the latter case p is indeed prime, but the testing number g which the algorithm picks at random is a wrong one. When we have studied Theorem 5.12 in Chapter 5 we will know that the wrong number g is not a “primitive root”.

To this end we know that Algorithm 4.6 is a one-sided-error Las-Vegas algorithm. We may revise the algorithm into one which does not terminate at a NO_DECISION answer, but carries on the testing step by picking another random tester g . The modified algorithm is still a Las-Vegas algorithm, and becomes “probably fast” since it’s possible that it always picks a non-primitive root as a tester. Fortunately, for any odd prime, the multiplicative group \mathbb{Z}_p^* contains plenty of primitive roots (to be picked with a non-trivial probability; we will establish the proportion of primitive roots in Chapter 5).

Recall our mentioning in the beginning of this chapter of the quantum computation model: a quantum computer can factor an integer in time polynomial in the size of the integer (i.e., FACTORIZATION $\in \mathcal{QP}$). Shor devised such a method (see, e.g., pages 108–115 of [WC00]). Shor’s quantum factorization procedure is also a Las-Vegas algorithm. To factor an integer N , a random element a modulo N is picked and a quantum computer will find the period of the function $f(x) = a^x \pmod{N}$, i.e., an even positive integer b returned from the quantum computer such that $f(b) = 1$. In Chapter 6 we shall see that for a composite N , a non-trivial proportion of integers $a < N$ has an even period. Let r be the largest odd factor of b . If $a^r \not\equiv \pm 1 \pmod{N}$, then $\gcd(a^r \pm 1, N)$ is a non-trivial factor of N , i.e., the algorithm has successfully factored N . If $a^r \equiv \pm 1 \pmod{N}$, then $\gcd(a^r \pm 1, N)$ is a trivial factor of N , i.e., 1 or N ; so the algorithm fails with no answer.

However, for randomly chosen element a , the probability for encountering $a^r = \pm 1 \pmod{N}$ is bounded, and therefore the procedure can be repeated using another random element a . By Theorem 4.4, Shor’s algorithm remains in polynomial time.

Las-Vegas algorithms and Monte-Carlo algorithms together are referred to as “randomized algorithms with one-sided error”. The subclass of languages recognizable by randomized algorithms with one-sided error is denoted by \mathcal{RP} .

4.6.4 Subclass “Probably Fast and Probably Correct”

A subclass of \mathcal{PP} is named \mathcal{BPP} (which stands for “**B**ounded error probability **P**robabilistic **P**olynomial time”) if the error probability bounds in (4.6.1) and (4.6.2) both hold for ϵ and δ being any constant in $(0, 1)$. Randomized algorithms with this error probability characterization are referred to as having “two-sided error”.

We know from Theorem 4.3 and Theorem 4.4 that these two error probabilities can be reduced to arbitrarily small if a “two-sided-error” algorithm is iterated suffi-

ciently many times. An example to be given here will provide a concrete illustration for this fact.

4.6.4.1 An example of “two-sided-error algorithm”

There is a famous protocol in **quantum cryptography** named the **quantum key distribution protocol** (the QKD protocol, see e.g. [BB89]). The QKD protocol allows a bit string to be transmitted from one communication entity to another without having the two parties to meet face to face, and yet that the two parties can be sure with a high confidence that the distributed bit string is exclusively shared between them. The QKD protocol is a two-sided-error randomized algorithm. Let us describe this algorithm and examine its two-sided-error property.

Let us first provide a brief description on the physics principle for the QKD protocol. The distribution of a secret bit string in the QKD protocol is achieved by a sender (let Alice be the sender) transmitting a string of four-way-polarized photons. Each of these photons is in a state (called a photon state or a state) denoted by one of the four following symbols:

$$—, |, /, \backslash.$$

The first two photon states are emitted by a polarizer which is set with a rectilinear orientation; the latter two states are emitted by a polarizer which is set with a diagonal orientation. Let us denote by $+$ and \times these two differently oriented polarizers, respectively. We can encode information into these four photon states. The following is a bit-to-photon encoding scheme:

$$+(0) = —, +(1) = |, \times(0) = /, \times(1) = \backslash. \quad (4.6.9)$$

This encoding scheme is the public knowledge. If Alice wants to transmit the bit 0 (respectively, 1), she may choose to use $+$ and consequently send out $—$ (respectively, $|$), or choose to use \times and consequently send out $/$ (respectively, \backslash). For each bit to be transmitted in the QKD protocol Alice will choose to use $+$ or \times at uniformly random.

To receive a photon state, a receiver (who may be Bob, the intended receiver, or Eve, an eavesdropper) must use a device called a photon observer which is also set with rectilinear or diagonal orientations. We shall also denote by $+$ and \times these two differently oriented devices, respectively. The states $—$ and $|$ can be observed by $+$ correctly. Likewise, $/$ and \backslash can be observed by \times correctly. However, if $—$ or $|$ are observed using \times then there will be a 50:50-chance for the receiver to see tiny $/$ or \backslash . We may imagine this phenomenon to be that the transmitted state is “rectified” in the receiver’s device by 45° clockwise or anti-clockwise, 50:50 chance either way. Clearly, both of these observations are incorrect. Likewise, if $/$

or \diagdown are observed using $+$ then the receiver will have a 50:50-chance to see $—$ or $|$; again, both observation results are wrong. These wrong observations are an inevitable result of the well-known “Heisenberg Uncertainty Principle”, which underlies the working principle for the QKD Protocol.

If the orientation setting of the receiver’s device agrees with that of Alice’s device then the public bit-to-photon encoding scheme in (4.6.9) is a 1-1 mapping between the binary bits and the photon states, that is, the bit sent by Alice can be correctly received. On the other hand, a wrong observation caused by an incorrect orientation setting will necessarily destroy a photon state while the receiver will have no idea which photon state has actually been sent and destroyed.

Now we are ready to specify the QKD Protocol. For presentation simplicity, we describe a version in which Alice and Bob try to agree on a single bit as a shared secret. The protocol is specified in Protocol 4.1.

Protocol 4.1: Quantum Key Distribution

What is to be achieved?

- I) Alice transmits a secret bit to Bob;
- II) $\text{Prob}[\text{eavesdropper being detected} \mid \text{existing eavesdropper}] = 1/4$.
 1. Alice generates two random bits $a_1, a_2 \in_U \{0, 1\}$; she further picks two randomly oriented polarizers $p_1, p_2 \in_U \{+, \times\}$; she sends to Bob two photon states $p_1(a_1), p_2(a_2)$ according to the bit-to-photon encoding scheme in (4.6.9);
 2. Bob picks two randomly oriented photon observers $o_1, o_2 \in_U \{+, \times\}$ and receives two photon states; he tells Alice: “All received”;
 3. Alice sends to Bob: p_1, p_2 ;
 4. If $p_1 \neq o_1$ and $p_2 \neq o_2$, the run fails;
 5. (* From now on $p_1 = o_1$ and $p_2 = o_2$ *) Bob uses the bit-to-photon encoding scheme in (4.6.9) to obtain b_1, b_2 from the received states;
 6. Alice and Bob compare the test bit a_2 and b_2 ; if $a_2 \neq b_2$, they announce “Eavesdropper detected!”;
 7. The run terminates successfully with $a_1 = b_1$ being the shared bit.

Figure 4.8. A Two-sided-error (\mathcal{BPP}) Algorithm

Let us explain how this protocol works.

Steps 1, 2 and 3 are quite straightforward: Alice sends to Bob two photon states (Step 1); Bob has to observe them in a random process (Step 2), and then Alice can tell Bob if both of his observations are correct (Step 3).

Step 4: if any of Bob devices is set incorrectly then wrong state(s) will be received. The wrong state(s) received cannot serve the purposes of (i) transmitting a correct bit (if the first state cannot be correctly received), and/or (ii) detecting Eve (if the second state cannot be correctly received). Case (i) means that Alice and Bob will fail to agree a bit; case (ii) means that they will have no idea whether or not Eve is listening between them. The protocol run fails in either or both of these cases. We notice that this is an *error* of this protocol in *one side*. We can consider that this sided error is caused by Bob's random move. We know that the probability for Bob to have the correct setting is $1/4$, that is, the probability for this sided error to occur is $3/4$.

Step 5: Bob now knows that he has set both of his observers correctly. If Eve is not overhearing the communications then Bob should have received both of Alice's bits correctly since now the bit-to-photon encoding scheme in (4.6.9) is a 1-1 mapping. From now on we must remember that Alice and Bob share the same settings for their respective devices, and consequently, they should share the both bits transmitted *unless* Eve is working between them.

Step 6: Alice and Bob now test if $a_2 = b_2$. Inequality necessarily implies the existence of Eve since otherwise the both bits must be received correctly, a contrary to the inequality.

Step 7: To reach this step is to reach a happy ending of this protocol. Now the question is: if the channel has been eavesdropped, what is the probability for reaching this happy ending? In other words, what is the probability for Eve not being detected? Let us measure it.

The only way for Eve to observe the photon states sent from Alice is to use the same technique that Bob uses. So Eve has to pick random orientation for her observers, and then passes to Bob whatever state she has observed. Of course, Eve may choose to send to Bob completely new states of her own invention, but doing so will not help her to gain anything. Notice that since Alice and Bob only test the second bit, we only need to consider the orientation of the second observer that Eve has chosen.

If Eve has set her second observer correctly then she will receive a_2 correctly, and consequently there is no way for Alice and Bob to detect her. Since the probability for Eve to set her second observers correctly is $1/2$, we have so far $1/2$ as part of the probability value for non-detection.

If Eve has set her second observer incorrectly then the second state sent to Bob from Eve will be a wrong one. Nevertheless, Bob's observer will "rectify" the wrong state by 45° clockwise or anti-clockwise, 50:50 chance either way. Thus, Bob

may receive the second state correctly or incorrectly with either case having the probability $1/2$. A correct receipt will again leave Eve undetected. Notice that this un-detection is after Eve's wrong setting of her device which also has the probability $1/2$. If we conjunct the probabilities for Eve's wrong setting of her device and for Bob's correct receipt of the second bit, this case will add further $1/4$ on to the probability value for non-detection.

To this end we know that the probability for Eve not being detected is $3/4$. We notice that this is an *error* of this protocol in *the other side*. We can consider that this sided error is caused by Alice's random move (by considering Eve always fixing the orientation settings for her observers).

In real application of the QKD protocol, Alice should transmit to Bob sufficiently many, say n , photon states. Bob should find sufficiently many, say m ($m < n$), places of agreed settings for their respective devices. The two parties should then test sufficiently many, say ℓ ($\ell < m$), testing bits. In this way, they can achieve sharing of $m - \ell$ secret bits and they will have the probability $1 - (3/4)^\ell$ for detecting Eve. This value approaches 1 very quickly.

4.7 Efficient Algorithms

To this end of our introduction to the polynomial-time class and to the probabilistic polynomial-time (PPT) subclasses, we have established the following class inclusion relation:

$$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \begin{array}{c} \mathcal{RP}\text{-(Monte Carlo)} \\ \mathcal{RP}\text{-(Las Vegas)} \end{array} \subseteq \mathcal{BPP} \subseteq \mathcal{PP}.$$

Algorithms which can solve problems in any of these classes are called efficient algorithms.

Definition 4.6: Efficient Algorithms *An algorithm is said to be efficient if it is deterministic or randomized with execution time expressed by a polynomial in the size of the input.*

This definition characterizes a notion of **Turing computability**: whether deterministic or randomized, a polynomial-time problem is solvable, i.e., such a problem requires resources which are manageable even if the size of the problem can be very large.

However, since polynomials can have vastly different degrees, within \mathcal{P} or \mathcal{PP} , problems have vastly different time complexities. Therefore an efficient algorithm for solving a Turing computable problem need not be efficient in a practical sense. We will see a few protocol examples in §??, which have their time complexities bounded by polynomials in their input sizes (and hence, they are efficient by Definition 4.6); however, these protocols have little value for practical use because the

polynomials that bound their time complexities are simply too large (i.e., their degrees are too large). This is in contrast to the situations in applications where some algorithms with non-polynomial (to be defined in §4.9) time complexities are used for solving small instances of Turing-intractable problems effectively.

We shall use term **practical efficiency** to refer to polynomial-time algorithms where the polynomials have very small degrees. For example, Turing machine Div3 and algorithms gcd, mod_exp and Prime_Test are all practically efficient. Now let us see another example of a practically efficient algorithm which is widely used in modern cryptography.

An Example of Efficient Algorithm: Probabilistic Prime Generation

The idea of probabilistic primality test can be translated straightforwardly to an algorithm for generating a random **probabilistic prime** number of a given size. We say a number n probabilistic prime number if Prime_Test(n) returns the YES answer. Below is the specification of such an algorithm.

Algorithm 4.7: Random k -bit Probabilistic Prime Generation

INPUT k : a positive integer;
 (* the input is encoded to have the size of the input *)
 OUTPUT a k -bit random prime.

Prime_Gen(k)

1. $p \in_U (2^{k-1}, 2^k - 1]$ with p odd;
2. if (Prime_Test(p) == NO) return(Prime_Gen(k));
3. return(p).

Figure 4.9. A Probabilistic Algorithm for Prime Number Generation

First, let us suppose that Prime_Gen(k) terminates. This means that the algorithm eventually finds a number p which satisfies Prime_Test(p) == YES (in step 2). From our estimate on the error probability bound for Prime_Test, the probability for the output p not to be prime is bounded above by 2^{-k} where $k = \log_2 p$.

An obvious question arises: Will Prime_Gen(k) terminate at all?

The well-known prime number theorem (see e.g., page 28 of [Kra86]) states that the number of primes less than X is bounded below by $\frac{X}{\log X}$. So the number of

primes of exactly k binary bits is about

$$\frac{2^k}{k} - \frac{2^{k-1}}{k-1} \approx \frac{2^k}{2k}.$$

Thus, we can *expect* that $\text{Prime_Gen}(k)$ may recursively call itself $2k$ times in step 2 until a probabilistic prime is found, and then it terminates.

With the time complexity for $\text{Prime_Test}(p)$ being bounded by $O_B((\log p)^4) = O_B(k^4)$, after $2k$ calls of Prime_Test , the time complexity of $\text{Prime_Gen}(k)$ is bounded by $O_B(k^5)$.

Another question arises: While $O_B(k^5)$ is indeed a polynomial in k , can it be a polynomial in the *size* of k , which is the input to Algorithm $\text{Prime_Gen}(k)$?

When we encode a number n in the base- b representation for any $b > 1$, the size of the number n is $\log_b n$ and is always less than n . In order to make $\text{Prime_Gen}(k)$ a polynomial-time algorithm in the size of its input, we have explicitly required in the specification of $\text{Prime_Gen}(k)$ that its input should be encoded to have the size of the input. Using the **unary encoding** method, or base-1 representation, k can indeed be encoded to have the size k .

Definition 4.7: Unary Encoding of a Number *The unary encoding of a positive natural number n is*

$$1^n = \underbrace{11 \cdots 1}_n.$$

From now on we shall use $\text{Prime_Gen}(1^k)$ to denote an invocation instance of the algorithm Prime_Gen . In the rest of this book, the unary encoding of a number always provides an explicit emphasis that the size of that number is the number itself.

4.8 Nondeterministic Polynomial-Time

Consider the following decisional problem:

Problem Square Freeness

INPUT N : a positive and odd composite integer;

QUESTION Is N square free?

Answer YES if there exists no prime p such that $p^2 | N$.

Question Square Freeness is very difficult. To date there exists no known algorithm (whether deterministic or probabilistic) which can answer it in time polynomial in the size of the input. Of course, there exists algorithms to answer this

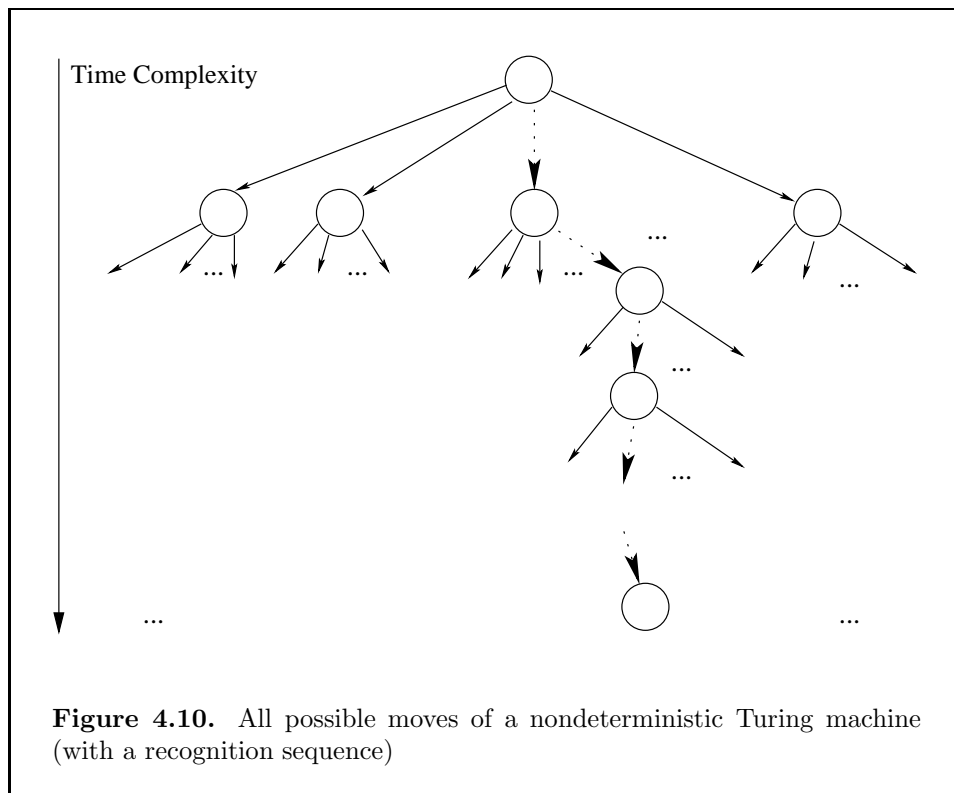
question. For example, the following is one: on input N , perform trial division exhaustively using the square of all odd primes up to $\lfloor \sqrt{N} \rfloor$, and answer YES when all divisions fail. However, for N being a general instance input, this method runs in time $O(\lfloor \sqrt{N} \rfloor) = O(e^{\frac{\log N}{2}})$, i.e., in time exponential in (half) size of N .

Nevertheless, Question **Square Freeness** should not be regarded to be too difficult. If we know some “internal information” of the problem, called a **witness** (or a **certificate** or an **auxiliary input**), then an answer can be *verified* in time polynomial in the size of the input. For example, for input N , the integer $\phi(N)$, which is named **Euler’s phi function** of N and is the number of all positive numbers less than N and co-prime to N (see Definition 5.11), can be used as a witness for an efficient verification algorithm to verify an answer to whether N is square free. The fast verification algorithm involves to compute $\gcd(N, \phi(N))$. To verify the YES answer, one computes $n = \gcd(N, \phi(N))$; if $n = 1$ or $n^2 \nmid N$ then the YES answer is confirmed. The reader may consider how to verify the NO answer. This verification algorithm is due to a basic number theoretic fact which will become apparent to us in Chapter 6 (§6.2). From our study of the time complexity of the great common divisor algorithm (§4.5.3), it is clear that this algorithm runs in time polynomial in the size of N .

Now let us describe a computation device: it models a method to solve the class of problems which share the same property with Question **Square Freeness**. The computation of the device can be described by a tree in Figure 4.10.

The device is called a **nondeterministic Turing machine**. This is a variant Turing machine (review our description of Turing machines in §4.2). At each step the machine will have a finite number of choices as to its next-step move. An input string is said to be recognized if there exists at least one sequence of legal moves which starts from the machine’s initial state when the machine scans the first input symbol, and leads to a state after the machine has completed scanning the input string where a terminating condition is satisfied. We shall name such a sequence of moves a *recognition sequence*.

We can imagine that a nondeterministic Turing machine finds a solution to a recognizable input instance by a series of guesses; a sequence of moves that consist of correct guesses forms a recognition sequence. Thus, all possible moves that the machine can make form a tree (called *computational tree* of a nondeterministic Turing machine, see picture in Figure 4.10). The size (the number of nodes) of the tree is obviously an exponential function in the size of the input. However, since the number of moves in a recognition sequence for a recognizable input instance is the depth d of the tree, we have $d = O(\log(\text{number of nodes in the tree}))$ and therefore the number of moves in a recognition sequence must be bounded by a polynomial in the size of the input instance. Thus, the time complexity for recognizing a recognizable input, via a series of correct guesses, is a polynomial in the size of the input.



Definition 4.8: Class \mathcal{NP} We write \mathcal{NP} to denote the class of languages recognizable by nondeterministic Turing machines in polynomial time.

It is straightforward to see

$$\mathcal{P} \subseteq \mathcal{NP}$$

namely, every language (decisional problem) in \mathcal{P} is trivially recognizable by a nondeterministic Turing machine. It is also trivial to see

$$\mathcal{ZPP}, \mathcal{RP}\text{-(Monte Carlo)}, \mathcal{RP}\text{-(Las Vegas)} \subseteq \mathcal{NP}.$$

In fact, \mathcal{ZPP} , $\mathcal{RP}\text{-(Monte Carlo)}$ and $\mathcal{RP}\text{-(Las Vegas)}$ are all genuine NP problems since they are indeed *nondeterministic polynomial-time* problems^d. The only reason for these sub-classes of NP problems to be efficiently solvable is because these NP problems have abundant witnesses which can be easily found via random guessing. It is only a customary convention that we usually confine \mathcal{NP} to be a

^dRecall the reason given in the beginning of §4.6 for re-naming a sub-class of nondeterministic polynomial-time Turing machines into “probabilistic Turing machines”.

class of nondeterministic polynomial-time (decisional) problems which have scarce witnesses.

With the involvement of random guessing steps, nondeterministic Turing machines do not really offer any useful (i.e., efficient) algorithmic method for NP problems which have scarce witnesses. This is different from the cases of nondeterministic Turing machines being efficient devices for NP problems with abundant witnesses. For NP problems with scarce witnesses, nondeterministic Turing machines merely model a class of decisional problems which share the following property:

An answer to a decisional problem can be verified in polynomial time using a witness.

A witness for an NP problem is modelled by a recognition sequence in the computational tree for a nondeterministic Turing machine (see the dashed branches in the computational tree in Figure 4.10).

Now we ask: without using a witness, what is the exact time complexity for any given problem in \mathcal{NP} ? The answer is *not known*. All known algorithms for solving any problem in \mathcal{NP} without using witness show polynomially-unbounded time complexities. Yet to date no one has been able to prove if this is necessary, i.e., to prove $\mathcal{P} \neq \mathcal{NP}$. Also, no one has so far been able to demonstrate the opposite case, to prove $\mathcal{P} = \mathcal{NP}$. The question

$$\mathcal{P} = \mathcal{NP} ?$$

is a well-known open question in theoretic computer science.

Definition 4.9: Lower- and Upper- Complexity Bounds *A quantity B is said to be the lower- (computational complexity) bound for a problem P if any algorithm A solving P has a computational complexity $C(A) \geq B$.*

A quantity U is said to be an upper- (computational complexity) bound for a problem P if there exists an algorithm A solving P and A has a computational complexity $C(A) \leq U$.

It is usually possible to identify the lower-bound for a problem in \mathcal{P} , namely, to pinpoint exactly the polynomial bound that declares the necessary number of steps needed for solving the problem. Machine DIV3 (Example 4.1) provides such an example: it recognizes an n -bit string in precisely n steps, i.e., using the lowest possible number of steps permitted by the encoding scheme for the input instance.

However, for problems in \mathcal{NP} , it is always difficult to identify the (non-polynomial) lower time complexity bound. All known non-polynomial bounds for NP problems are upper-bounds. For example, as we have “demonstrated” above, $\lfloor \sqrt{N} \rfloor$ is an upper-bound for answering Question Square Freeness with input N (via trial division). An upper-bound essentially says: “only this number of steps are needed for

solving this problem” without adding an important untold part: “but fewer steps may be possible”. In fact, for Question **Square Freeness**, the Number Field Sieve method for factoring N has complexity given by (4.9.1) which has fewer steps than $\lfloor \sqrt{N} \rfloor$.

One should not be confused by “the lower-bound” and “a lower bound”; the latter often appears in the literature (e.g., used by Cook in his famous article [Coo71] that discovered “Satisfiability Problem” being “NP-complete”). The latter only means a newly identified (hence lower) upper-bound, that is, a new complexity result for a problem whose (the) lower-bound has not been identified.

The difficulty for identifying the lower non-polynomial bound for NP problems has a serious consequence in modern cryptography which is based on NP problems. We shall discuss this in §4.11.

4.8.1 Nondeterministic Polynomial-time Complete

Even though we do not know whether or not $\mathcal{P} = \mathcal{NP}$, we do know that certain problems in \mathcal{NP} are as difficult as any in \mathcal{NP} , in the sense that if we had an efficient algorithm to solve one of these problems, then we could find an efficient algorithm to solve any problem in \mathcal{NP} . These problems are called **nondeterministic polynomial-time complete** (**NP-complete** for short).

Definition 4.10: Polynomial Reducible *We say that a language L is polynomially reducible to another language L_0 if there exists a deterministic polynomial-time-bounded Turing machine M which will convert each instance $I \in L$ into an instance $I_0 \in L_0$, such that $I \in L$ if and only if $I_0 \in L_0$.*

Definition 4.11: NP-Complete *A language $L_0 \in \mathcal{NP}$ is nondeterministic polynomial time complete (NP-complete) if any $L \in \mathcal{NP}$ can be polynomially reducible to L_0 .*

A well-known NP-complete problem is so-called **Satisfiability** problem (identified by Cook [Coo71]), which is the first problem found to be NP-complete (page 344 of [Oxf91]). Let $E(x_1, x_2, \dots, x_n)$ denote a Boolean expression constructed from n Boolean variables x_1, x_2, \dots, x_n using Boolean operators, such as \wedge , \vee and \neg .

Problem Satisfiability

INPUT $X = (x_1, \neg x_1, x_2, \neg x_2, \dots, x_n, \neg x_n);$
 $E(x_1, x_2, \dots, x_n);$

OUTPUT YES if $E(x_1, x_2, \dots, x_n)$ is satisfiable.

The meaning for “ $E(x_1, x_2, \dots, x_n)$ is satisfiable” is as follows: there exists a truth assignment, i.e., a sublist X' of X such that for $1 \leq i \leq n$, X' contains either x_i or

$\neg x_i$ but not both, and that $E(X') = \text{True}$.

If a satisfiable truth assignment is given, then obviously the YES answer can be verified in time bounded by a polynomial in n . Therefore by Definition 4.8 we know **Satisfiability** $\in \mathcal{NP}$. Notice that there are 2^n possible truth assignments, and so far we know of no deterministic polynomial-time algorithm to determine whether there exists a satisfiable one.

A proof for **Satisfiability** being NP-complete (due to Cook [Coo71]) can be seen in Chapter 10 of [AHU74] (the proof is constructive, which transforms an arbitrary nondeterministic polynomial-time Turing machine to one that solves **Satisfiability**).

A large list of NP-complete problems has been provided in [GJ79].

For an NP-complete problem, any new result for lowering its upper-bound can be polynomially “reduced” (transformed) to a new result for a whole class of NP problems. Therefore it is desirable, as suggested by [DH76b], to consider to design cryptographic algorithms in NP-complete problems. A successful attack to such a cryptosystem should hopefully lead to solution to the whole class of difficult problems, which should be unlikely. However, such a reasonable desire has so far not led to fruitful results, either in terms of realizing a secure and practical cryptosystem, or in terms of solving the whole class NP problems using an attack to such a cryptosystem. We shall discuss this seemingly strange phenomenon in §4.11.2.

4.9 Non-Polynomial Bounds

There are plenty of functions larger than any polynomial.

Definition 4.12: Non-Polynomially-Bounded Quantity A function $f(n) : \mathbb{N} \mapsto \mathbb{R}$ is said to be unbounded by any polynomial in n if for any polynomial $p(n)$ there exists a natural number n_0 such that for all $n > n_0$, $f(n) > p(n)$.

A function $f(n)$ is said to be **polynomially bounded** if it is not a non-polynomially-bounded quantity.

Example 4.3: Show that for any $a > 1$, $0 < \epsilon < 1$, functions

$$f_1(n) = a^{n^\epsilon (\log n)^{1-\epsilon}}, \quad f_2(n) = n^{(\log \log \log n)^\epsilon}$$

are not bounded by any polynomial in n .

Let $p(n)$ be any polynomial; denoting by d its degree and by c its largest coefficient, then $p(n) \leq cn^d$. First, let $n_0 = \max(c, \lfloor (\frac{d+1}{\log a})^{\frac{2}{\epsilon}} \rfloor)$, then $f_1(n) > p(n)$ for all $n > n_0$; Secondly, let $n_0 = \max(c, \lfloor \exp(\exp(\exp((d+1)^{\frac{1}{\epsilon}}))) \rfloor)$, then $f_2(n) > p(n)$ for all $n > n_0$. \square

In contrast to polynomial-time problems (deterministic or randomized), a problem with time complexity which is non-polynomially bounded is considered to be computationally intractable or infeasible. This is because the resource requirement for solving such a problem grows too fast when the size of the problem instances grows, so fast that it quickly becomes impractically large. For instance, let N be a composite integer of size n (i.e., $n = \log N$); then function $f_1(\log N)$ in Example 4.3 with $a \approx \exp(1.923 + o(1))$ (where $o(1) \approx \frac{1}{\log N}$) and $\epsilon = \frac{1}{3}$ provides the time-complexity expression for factoring N by the Number Field Sieve method (see, e.g., [CDL⁺00]):

$$\exp((1.923 + o(1)) (\log N)^{\frac{1}{3}} (\log \log N)^{\frac{2}{3}}). \quad (4.9.1)$$

For N being a 1024-bit number, this expression provides a quantity larger than 2^{86} . This quantity is currently not manageable even with the use of a vast number of computers running in parallel. This time complexity formula also applies to the best algorithm for solving a “discrete logarithm problem” modulo a prime number N (see Definition 8.2).

We should, however, notice the **asymptotic** fashion in the comparison of functions used in Definition 4.12 ($f(n)$ in Definition 4.12 is also said to be asymptotically larger than any polynomial, or larger than any polynomial in n for sufficiently large n). Even if $f(n)$ is unbounded by any polynomial in n , often it is the case that for a quite large number n_0 , $f(n)$ is less than some polynomial $p(n)$ for $n \leq n_0$. For instance, function $f_2(n)$ in Example 4.3 with $\epsilon = 0.5$ remains to be a smaller quantity than the quadratic function n^2 for all $n \leq 2^{742762245454927736743541}$, even though $f_2(n)$ is asymptotically larger than n^d for any $d \geq 1$. That is why in practice, some algorithms with non-polynomially-bounded time complexities can still be effective for solving problems of small input size; we will see several such algorithms in a few later chapters.

While using the order notation (see Definition 4.2) we deliberately neglect any constant coefficient in complexity expressions, however, we should notice the significance of a constant coefficient which appears in the exponent position of a non-polynomial-bounded quantity (e.g., 1.923 in the expression (4.9.1)). For example, if a new factoring algorithm advances from the current NFS method by reducing the constant 1.923 in the expression in (4.9.1) to 1, then the time complexity for factoring a 1024-bit composite integer using that algorithm will be reduced from about 2^{86} to about 2^{45} . The latter is no longer regarded a too huge quantity for today’s computing technology.

We have defined the notion of non-polynomial bound for large quantities. We can also define a notion for small quantities.

Definition 4.13: Negligible Quantity *A function $\epsilon(n) : \mathbb{N} \mapsto \mathbb{R}$ is said to be a negligible quantity (or $\epsilon(n)$ is negligible) in n if its reciprocal, i.e., $\frac{1}{\epsilon(n)}$, is a non-polynomially-bounded quantity in n .*

If ϵ is a negligible quantity, then $1 - \epsilon$ is said to be an **overwhelming** quantity. If a quantity is not negligible, then we often say it is a non-negligible quantity, or a **significant** quantity.

A negligible function diminishes to 0 faster than the reciprocal of any polynomial does. If we regard a non-polynomially-bounded quantity an unmanageable one (for example, in resource allocation), then it should be harmless for us to neglect any quantity at the level of the reciprocal of a non-polynomially-bounded quantity.

For example,

$$\text{Prob} [\text{“Prime_Gen}(1^k) \text{ is not prime”}]$$

is negligible in k and

$$1 - \text{Prob} [\text{“Prime_Gen}(1^k) \text{ is not prime”}] = \text{Prob} [\text{“Prime_Gen}(1^k) \text{ is prime”}]$$

is overwhelming in k .

Review Example 3.6; for p being a k bit prime number ($q = \frac{p-1}{2}$ being also a prime), we can neglect quantities at the level of $\frac{1}{p-1}$ or smaller and thereby obtain $\text{Prob}[A] \approx \frac{3}{4}$.

4.10 Polynomial-time Indistinguishability

We have just considered that neglecting a negligible quantity is harmless. However, sometimes when we neglect a negligible quantity, we cannot console ourselves by treating it *harmless*. More often, we feel *hopeless* and are forced to abandon an attempt *not* to neglect a small quantity. Let us now describe such a situation through an example.

Consider two experiments over the space of large odd composite integers of a fixed length. Let one of them be called E_{2_Prime} , and the other, E_{3_Prime} . With a significant probability, E_{2_Prime} will yield an integer which has two large distinct prime factors, and otherwise it will yield an integer which has four or more distinct prime factors (that is, E_{2_Prime} will never yield an integer which is either a prime, a prime power, or has three distinct prime factors). For E_{3_Prime} , all integers it yields will have three or more distinct prime factors. Now let someone supply you an integer N by following one of these two experiments. Can you tell with confidence from which of these two experiments N is yielded? (Remember E_{2_Prime} and E_{3_Prime} yield integers of the same length.)

Following Definition 3.5, such an experiment result is a random variable. We know that random variables yielded from E_{2_Prime} and those yielded from E_{3_Prime} have drastically different probability distributions: E_{2_Prime} yields a two-prime product with a significant probability while E_{3_Prime} never does so. However, it is in fact a very hard problem to distinguish random variables from these two experiments.

Let us now define precisely what we mean by **indistinguishable experiments** (also called **indistinguishable ensembles**).

Definition 4.14: Distinguisher for experiments Let $E = \{e_1, e_2, \dots\}$, $E' = \{e'_1, e'_2, \dots\}$ be two sets of experiments (ensembles) in which e_i, e'_j are random variables in a finite sample space \mathbb{S} . Denote $k = \log_2 \#\mathbb{S}$. Let $a = (a_1, a_2, \dots, a_\ell)$ be random variables such that all of them are yielded from either E or E' , where ℓ is bounded by a polynomial in k .

A distinguisher \mathcal{D} for (E, E') is a probabilistic algorithm which halts in time polynomial in k with output in $\{0, 1\}$ and satisfies (i) $\mathcal{D}(a, E) = 1$ iff a is from E ; (ii) $\mathcal{D}(a, E') = 1$ iff a is from E' .

We say that \mathcal{D} distinguishes (E, E') with advantage $\text{Adv} > 0$ if

$$\text{Adv}(\mathcal{D}) = |\text{Prob}[\mathcal{D}(a, E) = 1] - \text{Prob}[\mathcal{D}(a, E') = 1]|.$$

It is important to notice the use of probability distributions in the formulation of an advantage for a distinguisher. A distinguisher is probabilistic polynomial-time (PPT) algorithm.

Example 4.4: Let $E = \{k\text{-bit Primes}\}$ and $E' = \{k\text{-bit Composites}\}$. Define $\mathcal{D}(a, E) = 1$ iff $\text{Prime_Test}(a) \rightarrow \text{YES}$, and $\mathcal{D}(a, E') = 1$ iff $\text{Prime_Test}(a) \rightarrow \text{NO}$ (Prime_Test is specified in Algorithm 4.5). Then \mathcal{D} is a distinguisher for E and E' . When $a \in E$, we have $\text{Prob}[\mathcal{D}(a, E) = 1] = 1$ and $\text{Prob}[\mathcal{D}(a, E') = 1] = 0$; when $a \in E'$, we have $\text{Prob}[\mathcal{D}(a, E) = 1] = 2^{-k}$ and $\text{Prob}[\mathcal{D}(a, E') = 1] = 1 - 2^{-k}$. Hence, $\text{Adv}(\mathcal{D}) \geq 1 - 2^{-(k-1)}$. \square

Definition 4.15: Polynomial-time Indistinguishability Let experiments E, E' and security parameter k be those defined in Definition 4.14. E, E' are said to be polynomially indistinguishable if there exists no distinguisher for (E, E') with advantage $\text{Adv} > 0$ non-negligible in k .

The following assumption is widely accepted to be plausible in computational complexity theory.

Assumption 4.1: (General Indistinguishability Assumption) There exists polynomially indistinguishable experiments.

Experiments $E_{2\text{-Prime}}$ and $E_{3\text{-Prime}}$ are assumed to be polynomially indistinguishable. In other words, if someone supplies us with a set of polynomially many integers which are either all from $E_{2\text{-Prime}}$, or all from $E_{3\text{-Prime}}$, and if we use the best known algorithm as our distinguisher, we will soon feel hopeless and have to abandon our distinguishing attempt.

Notice that since we can factor N and then be able to answer the question correctly, our advantage Adv must be no less than the reciprocal of the function in (4.9.1). However, that value is too small not to be neglected. We say that we are hopeless in distinguishing these two experiments because the best distinguisher we can have will have a negligible advantage in the size of the integer yielded from the experiment. Such an advantage is a slow-growing function of our computational resources. Here “slow-growing” means that even if we add our computational resources in a tremendous manner, the advantage will only grow in a marginal manner so that we will soon become hopeless.

Polynomial indistinguishability is an important security criterion for many cryptographic algorithms and protocols. There are many practical ways to construct polynomially indistinguishable experiments for being useful in modern cryptography. For example, a **pseudo-random number generator** is an important ingredient in cryptography; such a generator generates pseudo-random numbers which have a distribution totally determined (i.e., in a deterministic fashion) by a seed. Yet, a good pseudo-random number generator yields pseudo-random numbers which are polynomially indistinguishable from truly random numbers, that is, the distribution of the random variables output from such a generator is indistinguishable from the uniform distribution of strings which are of the same length as those of the pseudo-random variables. In fact, the following assumption is an instantiation of Assumption 4.1:

Assumption 4.2: (Indistinguishability between Pseudo-randomness and True Randomness) *There exists pseudo-random functions which are polynomially indistinguishable from truly random functions.*

In Chapter 13 we shall see a pseudo-random function (a pseudo-random number generator) which is polynomially indistinguishable from a uniformly random distribution. In that chapter we shall also study a well-known **public-key cryptosystem** named the **Goldwasser-Micali cryptosystem**; that cryptosystem has its security based on polynomially indistinguishable experiments which are related to E_{2_Prime} and E_{3_Prime} (we shall discuss the relationship in §6.4.1). For a further example, a **Diffie-Hellman tuple** (Definition 12.1) of four elements in some **abelian group** and a random quadruple in the same group form indistinguishable experiments which provide security basis for the **ElGamal cryptosystem** and many **zero-knowledge proof protocols**. We will frequently use the notion of polynomial indistinguishability in several later chapters.

4.11 Theory of Computational Complexity and Modern Cryptography

In the end of our short course in computational complexity, we shall provide a discussion on the relationship between the computational complexity and modern

cryptography.

4.11.1 A Necessary Condition

We are able to say that the complexity-theoretic-based modern cryptography uses the conjecture $\mathcal{P} \neq \mathcal{NP}$ as a necessary condition. An encryption algorithm should, on the one hand, provide a user who has in possession of correct encryption/decryption keys with efficient algorithms for encryption and/or decryption, and on the other hand, pose an intractable problem for one (an attacker or a cryptanalyst) who tries to extract plaintext from ciphertext, or to construct a valid ciphertext without using correct keys. Thus, a cryptographic key plays the role of a witness, or an auxiliary input (a more suitable name) to an NP-problem-based cryptosystem.

One might want to argue against our assertion on the necessary condition for modern cryptography. Maybe there exists a cryptosystem which were based on an asymmetric problem in \mathcal{P} : encryption were an $O(n)$ -algorithm and the best cracking algorithm were of order $O(n^{100})$. Indeed, even for the tiny case of $n = 10$, $O(n^{100})$ is a 2^{332} -level quantity which is *way way way beyond* grasp of the world-wide combination of the most advanced computation technologies. Therefore, were such a polynomial-time cryptosystem exists, we should be in a good shape even if it turns out $\mathcal{P} = \mathcal{NP}$. However, the trouble is, while \mathcal{P} does enclose $O(n^k)$ problems for *any* k , it is unlikely that \mathcal{P} contains any problem with such an *asymmetric* behavior. What we do know is that for any given problem in \mathcal{P} , if an **average case** instance is identified to be solvable in $O(n^k)$ time, then the **worst case** instance (the same size) of the same problem can be solved in no more than $O(n^{k+1})$ time. It seems that \mathcal{P} does not contain any asymmetric problem.

The conjecture also forms a necessary condition for the existence of **one-way function**. In the beginning of this book (§1.1.1) we have assumed that a one-way function $f(x)$ should have the following “magic property” (“Magic Property II” in page ??): for all x , it is easy to compute $f(x)$ from x while given any value $f(x)$ it is extremely difficult to find any non-trivial information about x . Now we know that the class \mathcal{NP} provides us with candidates for realizing a one-way function with such a “magic property”. For example, problem Satisfiability defines a one-way function from an n -tuple Boolean space to $\{\text{True}, \text{False}\}$.

In turn, the existence of one-way functions forms a necessary condition for the existence of **digital signatures**. A digital signature should have the following properties: easy to verify and difficult to forge.

Moreover, the notion of polynomial-time indistinguishability which we have studied in §4.10 is also based on the conjecture $\mathcal{P} \neq \mathcal{NP}$. This is the decisional case of hard problems in \mathcal{NP} . In Chapters 13, 14 and 15 we shall see the important role of polynomial-time indistinguishability plays in modern cryptography: the correctness of cryptographic algorithms and protocols.

In particular, we should mention the fundamentally important role that $\mathcal{P} \neq$

\mathcal{NP} conjecture plays in a fascinating subject of public-key cryptography: **zero-knowledge proof protocols** [SGR85] and interactive proof system.

A zero-knowledge protocol is an interactive procedure running between two principals called a **prover** and a **verifier** with the latter having a polynomially-bounded computational power. The protocol allows the former to prove to the latter that the former knows a YES answer to an NP-problem, because the former has in possession of an auxiliary input, without letting the latter learn how to conduct such a proof (i.e., without disclosing the auxiliary input to the latter). Hence the verifier gets “zero-knowledge” about the prover’s auxiliary input. Such a proof can be modelled by a nondeterministic Turing machine with an added random-tape. The prover can make use of auxiliary input and so the machine can always be instructed (by the prover) to move along a recognition sequence (i.e., to demonstrate the YES answer) regarding the input problem. Consequently, the time complexity for a proof is a polynomial in the size of the input instance. The verifier should challenge the prover to instruct the machine to move either along a recognition sequence, or along a different sequence, and the challenge should be uniformly random. Thus, from the verifier’s observation, the proof system behaves precisely in the fashion of a randomized Turing machine (review §4.6). As a matter of fact, it is the property that the error probability of such a randomized Turing machine can be reduced to a negligible quantity by repeated execution (as shown in Theorem 4.3 and Theorem 4.4) that forms the basis for convincing the verifier that the prover does know the YES answer to the input problem. In Chapters ?? and ?? we shall study zero-knowledge proof protocols.

4.11.2 Not a Sufficient Condition

The conjecture $\mathcal{P} \neq \mathcal{NP}$ does not provide a sufficient condition for a secure cryptosystem even if such a cryptosystem is based on an NP-complete problem. The well-known broken NP-complete **knapsack problem** provides a counterexample [MH78].

After our course in computational complexity, we are now able to provide two brief but clear explanations on why cryptosystems based on NP (or even NP-complete) problems are often broken.

First, as we have particularly pointed out in an early stage of our course (e.g., review Definition 4.1), the complexity-theoretic approach to computational complexity restricts a language L (a problem) in a complexity class with a universal-style quantifier: “any instance $I \in L$ ”. This restriction results in the worst-case complexity analysis: a problem is regarded to be difficult even if there only exists *one* difficult instance. In contrast, a cryptanalysis can be considered successful if it can break a non-trivial fraction of the instances. That is why breaking of an NP-complete-based cryptosystem does not lead to a solution to the underlying NP-complete problem. It is clear that the worst-case complexity criterion is hopeless

and useless for measuring the security for the practical cryptosystems.

The second explanation lies in the *inherent* difficulty to identify the lower-bounds for NP problems (review our discussion on lower- and upper-bounds in §4.8). The cryptanalysis task for an NP-problem-based cryptosystem, even if the task has been proven to be the underlying NP-problem, is at best an open problem since we only know an upper-bound complexity for the problem. More often, the underlying intractability for such an NP-based cryptosystem is not even clearly identified.

We should further remind the reader one of the themes of this book: non-textbook aspects of security for applied cryptography (review §1.1.3). Cryptography systems for real world applications can be erred in many practical ways which may have little to do with mathematical properties underlying the security of an algorithm.

A positive attitude toward the design and analysis of secure public-key cryptosystems, which gets wide acceptance recently, is to formally prove that a cryptosystem is secure (**provable security**) using polynomial reduction techniques (see Definition 4.10): to “reduce” via an efficient transformation *any* efficient attack on the cryptosystem to a solution to an instance of a known NP problem. Usually the NP problem is in a small set of widely accepted “pedigree class”. Such a reduction is usually called a **reduction to contradiction** because it is widely believed that the widely accepted “pedigree problem” does not have an efficient solution. Such a proof provides a high confidence of the security of the cryptosystem in question. We shall study this methodology in Chapters 13 and 14.

4.12 Chapter Summary

Computational complexity is one of the foundations (indeed, the most important foundation) for modern cryptography. Due to this importance, this chapter provides a self-contained and systematic introduction to this foundation.

We start with the notion of Turing computability as a class of polynomial-time (efficiently solvable) problems which include the class of deterministic ones (\mathcal{P}) and several classes of nondeterministic (probabilistic) ones (several sub-classes in \mathcal{PP}). We exemplify important algorithms (some of them are very useful) for each (sub-)class.

We then introduce the class \mathcal{NP} , problems which do not appear to be solvable by efficient algorithms, deterministic or otherwise, while with their membership in the class being efficiently verifiable given a witness. We identify that an \mathcal{NP} problem either has scarce witnesses or is an efficiently solvable one in a sub-class of \mathcal{PP} .

In our course, we also introduce various important notions in computational complexity. These include efficient algorithms (several important algorithms are constructed with precise time complexity analysis), order notation, polynomial reducibility, negligible quantity, lower, upper and non-polynomial bounds, and in-

distinguishability. These notions will be frequently used in the rest part of the book.

Finally, we conduct a discussion on the fundamental roles of \mathcal{NP} problems and the complexity-theoretic basis playing in modern cryptography.

ALGEBRAIC FOUNDATIONS

In this chapter we introduce three algebraic structures which not only are central concepts of abstract algebra, but also provide the basic elements and operations for modern cryptography and cryptographic protocols. These three structures are: **group**, **ring** and **field**. A more comprehensive study of these topics can be found in [LN97].

5.1 Groups

Roughly speaking, a group is a set of objects with an operation defined between any two objects in the set. The need for an operation over a set of objects is so natural. For example, upon every sunset, an ancient shepherd should have counted his herd of sheep by performing an “add 1” operation. Maybe the shepherd did not even know numbers, let alone the notion of addition; but these should not prevent him from performing his operation properly. He could keep with him a sack of stones and match each sheep against each stone. Then, as long as he always ended up his matching operation when no more stone left to match, then he knew that his herd of sheep were fine. Sheep or stones or some other objects, with an operation defined over the objects, and we have a group.

Definition 5.1: Group *A group G is a set together with an operation \circ satisfying the following conditions:*

$$1. \forall a, b \in G : a \circ b \in G \quad (\text{Closure Axiom})$$

$$2. \forall a, b, c \in G : a \circ (b \circ c) = (a \circ b) \circ c \quad (\text{Associativity Axiom})$$

$$3. \exists \text{ unique element } e \in G : \forall a \in G : a \circ e = e \circ a = a \quad (\text{Identity Axiom})$$

*The element e is called the **identity** element.*

$$4. \forall a \in G : \exists a^{-1} \in G : a \circ a^{-1} = a^{-1} \circ a = e \quad (\text{Inverse Axiom})$$

Definition 5.2: Infinite and Finite Groups A group G is said to be infinite if the number of elements in the set G is infinite, otherwise, the group is finite.

Definition 5.3: Abelian Group A group G is abelian if for all $a, b \in G$, $a \circ b = b \circ a$.

In other words, an abelian group is a **commutative group**. In this book we shall have no occasion to deal with non-abelian group. So all groups to appear in the rest of this book are abelian, and we shall often omit the prefix “abelian”.

Example 5.1: (Groups)

1. The set of integers \mathbb{Z} is a group under addition $+$ (and so is an **additive** group) with $e = 0$ and $a^{-1} = -a$. This is an infinite group (and is abelian). Same, the set rational numbers \mathbb{Q} , the set of real numbers \mathbb{R} , and the set of complex numbers \mathbb{C} are groups are additive, infinite groups with the same definitions for identity and inverse.
2. Non-zero elements of \mathbb{Q} , \mathbb{R} and \mathbb{C} under multiplication \cdot are groups with $e = 1$ and a^{-1} being the multiplicative inverse (defined in the usual way). We denote by \mathbb{Q}^* , \mathbb{R}^* , \mathbb{C}^* these groups, respectively. They are called **multiplicative** groups. They are infinite.
3. For any $n \geq 1$, the set of integers modulo n forms a finite additive group of n elements; here addition is in terms of modulo n , the identity element is 0, and for all element a in the group, $a^{-1} = n - a$ (property 2 of Theorem 4.2). We denote by \mathbb{Z}_n this group. (In §6.1, we will provide an algebraic definition and the standard notation for \mathbb{Z}_n .)
4. The numbers for hours over a clock form \mathbb{Z}_{12} under addition modulo 12. Let us name \mathbb{Z}_{12} “clock group”.
5. The subset of \mathbb{Z}_n containing elements relatively prime to n (i.e., $\gcd(a, n) = 1$) forms a finite multiplicative group; here multiplication is in terms of modulo n , $e = 1$, and for any element a in the group, a^{-1} can be computed using extended Euclid algorithm (Algorithm 4.2). We denote by \mathbb{Z}_n^* this group. For example, $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.
6. For set $B = \{F, T\}$, let $\circ = \oplus$ be (logical XOR): $F \oplus F = F$, $F \oplus T = T \oplus F = T$, $T \oplus T = F$. Then B under \oplus is a finite group with $e = F$ and $T^{-1} = T$.
7. The roots of $x^3 - 1 = 0$ is a finite group under multiplication with $e = 1$ (obviously 1 is a root). Denote by $\text{Roots}(x^3 - 1)$ this group. Let us find the other group elements in $\text{Roots}(x^3 - 1)$ and their inverses. As a degree-3 polynomial, $x^3 - 1$ has three roots only. Let α , β be the other two

roots. From $x^3 - 1 = (x - 1)(x^2 + x + 1)$, α and β must be the two roots of $x^2 + x + 1 = 0$. By the relation between the roots and the coefficient of a quadratic equation, we have $\alpha\beta = 1$. Thus, $\alpha^{-1} = \beta$ and $\beta^{-1} = \alpha$. The reader may check that Closure Axiom is satisfied (i.e., α^2 and β^2 are roots of $x^3 - 1 = 0$). \square

Definition 5.4: Shorthand Representation of Repeated Group Operations

Let G be a group with operation \circ . For any element $a \in G$, and for any non-negative integer $i \in \mathbb{N}$, we denote by $a^i \in G$ the following element

$$\underbrace{a \circ a \circ \cdots \circ a}_i.$$

We should pay attention to two points in the following remark.

Remark 5.1:

- i) We write $a^i \in G$ only for a shorthand presentation of $\underbrace{a \circ a \circ \cdots \circ a}_i$. Notice that the operations between the integer i and the group element a is not a group operation.
- ii) Some groups are conventionally written additively, e.g., \mathbb{Z}_n with the group operation being the conventional addition modulo n . For these groups, the reader may view a^i as $i \cdot a$. However, in this shorthand view, one must notice that “ \cdot ” here is not a group operation and the integer i is usually not a group element. \square

Definition 5.5: Subgroup A subgroup of a group G is a non-empty subset H of G which is itself a group under the same operation as that of G . We write $H \subseteq G$ to denote that H is a subgroup of G , and $H \subset G$ to denote that H is a proper subgroup of G (i.e., $H \neq G$).

Example 5.2:

1. Under addition, $\mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$;
2. Under addition, the set of even integers plus 0 is a subgroup of the groups in (1); so is the set of odd numbers plus 0.
3. Under addition modulo 12, set $\{0, 3, 6, 9\}$ is a subgroup of the “clock group” \mathbb{Z}_{12} , while $\{0, 2, 6, 9\}$ is not.
4. Under multiplication, $\mathbb{Q}^* \subseteq \mathbb{R}^* \subseteq \mathbb{C}^*$.

5. Let n be an odd positive integer and let $\text{Fermat}(n)$ denote the subset of \mathbb{Z}_n^* such that any $a \in \text{Fermat}(n)$ satisfies $a^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$. Then

$$\text{Fermat}(n) \subseteq \mathbb{Z}_n^*.$$

Moreover, if n is a prime number, then by Fermat's Little Theorem (Theorem 6.10), $\text{Fermat}(n) = \mathbb{Z}_n^*$; otherwise, $\text{Fermat}(n)$ is a proper subgroup of \mathbb{Z}_n^* .

6. $\{F\}$ is a proper subgroup of the group B in Example 5.1(6). However, $\{T\}$ is not a subgroup of B since it does not contain an identity (i.e., breach of Identity Axiom).
7. (Review Example 4.1) Polynomial-time language Div3 is a subgroup of \mathbb{Z} ;
8. Set $\{e\}$ is a subgroup of any group. □

Definition 5.6: Order of a Group *The number of elements in a finite group G is called the order of G and is denoted by $\#G$.*

Example 5.3:

1. $\#Z_n = n$;
2. In Example 5.1(6), $\#B = 2$;
3. In Example 5.1(7), $\#\text{Roots}(x^3 - 1) = 3$. □

5.1.1 Lagrange's Theorem

Let us now introduce a beautiful and important theorem in group theory.

Definition 5.7: Coset *Let G be a (abelian) group and $H \subseteq G$. For $a \in G$, set $a \circ H \stackrel{\text{def}}{=} \{a \circ h \mid h \in H\}$ is called a coset of H .*

Theorem 5.1: (Lagrange's Theorem) *If H is a subgroup of G then $\#H \mid \#G$, that is, $\#H$ divides $\#G$.*

Proof

For $H = G$, $\#H \mid \#G$ holds trivially. Let us consider $H \neq G$.

For any $a \in G \setminus H$, by Closure Axiom, coset $a \circ H$ is a subset of G . We can show the following two facts:

- i) For any $a \neq a'$, if $a \notin a' \circ H$ then $(a \circ H) \cap (a' \circ H) = \emptyset$.

ii) $\#(a \circ H) = \#H$.

For (i), suppose $\exists b \in (a \circ H) \cap (a' \circ H)$. So $\exists c, c' \in H$: $a \circ c = b = a' \circ c'$. Applying Inverse Axiom, Identity Axiom, Closure Axiom and Associative Axiom on elements in H , we have

$$a = a \circ e = a \circ (c \circ c^{-1}) = b \circ c^{-1} = (a' \circ c') \circ c^{-1} = a' \circ (c' \circ c^{-1}) \in a' \circ H.$$

This contradicts our assumption: $a \notin a' \circ H$. As a special case, for $a \notin H = e \circ H$, we have $H \cap (a \circ H) = \emptyset$.

For (ii), $\#(a \circ H) \leq \#H$ holds trivially by coset's definition. Suppose that the inequality is rigorous. This is only possible because for some $b \neq c$, $b, c \in H$, $a \circ b = a \circ c$. Applying Inverse Axiom in G , we reach $b = c$, contradicting to $b \neq c$.

Thus, G is partitioned by H and the family of its mutually disjoint cosets, each has the size $\#H$. Hence $\#H \mid \#G$. (In general, partitioning a set means splitting it into disjoint subsets.) \square

Lagrange's Theorem is not only very beautiful in group theory, it is also very important in applications. Review our probabilistic primality test algorithm `Prime_Test` in §???. That algorithm tests whether an odd number p is prime by testing congruence

$$x^{(p-1)/2} \equiv \pm 1 \pmod{p}$$

using random $x \in_U \mathbb{Z}_p^*$. In Example 5.2(5) we have shown that elements in \mathbb{Z}_p^* which holds this congruence form the set $\text{Fermat}(p)$ which is a proper subgroup of \mathbb{Z}_p^* if p is not prime. Thus, by Lagrange's Theorem, $\#\text{Fermat}(p) \mid \#\mathbb{Z}_p^*$. Hence, if p is not prime, $\#\text{Fermat}(p)$ can be at most as half the quantity $\#\mathbb{Z}_p^*$. This provides us with the error probability bound $\frac{1}{2}$ for each step of test, i.e., the working principle of `Prime_Test`.

We will discuss another important application of Lagrange's Theorem in public-key cryptography in §5.1.2.

Definition 5.8: Quotient Group Let G be a (abelian) group and $H \subseteq G$. The quotient group of G modulo H , denoted by G/H , is the set of all cosets $a \circ H$ with a ranging over G , with the group operation \star defined by $(a \circ H) \star (b \circ H) = (a \circ b) \circ H$, and with the identity element being $e \circ H$.

Corollary 5.1: Let G be a finite (abelian) group and $H \subseteq G$. Then

$$\#(G/H) = \frac{\#G}{\#H}.$$

\square

5.1.2 Order of Group Element

If we say that in a group, the identity element is special in a unique way, then other elements also have some special properties. One of such properties can be thought of as the “distance” from the identity element.

Definition 5.9: Order of Group Element *Let G be a group and $a \in G$. The order of the element a is the least positive integer $i \in \mathbb{N}$ satisfying $a^i = e$, and is denoted by $\text{ord}(a)$. If such an integer i does not exist, then a is called an element of infinite order.*

We should remind the reader the shorthand meaning of a^i where i is an integer and a is a group element. The shorthand meaning of the notation has been defined in Definition 5.4 and further explained in Remark 5.1.

Example 5.4:

1. In the “clock group” \mathbb{Z}_{12} , $\text{ord}(1) = 12$, since 12 is the least positive number satisfying $12 \cdot 1 \equiv 0 \pmod{12}$; the reader may verify the following: $\text{ord}(2) = 6$, $\text{ord}(3) = 4$, $\text{ord}(4) = 3$, $\text{ord}(5) = 12$. Try to find the orders for the rest of the elements.
2. In B in Example 5.1(6), $\text{ord}(F) = 1$ and $\text{ord}(T) = 2$.
3. In $\text{Roots}(x^3 - 1)$ in Example 5.1(7), $\text{ord}(\alpha) = \text{ord}(\beta) = 3$, and $\text{ord}(1) = 1$.
4. In \mathbb{Z} , $\text{ord}(1) = \infty$. □

Corollary 5.2: (Lagrange) *Let G be a finite group and $a \in G$ be any element. Then $\text{ord}(a) \mid \#G$.*

Proof For any $a \in G$, if $a = e$ then $\text{ord}(a) = 1$ and so $\text{ord}(a) \mid \#G$ is a trivial case. Let $a \neq e$. Since G is finite, we have $1 < \text{ord}(a) < \infty$. Elements

$$a, a^2, \dots, a^{\text{ord}(a)} = e \tag{5.1.1}$$

are necessarily distinct. Suppose they were not, then $a^r = a^s$ for some non-negative integers r and s satisfying $1 \leq r < s \leq \text{ord}(a)$. Applying “Inverse Axiom” of $(a^r)^{-1}$ to both sides, we will have, $a^{s-r} = e$ where $0 < s - r < \text{ord}(a)$. This contradicts the definition of $\text{ord}(a)$ being the least positive integer satisfying $a^{\text{ord}(a)} = e$.

It is easy to check that the $\text{ord}(a)$ elements in (5.1.1) form a subgroup of G . By Lagrange’s Theorem, $\text{ord}(a) \mid \#G$. □

Corollary 5.2, which we have shown to be a direct application of Lagrange Theorem’s, provides a relationship between the order of a group and the orders of

elements in the group. This relationship has an important application in public-key cryptography: the famous cryptosystems of Rivest, Shamir and Adleman (RSA) [RSA78] work in a group of a secret order which is known exclusively to the key owner. A ciphertext can be considered as a random element in the group. With the knowledge of the group order the key owner can use the relationship between the order of the element and the order of the group to transform the ciphertext back to plaintext (i.e., to decrypt). We will study the RSA cryptosystems in §??.

5.1.3 Cyclic Groups

Example 5.1(4) indicates that we can conveniently view \mathbb{Z}_n as n points dividing a circle. This circle is (or these n points are) formed by n repeated operations a^1, a^2, \dots, a^n for some element $a \in \mathbb{Z}_n$. This is a *cyclic view* of \mathbb{Z}_n . For addition modulo n as the group operation, $a = 1$ provides a cyclic view of \mathbb{Z}_n . The reader may check that for the case of $n = 12$ as in Example 5.1(4), 5, 7, 11 are the other three elements which can also provide cyclic views for \mathbb{Z}_{12} .

Informally speaking, if a group has a cyclic view, then we say that the group is a **cyclic group**. Cyclic groups are groups with nice properties. They have wide applications in cryptography.

Definition 5.10: Cyclic Group, Group Generator *A group G is said to be cyclic if there exists an element $a \in G$ such that for any $b \in G$, there exists an integer $i \geq 0$ such that $b = a^i$. Element a is called a generator of G . G is also called the group generated by a .*

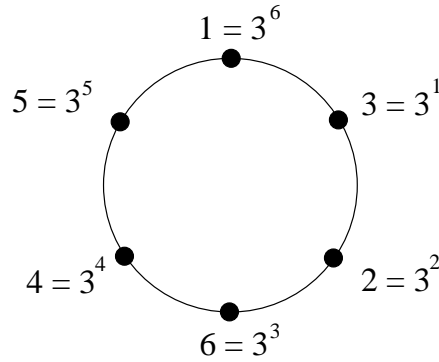
When a group is generated by a , we can write $G = \langle a \rangle$.

A generator of a cyclic group is also called a **primitive root** of the group's identity element. The meaning of this name will become clear in §5.3.3 (Theorem 5.11).

Example 5.5:

1. For $n \geq 1$, the additive \mathbb{Z}_n is cyclic because, obviously, 1 is a generator.
2. B in Example 5.1(6) is cyclic and is generated by T .
3. $\text{Roots}(x^3 - 1)$ in Example 5.1(7) is cyclic and is generated by α or β .
4. Let p be a prime number. Then the multiplicative group \mathbb{Z}_p^* is cyclic. This is because \mathbb{Z}_p^* contains element of order $p - 1 = \#\mathbb{Z}_p^*$ and hence such an element generates the whole group. In Algorithm 4.6 we have seen informally an evidence for \mathbb{Z}_p^* containing a generator. We will provide a formal proof of \mathbb{Z}_p^* being cyclic in Theorem 5.12.

5. In group \mathbb{Z}_7^* , the element 3 is a generator; this element provides a cyclic view for \mathbb{Z}_7^* as follows (remember the group operation being multiplication modulo 7):



□

Definition 5.11: Euler's Function For $n \in \mathbb{N}$ with $n \geq 1$, Euler's function $\phi(n)$ is the number of integers k with $0 \leq k < n$ and $\gcd(k, n) = 1$.

A number of useful results can be derived for cyclic groups.

Theorem 5.2:

1. Every subgroup of a cyclic group is cyclic.
2. For every positive divisor d of $\# \langle a \rangle$, $\langle a \rangle$ contains precisely one subgroup of order d .
3. If $\# \langle a \rangle = m$, then $\# \langle a^k \rangle = \text{ord}(a^k) = m / \gcd(k, m)$.
4. For every positive divisor d of $\# \langle a \rangle$, $\langle a \rangle$ contains $\phi(d)$ elements of order d .
5. Let $\# \langle a \rangle = m$. Then $\langle a \rangle$ contains $\phi(m)$ generators. They are elements a^r such that $\gcd(r, m) = 1$.

Proof

1. Let $H \subseteq \langle a \rangle$. If $H = \langle e \rangle$ or $H = \langle a \rangle$ then H is obviously cyclic. So we only consider other cases of H . Let $d > 1$ be the least integer such that $a^d \in H$, and let $a^s \in H$ for some $s > d$. Dividing s by d : $s = dq + r$ for some $0 \leq r < d$. Since $a^{dq} \in H$ we have $a^r = a^{s-dq} \in H$. The minimality of d and $H \neq \langle a \rangle$ imply $r = 0$. So s is a multiple of d . So H only contains the powers of a^d , and hence is cyclic.

2. Let $d > 1$ and $d|m = \# \langle a \rangle$. Then $\langle a^{\frac{m}{d}} \rangle$ is an order- d subgroup of $\langle a \rangle$ since d is the least integer satisfying $(a^{\frac{m}{d}})^d = e$. Let us assume that there exists another order- d subgroup of $\langle a \rangle$ which is different from $\langle a^{\frac{m}{d}} \rangle$. By 1, such a subgroup must be cyclic and hence be $\langle a^k \rangle$ for some $k > 1$. From $a^{kd} = e$ with minimality of m we have $m|kd$, or equivalently, $\frac{m}{d}|k$. So $a^k \in \langle a^{\frac{m}{d}} \rangle$, i.e., $\langle a^k \rangle \subseteq \langle a^{\frac{m}{d}} \rangle$. The same order of these two groups means $\langle a^k \rangle = \langle a^{\frac{m}{d}} \rangle$. This contradicts our assumption $\langle a^k \rangle \neq \langle a^{\frac{m}{d}} \rangle$.
3. Let $d = \gcd(k, m)$. Then by 2 there exists a unique order- d subgroup of $\langle a \rangle$. Let this subgroup be $\langle a^\ell \rangle$ for some least $\ell > 1$, i.e., ℓ is the least integer satisfying $a^{d\ell} = e$. By the minimality of m , we have $m|d\ell$, or equivalently, $\frac{m}{d}|\ell$. The least case for ℓ is when $d = \gcd(\ell, m)$, i.e., $\ell = k$.
4. Let $d|m = \# \langle a \rangle$ and let a^k be any element in $\langle a \rangle$ for $0 \leq k < m$. By 3, element a^k is of order $\frac{m}{d}$ if and only if $\frac{m}{d} = \gcd(k, m)$. Write $k = c\frac{m}{d}$ with $0 \leq c < d$. Then $\gcd(k, m) = \frac{m}{d}$ is equivalent to $\gcd(c, d) = 1$. By Definition 5.11, there are $\phi(d)$ such c .
5. For $m = \# \langle a \rangle$, by 4, $\langle a \rangle$ contains $\phi(m)$ elements of order m , and they are of order m and hence are generators of $\langle a \rangle$. Further by 3, these generators are a^r with $\gcd(r, m) = 1$. \square

Corollary 5.3: *A prime-order group is cyclic, and any non-identity element in the group is a generator.*

Proof Let G be a group of prime order p . Let $a \in G$ be any non-identity element. From Corollary 5.2, $\text{ord}(a)|\#G = p$. Since $a \neq e$, $\text{ord}(a) \neq 1$. Then it has to be $\text{ord}(a) = p$. Therefore $\langle a \rangle = G$, i.e., a is a generator of G . \square

Example 5.6: Consider the “clock group” \mathbb{Z}_{12} :

- for $1|12$, it contains an order-1 subgroup $\{0\}$; because $\phi(1) = 1$, 0 is the only element of order 1;
- for $2|12$, it contains an order-2 subgroup $\{0, 6\}$; because $\phi(2) = 1$, 6 is the only element of order 2;
- for divisor 3 of 12, it contains an order-3 subgroup $\{0, 4, 8\}$; 4 and 8 are the $2 = \phi(3)$ elements of order 3;
- for divisor 4 of 12, it contains an order-4 subgroup $\{0, 3, 6, 9\}$; 3 and 9 are the $2 = \phi(4)$ elements of order 4;
- for divisor 6 of 12, it contains an order-6 subgroup $\{0, 2, 4, 6, 8, 10\}$; 2 and 10 are the $2 = \phi(6)$ elements of order 6;

- for divisor 12 of 12, it contains an order-12 subgroup \mathbb{Z}_{12} ; because $\phi(12) = 4$, 1, 5, 7 and 11 are the 4 elements of order 12.

The reader may analyse the multiplicative \mathbb{Z}_7^* analogously. \square

5.1.4 The Multiplicative Group \mathbb{Z}_N^*

Let $N = PQ$ for P and Q being distinct odd prime numbers. The multiplicative group \mathbb{Z}_N^* is very important in modern cryptography. Let us now have a look at its structure. We stipulate that all N in this subsection is such a composite.

Since elements in \mathbb{Z}_N^* are positive integers less than N and co-prime to N . By Definition 5.11, this group contains $\phi(N) = (P-1)(Q-1)$ elements (see Lemma 6.1 to confirm $\phi(N) = (P-1)(Q-1)$).

Theorem 5.3: Any element in \mathbb{Z}_N^* has an order dividing $\text{lcm}(P-1, Q-1)$.

Proof Let $a \in \mathbb{Z}_N^*$. By Fermat's Little Theorem (Theorem 6.10) we know

$$a^{(P-1)} \equiv 1 \pmod{P}.$$

Denoting $\lambda = \text{lcm}(P-1, Q-1)$, trivially we have

$$a^\lambda \equiv 1 \pmod{P}.$$

Symmetrically we can also derive

$$a^\lambda \equiv 1 \pmod{Q}.$$

These two congruences say that $a^\lambda - 1$ is a multiple of P and is also a multiple of Q . Since P and Q are prime numbers, $a^\lambda - 1$ must be a multiple of $N = PQ$. This means

$$a^\lambda \equiv 1 \pmod{N}.$$

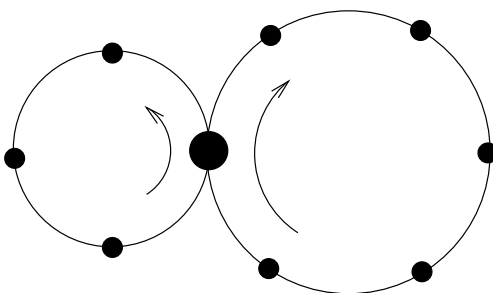
Therefore, λ is a multiple of the order of a modulo N . \square

Notice that both $P-1$ and $Q-1$ are even, therefore $\text{lcm}(P-1, Q-1) < (P-1)(Q-1) = \phi(N)$. Theorem 5.3 says that there is no element in \mathbb{Z}_N^* is of order $\phi(N)$. That is, \mathbb{Z}_N^* contains no generator. So by Definition 5.10, \mathbb{Z}_N^* is non-cyclic.

Corollary 5.4: The multiplicative group \mathbb{Z}_N^* is non-cyclic. \square

Example 5.7: For $N = 5 \times 7 = 35$, let $a \in \mathbb{Z}_{35}^*$ be such an element: $a \pmod{5} \in \mathbb{Z}_5^*$ has the maximum order 4 and provides a cyclic view for the cyclic group \mathbb{Z}_5^* (the left circle below, of period 4); $a \pmod{7} \in \mathbb{Z}_7^*$ has the maximum order 6 and provides a cyclic view for the cyclic group \mathbb{Z}_7^* (the right circle below, of period 6). Then

the order of $a \in \mathbb{Z}_{35}^*$ can be viewed as the period decided by two engaged toothed wheels. One has four teeth and the other has six teeth. We initially chalk-mark a large dot at the engaged point of the two wheels. Now let the engaged gear revolve, and the large chalk mark becomes two separate marks on the two wheels. These two separate marks will meet again after the mark on the four-toothed wheel has travelled 3 revolutions, and that on the six-toothed wheel, 2 revolutions. Therefore, the order (period) of $a \in \mathbb{Z}_{35}^*$ is exactly the distance between the separation and the reunion of the large chalk mark, and is $3 \times 4 = 2 \times 6 = 12 = \text{lcm}((5-1), (7-1))$.



□

The reader may use the same method to find the order of an element b modulo 35: $b \pmod{5} \in \mathbb{Z}_5^*$ has the maximum order 4, but $b \pmod{7} \in \mathbb{Z}_7^*$ has a non-maximum order 3.

Let $\text{ord}_X(a)$ denote the order of an element modulo a positive number X . In general, any element $a \in \mathbb{Z}_N^*$ has the order $\text{ord}_N(a)$ defined by $\text{ord}_P(a)$ and $\text{ord}_Q(a)$ in the following relation:

$$\text{ord}_N(a) = \text{lcm}(\text{ord}_P(a), \text{ord}_Q(a)). \quad (5.1.2)$$

Since \mathbb{Z}_P^* and \mathbb{Z}_Q^* are both cyclic, they have elements of maximum orders $P-1$ and $Q-1$, respectively. Consequently, \mathbb{Z}_N^* contains elements of the maximum order $\text{lcm}(P-1, Q-1)$. However, it is possible for some $a \in \mathbb{Z}_N^*$ to satisfy $\text{ord}_N(a) = \text{lcm}(P-1, Q-1)$ but $\text{ord}_P(a) < P-1$ or $\text{ord}_Q(a) < Q-1$ (see the problem we left for the reader to solve following Example 5.7).

In the next chapter we will study how to construct elements of certain properties, e.g., (5.1.2) in \mathbb{Z}_N^* using elements (of certain properties) in \mathbb{Z}_P^* and \mathbb{Z}_Q^* .

5.2 Rings and Fields

One day our ancient shepherd settled down and became a farmer. He needed to figure out with his neighbors the areas of their lands. The shepherds-turned-farmers began to realize that it was no longer possible for them to use one basic operation for everything: they needed not only sum, but also product. The need for *two* operations over a set of objects started then.

Definition 5.12: Ring *A ring R is a set together with two operations: (addition) $+$ and (multiplication) \cdot , and has the following properties:*

1. *Under addition $+$, R is an abelian group; denote by $\mathbf{0}$ the additive identity (called the **zero-element**);*
2. *Under multiplication \cdot , R satisfies Closure Axiom, Associativity Axiom and Identity Axiom; denote by $\mathbf{1}$ the multiplicative identity (called the **unity-element**); $\mathbf{1} \neq \mathbf{0}$;*
3. $\forall a, b \in R : a \cdot b = b \cdot a$ (Commutative Axiom)
4. $\forall a, b, c \in R : a \cdot (b + c) = a \cdot b + a \cdot c$ (Distribution Axiom)

In this definition, the bold form $\mathbf{0}$ and $\mathbf{1}$ are used to highlight that these two elements are abstract elements and are not necessarily their integer counterparts (see, e.g., Example 5.8(3) in a moment).

Similar to our confinement of ourselves to the commutative groups, in Definition 5.12 we have stipulated multiplication to satisfy Commutative Axiom. So Definition 5.12 defines a **commutative ring** and that is the ring to be considered in this book. We should also stress that $+$ and \cdot are abstract operations: that is, they are not necessarily the ordinary addition and multiplication between integers. Whenever possible, we shall shorten $a \cdot b$ into ab ; explicit presentation of the operation “ \cdot ” will only be needed where the operation is written without operands.

Example 5.8: (Rings)

1. \mathbb{Z} , \mathbb{Q} , \mathbb{R} and \mathbb{C} are all rings under usual addition and multiplication with $\mathbf{0} = 0$ and $\mathbf{1} = 1$.
2. For any $n > 0$, Z_n is a ring under addition and multiplication modulo n with $\mathbf{0} = 0$ and $\mathbf{1} = 1$.
3. Let B be the additive group defined in Example 5.1(6) with the zero-element F . Let the multiplication operation be \wedge (logical And): $F \wedge F = F$, $F \wedge T = T \wedge F = F$, $T \wedge T = T$. Then B is a ring with the unity-element T . \square

At a first glance, Definition 5.12 has only defined multiplication for non-zero elements. In fact, multiplication between the zero-element and other elements has been defined by Distribution Axiom. For example, $\mathbf{0}a = (b + (-b))a = ba + (-b)a = ba - ba = \mathbf{0}$. Moreover, a ring can have **zero-divisors**, that is, elements a, b satisfying $ab = \mathbf{0}$ with $a \neq \mathbf{0}$ and $b \neq \mathbf{0}$. For example, for $n = k\ell$ being a nontrivial factorization of n , both k and ℓ are non-zero elements in the ring Z_n , and the product $k\ell = n = 0 \pmod{n}$ is the zero-element.

Definition 5.13: Field *If the non-zero elements of a ring forms a group under multiplication, then the ring is called a field.*

The Closure Axiom for the multiplicative group (i.e., the non-zero elements) of a field implies that a field F cannot contain a zero-divisor, that is, for any $a, b \in F$, $ab = \mathbf{0}$ implies either $a = \mathbf{0}$ or $b = \mathbf{0}$.

Example 5.9: (Fields)

1. \mathbb{Q} , \mathbb{R} and \mathbb{C} are all fields under usual addition and multiplication with $\mathbf{0} = 0$ and $\mathbf{1} = 1$.
2. The two-element ring B in Example 5.8(3) is a field. □

We shall meet some other fields in a moment.

Note that \mathbb{Z} under integer addition and multiplication is not a field because any non-zero element does not have a multiplicative inverse in \mathbb{Z} (violation of the Inverse Axiom). Also, for n being a composite, \mathbb{Z}_n is not a field too since we have seen that \mathbb{Z}_n contains zero-divisors (violation of the Closure Axiom).

Sometimes there will be no need for us to care the difference among a group, a ring or a field. In such situation we shall use an **algebraic structure** to refer to either of these structures.

The notions of finite group, subgroup, quotient group and the order of group can be extended straightforwardly to rings and fields.

Definition 5.14: *An algebraic structure is said to be finite if it contains a finite number of elements. The number of elements is called the order of the structure.*

A substructure of an algebraic structure A is a non-empty subset S of A which is itself an algebraic structure under the operation(s) of A . If $S \neq A$ then S is called a proper substructure of A .

Let A be an algebraic structure and $B \subseteq A$ be a substructure of A . The quotient structure of A modulo B , denoted by A/B , is the set of all cosets $a \circ B$ with a ranging over A , with the operation \star defined by $(a \circ B) \star (b \circ B) = (a \circ b) \circ B$, and with the identity elements being $\mathbf{0} \circ B$ and $\mathbf{1} \circ B$.

From Definition 5.14, a ring (respectively, a field) not only can have a subring (respectively, a subfield), but also can have a subgroup (respectively, a subring and a subgroup). We shall see such examples in §5.3.

5.3 Structure of Finite Fields

Finite fields find wide applications in cryptography and cryptographic protocols. The pioneer work of Diffie and Hellman in public-key cryptography, the Diffie-Hellman key exchange protocol [DH76b] (§??), is originally proposed to work in finite fields of a particular form. Since the work of Diffie and Hellman, numerous finite-fields-based cryptosystems and protocols have been proposed: the El-Gamal cryptosystems [ElG85], the Schnorr identification protocol and signature scheme [Sch91], the zero-knowledge undeniable signatures of Chaum, and the zero-knowledge proof protocols of Chaum and Pedersen [CP93], are well-known examples. Some new cryptosystems, such as the Advanced Encryption Standard [NIS01b] (§7.6) and the XTR cryptosystems [LV00], work in finite fields of a more general form. Finite fields also underlie elliptic curves which in turn form the basis of a class of cryptosystems (e.g., [Kob87]).

Let us now conduct a self-contained course in the structure of finite fields.

5.3.1 Finite Fields of Prime Numbers of Elements

Finite fields with the simplest structure are those of orders (i.e., the number of elements) as prime numbers. Yet, such fields have been the most widely used ones in cryptography.

Definition 5.15: Prime Field *A field that contains no proper subfield is called a prime field.*

For example, \mathbb{Q} is a prime field whereas \mathbb{R} is not, since \mathbb{Q} is a proper subfield of \mathbb{R} . But \mathbb{Q} is an infinite field. In finite fields, we shall soon see that a prime field must contain a prime number of elements, that is, must have a prime order.

Definition 5.16: Homomorphism and Isomorphism *Let A, B be two algebraic structures. A mapping $f : A \mapsto B$ is called a homomorphism of A into B if f preserves operations of A . That is, if \circ is an operation of A and \star , an operation of B , then $\forall x, y \in A$, we have $f(x \circ y) = f(x) \star f(y)$. If f is a one-to-one homomorphism of A onto B , then f is called an isomorphism and we say that A and B are isomorphic.*

If $f : A \mapsto B$ is a homomorphism and e is an identity element in A (either additive or multiplicative), then

$$f(e) \star f(e) = f(e \circ e) = f(e),$$

so that $f(e)$ is the identity element in B . Also, for any $a \in A$

$$f(a) \star f(a^{-1}) = f(a \circ a^{-1}) = f(e),$$

so that $f(a^{-1}) = f(a)^{-1}$ for all $a \in A$. Moreover, if the mapping is one-one onto (i.e., A and B are isomorphic), then A and B have the same number of elements. Two isomorphic algebraic structures will be viewed to have the same structure.

Example 5.10: (Isomorphic Algebraic Structures)

- i) Denote by \mathbb{F}_2 the set $\{0, 1\}$ with operations $+$ and \cdot being integer addition modulo 2 and integer multiplication, respectively. Then \mathbb{F}_2 must be a field because it is isomorphic to field B in Example 5.9(2). The isomorphism $f : \mathbb{F}_2 \mapsto B$ is: $f(0) = F$, $f(1) = T$.
- ii) For any prime number p , additive group \mathbb{Z}_{p-1} is isomorphic to multiplicative group \mathbb{Z}_p^* . The isomorphism $f : \mathbb{Z}_{p-1} \mapsto \mathbb{Z}_p^*$ is $f(x) = g^x \pmod{p}$ where g is a generator of \mathbb{Z}_p^* . \square

Clearly, all fields of two elements are isomorphic to each other and hence to \mathbb{F}_2 . A field of two elements is the simplest field: It contains the two necessary elements, namely, the zero-element and the unity-element, and nothing else. Since under isomorphisms, there is no need to differentiate these fields, we can treat \mathbb{F}_2 as the unique field of order 2.

Example 5.11: (Finite Field of Prime Order) Let p be any prime number. Then \mathbb{Z}_p , the integers modulo p , is a finite field of order p (i.e., of p elements) with addition and multiplication modulo p as the field operations. Indeed, we have already shown, in Example 5.8(2) that \mathbb{Z}_p is an additive ring, and in Example 5.1(5) that the non-zero elements of \mathbb{Z}_p , denoted by \mathbb{Z}_p^* , forms a multiplicative group. \square

Definition 5.17: Field \mathbb{F}_p Let p be a prime number. We denote by \mathbb{F}_p the finite field \mathbb{Z}_p .

Let F be any finite field of a prime-order p . Since we can construct a one-one mapping from F onto \mathbb{F}_p (i.e., the mapping is an isomorphism), any finite field of order p is isomorphic to \mathbb{F}_p . As there is no need for us to differentiate fields which are isomorphic to each other, we can harmlessly call \mathbb{F}_p the finite field of order p .

Let A be a finite algebraic structure with additive operation “+”, and let a be any non-zero element in A . Observe the following sequence:

$$a, 2a = a + a, 3a = a + a + a, \dots \quad (5.3.1)$$

Since A is finite, the element a has a finite order and therefore in this sequence there must exist a pair (ia, ja) with $i < j$ being integers and $ja - ia = (j - i)a = \mathbf{0}$.

We should remind the reader to notice Definition 5.4 and Remark 5.1 for the shorthand meaning of writing multiplication ia where i is an integer and a is an algebraic element.

Definition 5.18: Characteristic of an Algebraic Structure *The characteristic of an algebraic structure A , denoted by $\text{char}(A)$, is the least positive integer n such that $na = \mathbf{0}$ for every $a \in A$. If no such positive integer n exists, then A is said to have characteristic 0.*

Theorem 5.4: *Every finite field has a prime characteristic.*

Proof

Let F be a finite field and $a \in F$ be any non-zero element. With $(j - i)a = \mathbf{0}$ and $j > i$ derived from the sequence in (5.3.1) we know F must have a positive characteristic. Let it be n . Since F has at least two elements (i.e., the zero-element and the unity-element), $n \geq 2$. If $n > 2$ were not prime, we could write $n = k\ell$ with $k, \ell \in \mathbb{Z}$, $1 < k, \ell < n$. Then

$$\mathbf{0} = n\mathbf{1} = (k\ell)\mathbf{1} = (k\ell)\mathbf{1}\mathbf{1} = (k\mathbf{1})(\ell\mathbf{1}).$$

This implies either $k\mathbf{1} = \mathbf{0}$ or $\ell\mathbf{1} = \mathbf{0}$ since non-zero elements of F form a multiplicative group (which does not contain $\mathbf{0}$). It follows either $ka\mathbf{1} = (k\mathbf{1})a = \mathbf{0}$ for all $a \in F$ or $\ell a\mathbf{1} = (\ell\mathbf{1})a = \mathbf{0}$ for all $a \in F$, in contradiction to the definition of the characteristic n . \square

5.3.2 Finite Fields Modulo Irreducible Polynomials

The order of a finite prime field is equal to the characteristic of the field. However, this is not the general case for finite fields. A more general form of finite fields can be constructed using polynomials.

5.3.2.1 Polynomials Over an Algebraic Structure

In Chapter 4 we have already used polynomials over integers. Now let us be familiar with polynomials over an abstract algebraic structure.

Definition 5.19: Polynomials Over an Algebraic Structure *Let A be an algebraic structure with addition and multiplication. A polynomial over A is an expression of the form*

$$f(x) = \sum_{i=0}^n a_i x^i$$

where n is a non-negative integer, the coefficients a_i , $0 \leq i \leq n$ are elements in A , and x is a symbol not belonging to A . The coefficient a_n is called the leading coefficient and is not the zero-element in A for $n \neq 0$. The integer n is called the degree of $f(x)$ and is denoted by $n = \deg(f(x)) = \deg(f)$. If the leading coefficient is a_0 , then f is called a constant polynomial. If the leading coefficient is $a_0 = \mathbf{0}$, then f is called the zero-polynomial and is denoted by $f = 0$. We denote by $A[x]$ the set of all polynomials over A .

For $f, g \in A[x]$ with

$$f(x) = \sum_{i=0}^n a_i x^i \quad \text{and} \quad g(x) = \sum_{i=0}^m b_i x^i,$$

we have

$$f(x) + g(x) = \sum_{i=0}^{\max(n,m)} c_i x^i \quad \text{where} \quad c_i = \begin{cases} a_i + b_i & i = 0, 1, \dots, \min(n, m) \\ a_i & i = m+1, \dots, n \\ b_i & i = n+1, \dots, m \end{cases} \quad (5.3.2)$$

and

$$f(x)g(x) = \sum_{k=0}^{n+m} c_k x^k \quad \text{where} \quad c_k = \sum_{\substack{i+j=k \\ 0 \leq i \leq n \\ 0 \leq j \leq m}} a_i b_j \quad (5.3.3)$$

It is easy to see that if A is a ring, then $A[x]$ is a ring with A being a subring of $A[x]$. Addition and multiplication between polynomials over a ring will result in the following relationship on the polynomial degrees:

$$\deg(f + g) \leq \max(\deg(f), \deg(g)),$$

$$\deg(fg) \leq \deg(f) + \deg(g).$$

Now if A is a field, then because a field has no zero-divisors, we will have $c_{n+m} = a_n b_m \neq \mathbf{0}$ for $a_n \neq \mathbf{0}$ and $b_m \neq \mathbf{0}$. So if A is a field, then

$$\deg(fg) = \deg(f) + \deg(g).$$

Let $f, g \in A[x]$ such that $g \neq 0$. Analogous to the case of division between integers (see §4.5.1), we can always write

$$f = gq + r \quad \text{for } q, r \in A[x] \text{ with } \deg(r) < \deg(g). \quad (5.3.4)$$

Example 5.12: Consider $f(x) = x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$, $g(x) = x^3 + x + 1 \in \mathbb{F}_2[x]$. We can compute $q, r \in \mathbb{F}_2[x]$ by long division

$$\begin{array}{r} x^2 + x \\ x^3 + x + 1 \overline{) x^5 + x^4 + x^3 + x^2 + x + 1} \\ \underline{x^5 + x^4 + x^3 + x^2} \\ x^4 \\ \underline{x^4} \\ x^2 + x + 1 \end{array}$$

Therefore $q = x^2 + x$ and $r = x^2 + 1$. □

Definition 5.20: (Irreducible Polynomial) Let A be an algebraic structure. A polynomial $f \in A[x]$ is said to be irreducible over A (or irreducible in $A[x]$, or prime in $A[x]$) if f has a positive degree and $f = gh$ with $g, h \in A[x]$ implies that either g or h is a constant polynomial. A polynomial is said to be reducible over A if it is not irreducible over A .

Notice that the reducibility of a polynomial depends on the algebraic structure over which the polynomial is defined. A polynomial can be reducible over one structure, but is irreducible over another.

Example 5.13: For quadratic polynomial $f(x) = x^2 - 2x + 2$: (i) Discuss its reducibility over the usual infinite algebraic structures; (ii) Investigate its reducibility over finite fields \mathbb{F}_p for any odd prime number p ; (iii) Factor $f(x)$ over \mathbb{F}_p for $p < 10$.

Using the rooting formula in elementary algebra, we can compute the two roots of $f(x) = 0$ as

$$\alpha = 1 + \sqrt{-1}, \quad \beta = 1 - \sqrt{-1}.$$

i) Since $\sqrt{-1}$ is not in \mathbb{R} , $f(x)$ is irreducible over \mathbb{R} (and hence is irreducible over \mathbb{Z} or \mathbb{Q}). But because $\sqrt{-1} = i \in \mathbb{C}$, therefore $f(x)$ is reducible over \mathbb{C} :

$$f(x) = (x - 1 - i)(x - 1 + i).$$

ii) Clearly, $f(x)$ is reducible over \mathbb{F}_p for any odd prime p if and only if $\sqrt{-1}$ is an element in \mathbb{F}_p , or equivalently, -1 is a square number modulo p .

A number x is a square modulo p if and only if there exists $y \pmod{p}$ satisfying $y^2 \equiv x \pmod{p}$. By Fermat's Little Theorem (Theorem 6.10), we know that all $x \pmod{p}$ satisfies $x^{p-1} \equiv 1 \pmod{p}$. For p being an odd prime, Fermat's Little Theorem is equivalent to

$$x^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}, \tag{5.3.5}$$

for all x with $0 < x < p$ (where -1 denotes $p-1$). Then if x is a square modulo p , (5.3.5) becomes

$$x^{\frac{p-1}{2}} \equiv (y^2)^{\frac{p-1}{2}} \equiv y^{p-1} \equiv 1 \pmod{p}.$$

Therefore, we know that (5.3.5) provides a criterion for testing whether x is a square modulo an odd prime p : x is a square (respectively, non-square) modulo p if the test yields 1 (respectively, -1).

To this end we know that for any odd prime p , $f(x)$ is reducible over \mathbb{F}_p if and only if $(-1)^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, and is irreducible if and only if $(-1)^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. In other words, $f(x)$ is reducible (or irreducible) over \mathbb{F}_p if $p \equiv 1 \pmod{4}$ (or $p \equiv 3 \pmod{4}$).

iii) For $p = 2$, $f(x) = x^2 - 2x + 2 = x^2 - 0x + 0 = x^2$ and is reducible over \mathbb{F}_2 .

The only odd prime less than 10 and congruent to 1 modulo 4 is 5. Since $-1 \equiv 4 \equiv 2^2 \pmod{5}$, i.e., $\sqrt{-1} \equiv 2 \pmod{5}$, we can completely factor $f(x)$ over \mathbb{F}_5 :

$$f(x) = (x - 1 - \sqrt{-1})(x - 1 + \sqrt{-1}) = (x - 1 - 2)(x - 1 + 2) = (x + 2)(x + 1).$$

The other square root of -1 in \mathbb{F}_5 is 3. The reader may check that the root 3 will provide the same factorization of $f(x)$ over \mathbb{F}_5 as does the root 2. \square

5.3.2.2 Field Construction Using Irreducible Polynomial

Let us construct finite field using an irreducible polynomial.

Definition 5.21: (The Set $A[x]$ Modulo a Polynomial) Let A be an algebraic structure and let $f, g, q, r \in A[x]$ with $g \neq 0$ satisfy the division expression (5.3.4), we say r is the remainder of f divided by g and denote $r \equiv f \pmod{g}$.

The set of the remainders of all polynomials in $A[x]$ modulo g is called the polynomials in $A[x]$ modulo g , and is denoted by $A[x]_g$.

Analogous to the integers modulo a positive integer, $A[x]_f$ is the set of all polynomials of degrees less than $\deg(f)$.

Theorem 5.5: Let F be a field and f be a non-zero polynomial in $F[x]$. Then $F[x]_f$ is a ring, and is a field if and only if f is irreducible over F .

Proof First, $F[x]_f$ is obviously a ring under addition and multiplication modulo f defined by (5.3.2), (5.3.3) and (5.3.4) with the zero-element and the unity-element the same as those of F .

Secondly, let $F[x]_f$ be a field. Suppose $f = gh$ for g, h being non-constant polynomials in $F[x]$. Then because $0 < \deg(g) < \deg(f)$ and $0 < \deg(h) < \deg(f)$, g and h are non-zero polynomials in $F[x]_f$ whereas f is the zero polynomial in $F[x]_f$. This violates the Closure Axiom for the multiplicative group of $F[x]_f$. So $F[x]_f$ cannot be a field. This contradicts the assumption that $F[x]_f$ is a field.

Finally, let f be irreducible over F . Since $F[x]_f$ is a ring, it suffices for us to show that any non-zero element in $F[x]_f$ has a multiplicative inverse in $F[x]_f$. Let r be a non-zero polynomial in $F[x]_f$ with $\gcd(f, r) = c$. Because $\deg(r) < \deg(f)$ and f is irreducible, c must be a constant polynomial. Writing $r = cs$, we have $c \in F$ and $s \in F[x]_f$ with $\gcd(f, s) = 1$. Analogous to the integer case, we can use the extended Euclid algorithm for polynomials to compute $s^{-1} \pmod{f} \in F[x]_f$. Also since $c \in F$, there exists $c^{-1} \in F$. Thus we obtain $r^{-1} = c^{-1}s^{-1} \in F[x]_f$. \square

For finite field $F[x]_f$, let us call the irreducible polynomial f **definition polynomial** of the field $F[x]_f$.

Theorem 5.6: *Let F be a field of p elements, and f be a degree- n irreducible polynomial over F . Then the number of elements in the field $F[x]_f$ is p^n .*

Proof From Definition 5.21 we know $F[x]_f$ is the set of all polynomials in $F[x]$ of degrees less than $\deg(f) = n$ with the coefficients ranging through F of p elements. There are exactly p^n such polynomials in $F[x]_f$. \square

Corollary 5.5: *For every prime p and for every positive integer n there exists a finite field of p^n elements.* \square

As indicated by Corollary 5.5, for F being a prime field \mathbb{F}_p , the structure of the field $\mathbb{F}_p[x]_f$ is very clear: it is merely the set of all polynomials of degree less than n with coefficients in \mathbb{F}_p . Under isomorphism, we can even say that $\mathbb{F}_p[x]_f$ is the finite field of order p^n .

Example 5.14: (Integer Representation of Finite Field Element) Polynomial $f(x) = x^8 + x^4 + x^3 + x + 1$ is irreducible over \mathbb{F}_2 . The set of all polynomials modulo $f(x)$ over \mathbb{F}_2 forms a field of 2^8 elements; they are all polynomials over \mathbb{F}_2 of degree less than 8. So any element in the field $\mathbb{F}_2[x]_f$ is

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

where $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0 \in \mathbb{F}_2$. Thus, any element in this field can be represented as an integer of 8 binary bits $b_7b_6b_5b_4b_3b_2b_1b_0$, or a byte. In the hexadecimal encoding, we can use a letter to encode an integer value represented by 4 bits:

$$'0' = 0000(= 0), \dots, '9' = 1001(= 9), 'A' = 1010(= 10), \dots, 'F' = 1111(= 15).$$

So the hexadecimal encoding of a byte will be in the interval $['00', 'FF']$. So a field element can be viewed as a byte in this interval.

Conversely, any byte in the interval $['00', 'FF']$ can be viewed as an element in the field $\mathbb{F}_2[x]_f$. For example, the byte 01010111 (or the hexadecimal value '57') corresponds to the element (polynomial)

$$x^6 + x^4 + x^2 + x + 1. \quad \square$$

From Corollary 5.5 and Example 5.14, we can view the field $\mathbb{F}_2[x]_f$ as the field of all non-negative integers up to $\deg(f)$ binary bits. Clearly, this field has $2^{\deg(f)}$ elements. Therefore, for any integer $n > 0$, all non-negative integers up to n binary bits form a finite field of 2^n elements. Let us use “ n -bit binary field” to name this field. Operations in this field follows the operations between polynomials of degrees less than n over \mathbb{F}_2 . Addition is very simple as shown in Example 5.15.

Example 5.15: Let f be a degree-8 irreducible polynomial over \mathbb{F}_2 . In the 8-bit binary field, addition follows polynomial addition by adding coefficients modulo 2 (so $1 + 1 = 0$). For example (in hexadecimal) $'57' + '83' = 'D4'$:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2.$$

So, addition in this field is independent from the definition polynomial f . □

Multiplication in the binary field $\mathbb{F}_2[x]_f$ depends on the definition polynomial f : it is multiplication between two polynomials modulo f . The modulo operation can be done by applying the extended Euclid algorithm for polynomials. Later (in Example 5.16) we shall show another method for achieving field multiplication which is based on a different way for the field representation.

The n -bit binary field is a useful field because of its natural interpretation of integers. A new encryption standard, the Advanced Encryption Standard (AES), works in the 8-bit binary field. We will introduce the AES in Chapter 7.

Finally we notice that in Theorem 5.6 we have never assumed p to be prime. In fact, in Theorem 5.5, F can be any field, and $F[x]_f$ is called an **extended field** from the **underlying subfield** F via **field extension**. Since F can be any field, it can of course be an extended field from another underlying subfield. In many applications of finite fields, we need to know more information about the relation between extended fields and underlying subfields (for example, we will need to know this relation when we study the AES later). Also, a different way for finite fields representation may also ease computation (e.g., the multiplication in Example 5.15 can be eased without using the Euclid algorithm if we use a different field representation). The next section serves the purpose for a better understanding of the structure of finite fields.

5.3.3 Finite Fields Constructed Using Polynomial Basis

This section is intended to provide the knowledge for helping a better understanding of some cryptosystems based on a general form of finite fields. We present it by assuming that the reader is familiar with the knowledge of vector space in linear algebra. However, this section may be skipped without causing difficulty for reading most parts of the rest of this book.

In §5.3.2 we have shown that under isomorphism, field $\mathbb{F}_p[x]_f$ is *the* finite field of order $p^{\deg(f)}$. However, often it may not be very convenient for us to use fields modulo an irreducible polynomial. In this final part of our course in algebraic foundations, let us construct finite fields using the roots of an irreducible polynomial over a finite field F . Fields constructed this way are more frequently used in applications.

Let F be a finite field and n be any positive integer. Let $f(x)$ be an irreducible polynomial over F of degree n . We know that $f(x)$ has exactly n roots in somewhere

since $f(x)$ can be factored into n linear polynomials there. We shall see in a moment that “somewhere” or “there” is exactly the space we are constructing.

Denote these n roots of $f(x) = 0$ by

$$\theta_0, \theta_1, \dots, \theta_{n-1} \quad (5.3.6)$$

Since $f(x)$ is irreducible over F , none of these roots can be in F .

Theorem 5.7: *Let F be any finite field and let $f(x) \in F[x]$ be an irreducible polynomial of degree n over F . Then for θ being any root of $f(x) = 0$, elements*

$$1, \theta, \theta^2, \dots, \theta^{n-1}$$

are linearly independent over F , that is, for $r_i \in F$ with $i = 0, 1, 2, \dots, n-1$:

$$r_0 + r_1\theta + r_2\theta^2 + \dots + r_{n-1}\theta^{n-1} = 0 \text{ implies } r_0 = r_1 = \dots = r_{n-1} = 0. \quad (5.3.7)$$

Proof Let θ be any root of $f(x) = 0$. We know $\theta \neq 1$ since $f(x)$ is irreducible over field F which contains 1. Suppose that the elements $1, \theta, \theta^2, \dots, \theta^{n-1}$ were not linearly independent over F . That is, the linear combination (5.3.7) is possible for some $r_i \in F$ which are not all zero ($i = 0, 1, 2, \dots, n-1$). This is equivalent to θ being a root of

$$r(x) = r_0 + r_1x + r_2x^2 + \dots + r_{n-1}x^{n-1} = 0.$$

With $r_i \in F$ ($i = 0, 1, \dots, n-1$), by Definition 5.21, $r(x)$ is an element in the field $F[x]_f$ and therefore $r(x) = 0$ means $r(x) \equiv 0 \pmod{f(x)}$. Let a_n be the leading coefficient of $f(x)$. Then $a_n \in F$, $a_n \neq 0$ and $a_n^{-1}f(x) \mid r(x)$. But this is impossible since $a_n^{-1}f(x)$ is of degree n while $r(x)$ is of degree less than n , unless $r(x)$ is the zero polynomial. This contradicts the supposed condition that $r_i \in F$ are not all zero ($i = 0, 1, \dots, n-1$). \square

Definition 5.22: (Polynomial Basis) *Let F be a finite field and $f(x)$ be a degree- n irreducible polynomial over F . Then for any root θ of $f(x) = 0$, elements $1, \theta, \theta^2, \dots, \theta^{n-1}$ are called a (polynomial) basis (of a finite vector space) over F .*

We know from a fact in linear algebra that a basis of n elements spans an n -dimension vector space. The spanning uses the scalars in F , that is, the space so spanned has the following structure

$$\left\{ \sum_{i=0}^{n-1} r_i \theta^i \mid r_0, r_1, \dots, r_{n-1} \in F \right\}. \quad (5.3.8)$$

Theorem 5.8: *Let F be a finite field and $f(x)$ be a degree- n irreducible polynomial over F . Then for any root θ of $f(x) = 0$, the vector space in (5.3.8) is a finite field of $(\#F)^n$ elements.*

Proof First, we show that the space in (5.3.8) is a ring. The only non-trivial part is to show that Closure Axiom holds for multiplication. To do so, we note that from

$$f(\theta) = a_n\theta^n + a_{n-1}\theta^{n-1} + \cdots + a_0 = 0 \quad (5.3.9)$$

with $a_n \in F$ and $a_n \neq 0$, we have

$$\theta^n = a_n^{-1}(-a_{n-1}\theta^{n-1} - \cdots - a_0)$$

and so θ^n is a linear combination of the basis $1, \theta, \theta^2, \dots, \theta^{n-1}$. Multiplying θ to (5.3.9), we can further derive that for any positive integer $m \geq n$, θ^m can be expressed as a linear combination of the same basis. Therefore, for any u, v in the space in (5.3.8), uv , as a linear combination of $1, \theta, \dots, \theta^m$ for $m \leq 2(n-1)$, must be a linear combination of the basis $1, \theta, \dots, \theta^{n-1}$, and hence is in the space in (5.3.8). So we have shown Closure Axiom.

Secondly, to show that the space in (5.3.8) is a field, we only need to show that the space does not contain zero-divisors. To do so, we can use the linear independence relation in (5.3.7) and check that for $uv = 0$, either the scalars of u , or those of v , must all be zero, and hence either $u = 0$ or $v = 0$.

Finally, notice that since the spanning process uses $\#F$ elements of F as scalars and the basis of n elements, the space spanned has exactly $(\#F)^n$ elements. \square

Definition 5.23: (Finite Field \mathbb{F}_{q^n}) *Let q be the number of elements in a finite field F . The finite field spanned by a basis of n elements over F is denoted by \mathbb{F}_{q^n} .*

Theorem 5.9: *Let F be a finite field of q elements and let \mathbb{F}_{q^n} be a finite field spanned over F . Then*

- i) *the characteristic of \mathbb{F}_{q^n} is that of F ;*
- ii) *F is a subfield of \mathbb{F}_{q^n} ;*
- iii) *any element $a \in \mathbb{F}_{q^n}$ satisfying $a^q = a$ if and only if $a \in F$.*

Proof Let $1, \theta, \theta^2, \dots, \theta^{n-1}$ be a basis of \mathbb{F}_{q^n} over F .

i) Let $\text{char}(F)$ denote the characteristic of F . Then adding any element in \mathbb{F}_{q^n} to itself $\text{char}(F)$ times we obtain

$$\sum_{i=0}^{n-1} \text{char}(F) r_i \theta^i = \sum_{i=0}^{n-1} 0 \theta^i = 0.$$

Thus $\text{char}(\mathbb{F}_{q^n}) = \text{char}(F)$.

ii) Since the basis contain 1, using scalars in F , any element in F is a linear combination of 1 and hence is a linear combination of the basis.

iii) (\Leftarrow) Consider the subfield $F = \{0\} \cup F^*$ where F^* is a multiplicative group of the non-zero elements. So for any $a \in F$, either $a = 0$ or $a \in F^*$. The former case satisfies $a^q = a$ trivially. For the latter case, by Lagrange's Theorem (Corollary 5.2), $\text{ord}(a) \mid \#F^* = q - 1$ and therefore $a^{q-1} = 1$. So $a^q = a$ is also satisfied.

(\Rightarrow) Any $a \in \mathbb{F}_{q^n}$ satisfying $a^q = a$ must be a root of polynomial $x^q - x = 0$. This polynomial has degree q and therefore has at most q roots in \mathbb{F}_{q^n} including 0. By (ii), F is a subfield of \mathbb{F}_{q^n} , which already contains all the roots of $x^q - x = 0$. No other elements of \mathbb{F}_{q^n} can be a root of $x^q - x$. \square

In our course of spanning the field \mathbb{F}_{q^n} over a field F of q elements, we have never assumed or required that q be a prime number, that is, we have not assumed or required that F be a prime field. The following theorem provides the relationship between F and the field \mathbb{F}_{q^n} spanned over F and stipulates the nature of q .

Theorem 5.10: (Subfield Criterion) *Let p be a prime number. Then F is a subfield of \mathbb{F}_{p^n} if and only if F has p^m elements for m being a positive divisor of n .*

Proof (\Rightarrow) Let F be a subfield of \mathbb{F}_{p^n} . $F = \mathbb{F}_p$ or $F = \mathbb{F}_{p^n}$ are the two trivial cases. Let F be a proper subfield of \mathbb{F}_{p^n} other than \mathbb{F}_p . By Theorem 5.9(i), \mathbb{F}_{p^n} has characteristic p . Consequently F must also have characteristic p . So F contains \mathbb{F}_p as a subfield and is spanned over \mathbb{F}_p by a basis of m elements for some m with $1 \leq m \leq n$. We only need to show $m \mid n$. The two multiplicative groups $\mathbb{F}_{p^n}^*$ and F^* have $p^n - 1$ and $p^m - 1$ elements, respectively. Since the latter is a subgroup of the former, by Lagrange's Theorem (Theorem 5.1), $p^m - 1 \mid p^n - 1$. This is only possible if $m \mid n$.

(\Leftarrow) Let m be a positive proper divisor of n and let F be a field of p^m elements. Since n/m is a positive integer, using a degree- (n/m) irreducible polynomial over F we can span a field of $(p^m)^{n/m} = p^n$ elements. Denote by \mathbb{F}_{p^n} the spanned field, by Theorem 5.9(ii), F is a subfield of \mathbb{F}_{p^n} . \square

Let $f(x)$ be any degree- n irreducible polynomial over \mathbb{F}_p . Review Theorem 5.6, we now know \mathbb{F}_{p^n} is isomorphic to $\mathbb{F}_p[x]_f$. Even though two isomorphic fields should be viewed without essential difference, one can be much easier to work with than the other. Indeed, the ease of proving the Subfield Criterion Theorem for \mathbb{F}_{p^n} provides such a clear evidence. The following example provides another evidence.

Example 5.16: (Field \mathbb{F}_{2^8}) In Example 5.15 we have studied the field $\mathbb{F}_2[x]_{x^8+x^4+x^3+x+1}$ which is the set of all polynomials modulo the irreducible polynomial $x^8+x^4+x^3+x+1$ over \mathbb{F}_2 . That field has 2^8 elements. Now we know that \mathbb{F}_{2^8} is also a field of 2^8 elements and can be represented by the following space

$$\{b_7\theta^7 + b_6\theta^6 + b_5\theta^5 + b_4\theta^4 + b_3\theta^3 + b_2\theta^2 + b_1\theta + b_0\}$$

where θ is a root of (e.g.) the equation $x^8 + x^4 + x^3 + x + 1 = 0$, and the scalars $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0 \in \mathbb{F}_2$. Clearly, these two fields are isomorphic; in particular, we can also use a byte to represent an element in the latter representation of \mathbb{F}_{2^8} .

In Example 5.15 we mentioned that multiplication in $\mathbb{F}_2[x]_{x^8+x^4+x^3+x+1}$ is a bit complicated and needs modulo polynomial which requires the Euclid algorithm for polynomial division. Multiplication in \mathbb{F}_{2^8} spanned from polynomial basis can be easier: straightforward multiplying two elements and representing any resultant terms with θ^i for $i > 7$ using a linear combination of the basis $1, \theta, \dots, \theta^7$.

For example, let us compute $'57' \cdot '83'$, or

$$(\theta^6 + \theta^4 + \theta^2 + \theta + 1) \cdot (\theta^7 + \theta + 1) = \theta^{13} + \theta^{11} + \theta^9 + \theta^8 + \theta^6 + \theta^5 + \theta^4 + \theta^3 + 1.$$

Since

$$\theta^8 + \theta^4 + \theta^3 + \theta + 1 = 0,$$

we have the following linear combinations (notice $-1 = 1$ in \mathbb{F}_2):

$$\theta^8 = \theta^4 + \theta^3 + \theta + 1,$$

$$\theta^9 = \theta^5 + \theta^4 + \theta^2 + \theta,$$

$$\theta^{11} = \theta^7 + \theta^6 + \theta^4 + \theta^3,$$

$$\theta^{13} = \theta^9 + \theta^8 + \theta^6 + \theta^5.$$

Thus,

$$\theta^{13} + \theta^{11} + \theta^9 + \theta^8 + \theta^6 + \theta^5 + \theta^4 + \theta^3 + 1 = \theta^7 + \theta^6 + 1.$$

That is, we have $'57' \cdot '83' = 'C1'$. \square

We now provide a remark as a summary on our study of finite fields.

Remark 5.2: *We have studied two methods for constructing finite fields: field modulo an irreducible polynomial (§5.3.2) and field spanned from a polynomial basis (§5.3.3). In our study of finite fields we have used \mathbb{F}_q to denote a field of the latter construction. However, under isomorphism, two fields of the same number of elements can be viewed without difference. Therefore from now on, we will denote by \mathbb{F}_q any finite field of q elements where q is a prime power.* \square

5.3.4 Primitive Roots

We asserted in §4.8 that the complete factorization of $n - 1$ provides a piece of “internal information” (i.e., auxiliary input for verifying a problem in \mathcal{NP}) for answering whether n is prime with an efficient deterministic algorithm. Now with the knowledge of finite fields, that assertion can be easily proved.

Theorem 5.11: *The multiplicative group $(\mathbb{F}_{p^n})^*$ of the field \mathbb{F}_{p^n} is cyclic.*

Proof By Theorem 5.9(iii), the entire roots of polynomial $x^{p^n-1} - 1 = 0$ forms $(\mathbb{F}_{p^n})^*$. However, the entire roots of this polynomial are the $p^n - 1$ distinct (non-trivial) roots of 1, spread over the unity circle. So there exists a $(p^n - 1)$ -th root of 1, which generates the group $(\mathbb{F}_{p^n})^*$. Hence $(\mathbb{F}_{p^n})^*$ is cyclic. \square

Definition 5.24: (Primitive Root) *A multiplicative generator of the group $(\mathbb{F}_{p^n})^*$ is called a primitive root of the field \mathbb{F}_{p^n} .*

Theorem 5.12: *Let n be a positive integer with $n - 1 = r_1 r_2 \cdots r_k$ as the complete prime factorization of $n - 1$ (some of the prime factors may repeat). Then n is prime if and only if there exists a positive integer $a < n$ such that $a^{n-1} \equiv 1 \pmod{n}$ and $a^{(n-1)/r_i} \not\equiv 1 \pmod{n}$ for $i = 1, 2, \dots, k$.*

Proof (\Rightarrow) If n is prime, then by Theorem 5.11, the group $(\mathbb{F}_n)^*$ is cyclic and has a generator which is an $(n - 1)$ -th root of 1. Denoting by a this root, then a satisfies the conditions in the theorem statement.

(\Leftarrow) Let integer $a < n$ satisfy the conditions in the theorem statement. Then a, a^2, \dots, a^{n-1} are solutions of $x^{n-1} - 1 \equiv 0 \pmod{n}$. For any $1 \leq i < j \leq n - 1$, it is necessary $a^i \not\equiv a^j \pmod{n}$. Suppose otherwise $a^{j-i} \equiv 1 \pmod{n}$ for some i, j with $0 < j - i < n - 1$; then by Definition 5.9 $\text{ord}(a) | j - i | n - 1$, contradicting to the conditions in the theorem statement. Now we know that $\langle a \rangle$ is a multiplicative group of $n - 1$ elements (multiplication modulo n). This group can contain at most $\phi(n)$ elements. So $\phi(n) = n - 1$. Hence n is prime by definition of Euler's function (Definition 5.11). \square

Theorem 5.12 suggests an efficient algorithm for finding a primitive root modulo a prime p , i.e., a generator of the group \mathbb{Z}_p^* . The algorithm is specified in Algorithm 5.1.

By Theorem 5.2(4), we know that in the group \mathbb{Z}_p^* there are exactly $\phi(p - 1)$ elements of order $p - 1$, and these elements are generators of the group. Therefore Algorithm 5.1 is expected to terminate in

$$\frac{p - 1}{\phi(p - 1)} < 6 \log \log p - 1$$

(see e.g., page 65 of [MvOV97]) steps of recursive calls. Since the number of prime factors of $p - 1$ is bounded by $\log p$, the time complexity of the algorithm is bounded by $O_B((\log p)^4 \log \log p)$.

5.3.5 Field Construction Using Normal Basis

The roots of an irreducible polynomial can also form a different set of basis for field construction. Because this different way of field construction is actually more frequently used in cryptographic applications (for example, [LV00]), we should provide

Algorithm 5.1: Random Primitive Root Modulo Prime

INPUT p : a prime; q_1, q_2, \dots, q_k : all prime factors of $p - 1$;
 OUTPUT g : a random primitive root modulo p .

PrimitiveRoot(p, q_1, q_2, \dots, q_k)

1. pick $g \in_U [2, p - 1]$;
2. for ($i = 1, i++, k$) do
 if ($g^{(p-1)/q_i} \equiv 1 \pmod{p}$) return(PrimitiveRoot(p, q_1, q_2, \dots, q_k));
3. return(g).

Figure 5.1.

a brief investigation of such a basis. To do so, we first show that the n roots of irreducible polynomial $f(x) \in \mathbb{F}_p[x]$ listed in (5.3.6) have two useful properties.

Theorem 5.13: *Let $f(x)$ be a degree- n irreducible polynomial over \mathbb{F}_p , and let $\theta_0, \theta_1, \dots, \theta_{n-1}$ be the n roots of $f(x) = 0$. Then these n roots have the following two properties:*

1. *they can be written as*

$$\theta, \theta^p, \dots, \theta^{p^{n-1}}; \quad (5.3.10)$$

for θ being any root of $f(x) = 0$;

2. *they are all distinct.*

We need first show two facts.

Lemma 5.1: *Let F be a finite field of characteristic p . Then*

- i) *For any $a \in F$, $a^p = a$;*
- ii) *For any $a, b \in F$ and any positive integer $k \in \mathbb{Z}$, $(a + b)^{p^k} = a^{p^k} + b^{p^k}$.*

Proof

For (i), since p is prime, F^* is a multiplicative group of $p - 1$ elements. So $a^p = a$ follows Fermat's Little Theorem (Theorem 6.10).

For (ii), note that in the binomial unfolding of $(a + b)^p$, the general form of a coefficient is (for $1 \leq i \leq p$)

$$\binom{p}{i} = \left(= \frac{p!}{i!(p-i)!} \right).$$

For each $1 < i < p$, because the prime p cannot be divided by any factors of $i!(p-i)!$, the coefficient is a multiple of p . Consequently, in F of characteristic p , any element multiplying with such a coefficient (review Definition 5.4 for the meaning of multiplication between an integer and a field element) is $\mathbf{0}$. Therefore, in F , $(a + b)^p = a^p + b^p$. It is trivial to extend this result to $(a + b)^{p^k} = a^{p^k} + b^{p^k}$ for any positive integer k . \square

Now let us prove Theorem 5.13.

For Property 1. Let θ be any root of $f(x) = 0$, applying Lemma 5.1 we have

$$f(\theta^p) = \sum_{i=0}^n a_i (\theta^p)^i = \sum_{i=0}^n a_i^p (\theta^i)^p = \left(\sum_{i=0}^n a_i \theta^i \right)^p = f(\theta)^p = 0.$$

Clearly, this holds for every member listed in (5.3.6). Thus, the members listed in (5.3.10) are roots of $f(x) = 0$ too.

Let us now show Property 2.

Suppose on the contrary, $\theta^{p^j} = \theta^{p^k}$ for some integers j and k with $0 \leq j < k \leq n-1$. Raising this element to the power p^{n-k} , we get

$$\theta^{p^{n-k+j}} = \theta^{p^n} = \theta.$$

Therefore θ is also a root of polynomial $g(x) = x^{p^{n-k+j}} - x = 0$. Notice that both $f(x)$ and $g(x)$ are over \mathbb{F}_p and $f(x)$ is irreducible over \mathbb{F}_p , and so sharing a root between $f(x)$ and $g(x)$ implies $a_n^{-1}f(x)|g(x)$ where $a_n \in \mathbb{F}_p$ is the (non-zero) leading coefficient of $f(x)$. Thus, every root of $f(x) = 0$ must be a root of $g(x) = 0$.

Now consider \mathbb{F}_{p^n} as a field construction over \mathbb{F}_p using a basis formed by the roots of $f(x) = 0$. Since the roots of $f(x)$ is a subset of the roots of $g(x) = 0$, \mathbb{F}_{p^n} as a vector space spanned by such a basis must be a sub vector space spanned by a basis constructed using the roots of $g(x)$. By Theorem 5.9(iii), field $\mathbb{F}_{p^{n-k+j}}$ is a subfield containing all roots of $g(x)$. To this end we have discovered that \mathbb{F}_{p^n} is a subfield of $\mathbb{F}_{p^{n-k+j}}$. By “Subfield Criterion” (Theorem 5.10), this is only possible if $n|n-k+j$. But $n > n-k+j$. We arrive at a contradiction.

So we have proved Theorem 5.13. \square

Without loss of generality, we can equalize the roots listed (5.3.6) and those listed in (5.3.10) as

$$(\theta_0, \theta_1, \dots, \theta_{n-1}) = (\theta_0, \theta_0^p, \dots, \theta_0^{p^{n-1}}). \quad (5.3.11)$$

Similar to the case of spanning fields using a polynomial basis, in our investigation of these two properties of roots of irreducible polynomials, there is no need for us to assume that p be a prime number. As a matter of fact, Lemma 5.1 will remain true for p being a prime power. The proof of Property 2 of Theorem 5.13 also works for p being a prime power. Therefore, if $f(x)$ is an irreducible polynomial over a non-prime field, the two properties of the roots of $f(x) = 0$ can still be derived.

If the n roots in (5.3.11) are linearly independent, that is,

$$\sum_{i=0}^{n-1} r_j \theta^{p^i} = 0 \quad \text{implies} \quad r_0 = r_1 = \cdots = r_{n-1} = 0, \quad (5.3.12)$$

then $\theta, \theta^p, \dots, \theta^{p^{n-1}}$ indeed form a basis of an n -dimension vector space.

Definition 5.25: (Normal Basis) Let F be a finite field and $f(x)$ be a degree- n irreducible polynomial over F . Then the n distinct and linearly independent roots of $f(x) = 0$ in form of (5.3.11) is called a normal basis (of a finite vector space) over F .

In any vector space, any two sets of basis are equivalent, that is, any element in one basis can be expressed as a linear combination of the other basis. Therefore, the normal basis span the same vector space in (5.3.8).

At the end of our course in finite fields, we provide two examples to demonstrate the general existence of normal basis.

Example 5.17: Let p be any prime number such that polynomial

$$f(x) = (x^{n+1} - 1)/(x - 1) = x^n + x^{n-1} + \cdots + 1 \in \mathbb{F}_p[x]$$

is irreducible over \mathbb{F}_p . Show that the n roots of $f(x) = 0$ forms a normal basis over \mathbb{F}_p .

It suffices for us to show that the n distinct roots of $f(x) = 0$ in the form of (5.3.11) are linearly independent. To do so, we notice that for θ being any root of $f(x)$, we have

$$\theta f(\theta) = \theta^{n+1} + (\theta^n + \cdots + \theta + 1) - 1 = \theta^{n+1} - 1 = 0.$$

So every root of $f(x)$ satisfies $\theta^{n+1} = 1$. In fact, they are exactly the n distinct roots of 1 spread on the unity circle (that is why the polynomial $f(x)(x - 1) = x^{n+1} - 1$ is called a cyclotomic polynomial). So there exists a root θ as a multiplicative generator that generates all these roots. Consequently, the roots in the form of (5.3.10) are merely a re-arrangement (i.e., a **permutation**) of

$$\theta, \theta^2, \dots, \theta^n.$$

The linearly independence of these roots follows that of

$$1, \theta, \theta^2, \dots, \theta^{n-1}$$

which we have established in Theorem 5.7. \square

Example 5.18: Let p be any prime number. (i) Show that the roots of any irreducible quadratic function $f(x) = ax^2 + bx + c \in \mathbb{F}_p[x]$ (with $a \neq 0$) form a normal basis of \mathbb{F}_{p^2} over \mathbb{F}_p if and only if $b \neq 0$; and (ii) Construct \mathbb{F}_{p^2} over \mathbb{F}_p .

i) Let α, β be the two roots of $f(x) = 0$. From the relationship between the roots and the coefficients of a quadratic function, we know $\alpha + \beta = -a^{-1}b \in \mathbb{F}_p$, $\alpha\beta = a^{-1}c \in \mathbb{F}_p$.

(\Rightarrow) Let α, β be a normal basis of \mathbb{F}_{p^2} . If $b = 0$ then $\alpha + \beta = 0$. This contradicts the fact that α and β are linearly independent.

(\Leftarrow) Let $b \neq 0$. From $\alpha + \beta = -a^{-1}b$ we have $\alpha^2 = \alpha(-a^{-1}b - \alpha) = -a^{-1}b\alpha - \alpha^2$. So with $b \neq 0$ and $\alpha\beta \in \mathbb{F}_p$, we have $\alpha^2 \notin \mathbb{F}_p$. Suppose there exists $r, s \in \mathbb{F}_p$ with $r \neq 0$ or $s \neq 0$ satisfying $r\alpha + s\beta = 0$. Without loss of generality, suppose $r \neq 0$. Then $\alpha^2 = -r^{-1}s\alpha\beta$. Since $\alpha\beta \in \mathbb{F}_p$, we derive $\alpha^2 \in \mathbb{F}_p$. This contradicts what we have just shown $\alpha^2 \notin \mathbb{F}_p$. So $r = 0$. Analogously we can derive $s = 0$. Thus, α, β are linearly independent and hence do form a normal basis in \mathbb{F}_{p^2} .

ii) $\mathbb{F}_{p^2} = \{ r\alpha + s\beta \mid 0 \leq r, s < p \}$. \square

5.4 Chapter Summary

Cryptographic algorithms and protocols process messages. Encoding (encryption) and the necessary decoding (decryption) operations must transform messages to messages so that the transformation obeys a *closure property* inside the space of messages. In general, such a closure property is obtained by interpreting messages into elements in certain algebraic structures which have a closed property, e.g., the Closure Axiom which is generally obeyed by the usual algebraic structures group, ring, and field. For example, we have seen that for any positive integer n , all non-negative integers up to n binary bits form a finite field of 2^n elements, i.e., the structure is closed in addition and multiplication. Algebraic structures with the closure property generally provide the basic building blocks for constructing cryptographic algorithms and protocols.

Our course in algebraic structure provided in this chapter is not only self-contained for reference purpose for most readers, but also accompanied with plenty of digestion and explanation material so that an in-depth understanding of these subjects can be achieved by more mathematically inclined readers.

Chapter 6

NUMBER THEORY

In this chapter we study some basic facts in number theory which have important relevance to cryptography.

6.1 Congruences and Residue Classes

In §4.5.5 we have defined congruence system modulo a positive integer $n > 1$ and studied a few properties of such systems. Here we shall study a few more facts of the congruence systems.

Theorem 6.1: *For integer $n > 1$, the relation of congruence $(\text{mod } n)$ is **reflexive**, **symmetric** and **transitive**: That is, for every $a, b, c \in \mathbb{Z}$,*

- i) $a \equiv a \pmod{n}$;
- ii) If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$;
- iii) If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$. □

A relation having the three properties in Theorem 6.1 is called an **equivalence relation**. It is well known that an equivalence relation over a set partitions the set into **equivalence classes**. Let us denote by “ \equiv_n ” the equivalence relation of congruence modulo n . This relation is defined over the set \mathbb{Z} , and therefore it partitions \mathbb{Z} into exactly n equivalence classes, each class contains integers which are congruent to an integer modulo n . Let us denote these n classes by

$$\overline{0}, \overline{1}, \dots, \overline{n-1},$$

where

$$\overline{a} = \{x \in \mathbb{Z} \mid x \pmod{n} \equiv a\}. \quad (6.1.1)$$

We call each of them a **residue class** modulo n . Clearly, we can view

$$\mathbb{Z}_n = \{\overline{0}, \overline{1}, \dots, \overline{n-1}\}. \quad (6.1.2)$$

On the other hand, if we consider \mathbb{Z} to be a (trivial) subset of \mathbb{Z} , then coset $n\mathbb{Z}$ (Definition 5.7) is the set all integers which are multiples of n (consider multiplication as the group operation), i.e.,

$$n\mathbb{Z} = \{0, \pm n, \pm 2n, \dots\}. \quad (6.1.3)$$

Now consider quotient group (Definition 5.8) with addition as the group operation:

$$\mathbb{Z}/(n\mathbb{Z}) = \{x + n\mathbb{Z} \mid x \in \mathbb{Z}\}. \quad (6.1.4)$$

If we unfold (6.1.4) using $n\mathbb{Z}$ in (6.1.3), we have

$$\begin{aligned} \mathbb{Z}/(n\mathbb{Z}) &= \{x + n\mathbb{Z} \mid x \in \mathbb{Z}\} \\ &= \{0 + \{0, \pm n, \pm 2n, \dots\}, \\ &\quad 1 + \{0, \pm n, \pm 2n, \dots\}, \\ &\quad 2 + \{0, \pm n, \pm 2n, \dots\}, \\ &\quad \dots, \\ &\quad (n-1) + \{0, \pm n, \pm 2n, \dots\}\} \\ &= \{\{0, \pm n, \pm 2n, \dots\}, \\ &\quad \{1, \pm n + 1, \pm 2n + 1, \dots\}, \\ &\quad \{2, \pm n + 2, \pm 2n + 2, \dots\}, \\ &\quad \dots, \\ &\quad \{(n-1), \pm n + (n-1), \pm 2n + (n-1), \dots\}\}. \end{aligned} \quad (6.1.5)$$

There are only n distinct elements in the structure (6.1.5). No more case is possible. For example

$$n + \{0, \pm n, \pm 2n, \dots\} = \{0, \pm n, \pm 2n, \dots\},$$

and

$$(n+1) + \{0, \pm n, \pm 2n, \dots\} = \{1, \pm n + 1, \pm 2n + 1, \dots\},$$

and so on. Comparing (6.1.2) and (6.1.5) with noticing the definition of \bar{a} in (6.1.1), we now know exactly that for $n > 1$:

$$\mathbb{Z}_n = \mathbb{Z}/(n\mathbb{Z}).$$

$\mathbb{Z}/(n\mathbb{Z})$ is the standard notation (in fact, the definition) for the residue classes modulo n , although for presentation convenience, in this book we will always use the short notation \mathbb{Z}_n in place of $\mathbb{Z}/(n\mathbb{Z})$.

Theorem 6.2: For any $a, b \in \mathbb{Z}$, define addition and multiplication between the residue classes \bar{a} and \bar{b} by

$$\bar{a} + \bar{b} = \overline{a + b}, \quad \bar{a} \cdot \bar{b} = \overline{ab}.$$

Then for any $n > 1$, the mapping $f : \mathbb{Z} \mapsto \mathbb{Z}_n$ defined by “ $(\text{mod } n)$ ” is a homomorphism from \mathbb{Z} onto \mathbb{Z}_n . \square

6.1.1 Congruent Properties for Arithmetic in \mathbb{Z}_n

The homomorphism from \mathbb{Z} onto \mathbb{Z}_n means that arithmetic in \mathbb{Z}_n (arithmetic modulo n) inheres the properties of arithmetic in \mathbb{Z} , as shown in the following theorem.

Theorem 6.3: *For integer $n > 1$, if $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a \pm c \equiv b \pm d \pmod{n}$ and $ac \equiv bd \pmod{n}$.*

Although the statements in this theorem hold trivially as an immediate result of the homomorphic relationship between \mathbb{Z} and \mathbb{Z}_n , we provide a proof which is based on purely using the properties of arithmetic in \mathbb{Z}_n .

Proof If $n|a - b$ and $n|c - d$ then $n|(a \pm c) - (b \pm d)$.

Also $n|(a - b)(c - d) = (ac - bd) - b(c - d) - d(a - b)$. So $n|(ac - bd)$. \square

The properties of the arithmetic in \mathbb{Z}_n shown in Theorem 6.3 are called **congruent** properties, meaning: performing the same calculation on both sides of an equation derives a new equation. However, Theorem 6.3 has left out division. Division in \mathbb{Z} has the congruent property as follows:

$$\forall d \neq 0 : ad = bd \text{ implies } a = b. \quad (6.1.6)$$

The counterpart congruent property for division in \mathbb{Z}_n will take a formula which is slightly different from (6.1.6). Before we find out what this formula is, let us provide an explanation on (6.1.6) in \mathbb{Z} . We may imagine that \mathbb{Z} is the case of \mathbb{Z}_n for $n = \infty$, and that ∞ is divisible by any integer and the resultant quotient is still ∞ . Thus, we may further imagine that the first equation in (6.1.6) holds in terms of modulo ∞ while the second equation holds in terms of modulo ∞/d . Since $\infty/d = \infty$, the two equations in (6.1.6) take the same formula. This congruent property for division in \mathbb{Z} is inherited into \mathbb{Z}_n in the following formula.

Theorem 6.4: *For integer $n > 1$ and $d \neq 0$, if $ad \equiv bd \pmod{n}$ then $a \equiv b \pmod{\frac{n}{\gcd(d, n)}}$.*

Proof Denote $k = \gcd(d, n)$. Then $n|(ad - bd)$ implies $(n/k)|(d/k)(a - b)$. Since $\gcd(d/k, n/k) = 1$, we know $(n/k)|(k/k)(a - b)$ implies $(n/k)|(a - b)$. \square

To this end we know that the arithmetic in \mathbb{Z}_n fully preserves the congruent properties of the arithmetic in \mathbb{Z} . Consequently, we have

Corollary 6.1: *If $f(x)$ is a polynomial over \mathbb{Z} , and $a \equiv b \pmod{n}$ for $n > 1$, then $f(a) \equiv f(b) \pmod{n}$.* \square

6.1.2 Solving Linear Congruence in \mathbb{Z}_n

In Theorem 4.2 we have defined the multiplicative inverse modulo n and shown that for an integer a to have the multiplicative inverse modulo n , i.e., a unique

number $x < n$ satisfying $ax \equiv 1 \pmod{n}$, it is necessary and sufficient for a to satisfy $\gcd(a, n) = 1$. The following theorem provides the condition for general case of solving linear congruence equation.

Theorem 6.5: *For integer $n > 1$, a necessary and sufficient condition that the congruence*

$$ax \equiv b \pmod{n}, \quad (6.1.7)$$

be solvable is that $\gcd(a, n) | b$.

Proof By Definition 4.4, the congruence (6.1.7) is the linear equation

$$ax + kn = b, \quad (6.1.8)$$

for some integer k .

(\Rightarrow) Let (6.1.8) hold. Since $\gcd(a, n)$ divides the left-hand side, it must divide the right-hand side.

(\Leftarrow) For a and n , using Extended Euclid Algorithm (Algorithm 4.2) we can compute

$$a\lambda + \mu n = \gcd(a, n).$$

Since $b/\gcd(a, n)$ is an integer, multiplying this integer to both sides, we obtain (6.1.8) or (6.1.7), where $x = \frac{\lambda b}{\gcd(a, n)} \pmod{n}$ is one solution. \square

It is easy to check that given solution x for (6.1.7),

$$x + \frac{ni}{\gcd(a, n)} \pmod{n} \quad \text{for } i = 0, 1, 2, \dots, \gcd(a, n) - 1$$

are $\gcd(a, n)$ different solutions less than n . Clearly, $\gcd(a, n) = 1$ is the condition for the congruence (6.1.8) to have a unique solution less than n .

Example 6.1: Congruence

$$2x \equiv 5 \pmod{10}$$

is unsolvable since $\gcd(2, 10) = 2 \nmid 5$. In fact, the left-hand side, $2x$, must be an even number, while the right-hand side, $10k + 5$, can only be an odd number, and so trying to solve this congruence is an attempt to equalize an even number to an odd number, which is of course impossible.

On the other hand, congruence

$$6x \equiv 18 \pmod{36}$$

is solvable because $\gcd(6, 36) | 18$. The six solutions are 3, 9, 15, 21, 27, and 33. \square

Theorem 6.6: *For integer $n > 1$, if $\gcd(a, n) = 1$, then $ai + b \not\equiv aj + b \pmod{n}$ for all b, i, j such that $0 \leq i < j < n$.*

Proof Suppose on the contrary $ai + b \equiv aj + b \pmod{n}$. Then by Theorem 6.4 we have $i \equiv j \pmod{n}$, a contradiction to $0 \leq i < j < n$. \square

This property implies that for a satisfying $\gcd(a, n) = 1$, $ai + b \pmod{n}$ ($i = 0, 1, \dots, n-1$) is a **complete residue system** modulo n .

6.1.3 The Chinese Remainder Theorem

We have studied the condition for solving a single linear congruence in the form of (6.1.7). Often we will meet the problem of solving a system of simultaneous linear congruences with different moduli:

$$\begin{aligned} a_1x &\equiv b_1 \pmod{n_1} \\ a_2x &\equiv b_2 \pmod{n_2} \\ &\cdot \\ &\cdot \\ &\cdot \\ a_rx &\equiv b_r \pmod{n_r} \end{aligned} \tag{6.1.9}$$

where $a_i, b_i \in \mathbb{Z}$ with $a_i \neq 0$ for $i = 1, 2, \dots, r$.

For this system of congruences to be solvable it is clearly necessary for each congruence to be solvable. So for $i = 1, 2, \dots, r$ and denoting

$$d_i = \gcd(a_i, n_i),$$

by Theorem 6.5, it is necessary $d_i | b_i$. With this being the case, the congruent properties for multiplication (Theorem 6.3) and for division (Theorem 6.4) allow us to transform the system (6.1.9) into the following linear congruence system which is equivalent to but simpler than the system (6.1.9):

$$\begin{aligned} x &\equiv c_1 \pmod{m_1} \\ x &\equiv c_2 \pmod{m_2} \\ &\cdot \\ &\cdot \\ &\cdot \\ x &\equiv c_r \pmod{m_r} \end{aligned} \tag{6.1.10}$$

where for $i = 1, 2, \dots, r$:

$$m_i = n_i/d_i$$

and

$$c_i = (b_i/d_i)(a_i/d_i)^{-1} \pmod{m_i}.$$

Notice that $(a_i/d_i)^{-1} \pmod{m_i}$ exists since $\gcd(a_i/d_i, m_i) = 1$ (review Theorem 4.2).

In linear algebra, the system (6.1.10) can be represented by the following vector-space version:

$$A\vec{X} = \vec{C} \quad (6.1.11)$$

where

$$A = \begin{pmatrix} \bar{1}_{m_1} & & & \\ & \bar{1}_{m_2} & & \\ & & \ddots & \\ & & & \ddots & \\ & & & & \bar{1}_{m_r} \end{pmatrix}, \quad (6.1.12)$$

$$\vec{X} = \begin{pmatrix} x \\ x \\ \vdots \\ \vdots \\ x \end{pmatrix}, \quad (6.1.13)$$

$$\vec{C} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_r \end{pmatrix}. \quad (6.1.14)$$

Since the i -th equation (for $i = 1, 2, \dots, r$) in the system (6.1.10) is in terms of modulo m_i , in the diagonal part of the matrix A , $\bar{1}_{m_i}$ denotes the residue class 1 modulo m_i , that is,

$$\bar{1}_{m_i} = k_i m_i + 1 \quad (6.1.15)$$

for some integer k_i ($i = 1, 2, \dots, r$). The blank part of A represents 0.

Thus, given any r -dimension vector \vec{C} , the problem of solving the system (6.1.10), or its vector-space version (6.1.11), boils down to that of identifying the diagonal matrix A , or in other words, finding the residue class 1 modulo m_i as required in (6.1.15) for $i = 1, 2, \dots, r$. We know from a fact in linear algebra that if the matrix A exists, then because none of the elements in its diagonal line is zero, the matrix has the full rank r and consequently, there *exists* a *unique* solution.

When the moduli in (6.1.10) are pairwise relatively prime to each other, it is not difficult to find a system of residue classes 1. This is according to the useful Chinese Remainder Theorem (CRT).

Theorem 6.7: (Chinese Remainder Theorem) *For the linear congruence system (6.1.10), if $\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq r$, then there exists $\bar{1}_{m_i}$ satisfying*

$$\bar{1}_{m_i} \equiv 0 \pmod{m_j}. \quad (6.1.16)$$

Consequently, there exists $x \in \mathbb{Z}_M$ as the unique solution to the system (6.1.10) where $M = m_1 m_2 \cdots m_r$.

Proof We prove first the existence and then the uniqueness of the solution.

Existence

For each $i = 1, 2, \dots, r$, $\gcd(m_i, M/m_i) = 1$. Therefore by Theorem 4.2, there exists $y_i \in \mathbb{Z}_{m_i}$ satisfying

$$(M/m_i)y_i \equiv 1 \pmod{m_i}. \quad (6.1.17)$$

Moreover, for $j \neq i$, because $m_j | (M/m_i)$, we have

$$(M/m_i)y_i \equiv 0 \pmod{m_j}. \quad (6.1.18)$$

So $(M/m_i)y_i$ is exactly the number that we are looking for to play the role of $\bar{1}_{m_i}$. Let

$$x = \sum_{i=1}^r \bar{1}_{m_i} c_i \pmod{M}. \quad (6.1.19)$$

Then x is a solution to the system (6.1.10) and is a residue class modulo M .

Uniqueness

View the linear system defined by (6.1.11), (6.1.12), (6.1.13) and (6.1.14) such that the elements of the matrix A and those of the vector \vec{C} are all in \mathbb{Z} (i.e., they are all integers). Notice that in \mathbb{Z}

$$\det(A) = \bar{1}_{m_1} \bar{1}_{m_2} \cdots \bar{1}_{m_r} \neq 0. \quad (6.1.20)$$

This means that the r columns of the matrix A form a basis for the r -dimension vector space $\underbrace{\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}}_r$ (this basis is similar to a so-called “natural basis” in

linear algebra where the only non-zero element in any basis-vector is 1). Therefore, for any vector $\vec{C} \in \underbrace{\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}}_r$, the system (6.1.11) has a unique solution

$\vec{X} \in \underbrace{\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}}_r$. We have seen in the existence part of the proof that the

unique elements of \vec{X} are given by (6.1.19). \square

Theorem 6.7 is constructive, that is, it has constructed an algorithm for finding the solution to the system (6.1.10) (see Algorithm 6.1)

In Algorithm 6.1, the only time-consuming part is in step 2(a). This can be done by applying the Extended Euclid Algorithm (Algorithm 4.2). Considering $m_i < M$ for $i = 1, 2, \dots, r$, the time complexity of Algorithm 6.1 is $O_B(r(\log M)^2)$.

Algorithm 6.1: Chinese Remainder

INPUT integer tuple (m_1, m_2, \dots, m_r) , pairwise relatively prime;
integer tuple $(c_1 \pmod{m_1}, c_2 \pmod{m_2}, \dots, c_r \pmod{m_r})$;
OUTPUT integer $x < M = m_1 m_2 \cdots m_r$ satisfying the system (6.1.10).

1. $M \leftarrow m_1 m_2 \cdots m_r$;
2. for (i from 1 to r) do
 - (a) $y_i \leftarrow (M/m_i)^{-1} \pmod{m_i}$; (* by Extended Euclid Algorithm *)
 - (b) $\bar{1}_{m_i} \leftarrow y_i M/m_i$;
3. return($\sum_{i=1}^r \bar{1}_{m_i} c_i \pmod{M}$).

Figure 6.1.

It is also easy to see the following results from Theorem 6.7:

- i) every $x \in \mathbb{Z}_M$ yields a vector $\vec{C} \in \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_r}$;
- ii) elements 0 and 1 in \mathbb{Z}_M yield $\vec{0}$ and $\vec{1}$ in $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_r}$, respectively;

$$\text{iii) for } x, x' \text{ yielding } \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_r \end{pmatrix}, \begin{pmatrix} c'_1 \\ c'_2 \\ \vdots \\ c'_r \end{pmatrix}, \text{ respectively, } x \circ x' \text{ yields}$$

$$\begin{pmatrix} c_1 \circ c'_1 \pmod{m_1} \\ c_2 \circ c'_2 \pmod{m_2} \\ \vdots \\ c_r \circ c'_r \pmod{m_r} \end{pmatrix},$$

where \circ denotes the arithmetic in appropriate spaces.

Thus, we have also proven the following theorem.

Theorem 6.8: *If $\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq r$, then for $M = m_1 m_2 \cdots m_r$, \mathbb{Z}_M is isomorphic to $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_r}$, and the isomorphism*

$$f(x) : \mathbb{Z}_M \mapsto \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_r}$$

is

$$f(x) = (x \pmod{m_1}, x \pmod{m_2}, \dots, x \pmod{m_r}).$$

□

Theorem 6.8 is very useful in the study of cryptographic systems or protocols which use groups modulo composite integers. In many places in the rest of this book we will need to make use of the isomorphism between \mathbb{Z}_n^* and $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$ where $n = pq$ with p, q prime numbers. For example, we will make use of a property that the non-cyclic group \mathbb{Z}_n^* is generated by two generators of the cyclic groups \mathbb{Z}_p^* and \mathbb{Z}_q^* , respectively.

Let us now look at an application of the Chinese remainder theorem: a calculation is made easy by applying the isomorphic relationship.

Example 6.2: At this stage we do not yet know how to compute square root modulo an integer (we will study the techniques in §6.5). However in some cases a square number in some space (such as in \mathbb{Z}) is evident and so square rooting in that space is easy without need of using modulo arithmetic. Let us apply Theorem 6.8 to compute one of the square roots of 29 in \mathbb{Z}_{35} .

Limited to our knowledge for the moment, it is not evident to us that 29 is a square number in \mathbb{Z}_{35} and so for the time being we do not know how to root it directly. However, if we apply Theorem 6.8 and map 29 to the isomorphic space $\mathbb{Z}_5 \times \mathbb{Z}_7$, we have

$$29 \pmod{5} \mapsto 4, \quad 29 \pmod{7} \mapsto 1,$$

that is, the image is $(4, 1)$. Both 4 and 1 are evident square numbers with 2 being a square root of 4 and 1 being a square root of 1. By isomorphism, we know one of the square roots of 29 in \mathbb{Z}_{35} corresponds to $(2, 1)$ in $\mathbb{Z}_5 \times \mathbb{Z}_7$. Applying the Chinese remainder algorithm (Algorithm 6.1), we obtain

$$\overline{1}_5 = 21, \quad \overline{1}_7 = 15,$$

and

$$\sqrt{29} \equiv 21 \cdot 2 + 15 \cdot 1 \equiv 22 \pmod{35}.$$

Indeed, $22^2 = 484 \equiv 29 \pmod{35}$. □

As a matter of fact, 29 has four distinct square roots in \mathbb{Z}_{35}^* . For an exercise, the reader may find the other three square roots of 29.

6.2 The Euler's Phi Function

We have defined the Euler's phi function in Definition 5.11. Now let us study some useful properties of it.

Lemma 6.1: *Let $\phi(n)$ be Euler's phi function defined in Definition 5.11. Then*

- i) $\phi(1) = 1$.
- ii) If p is prime then $\phi(p) = p - 1$.
- iii) The Euler's phi function is multiplicative. That is, if $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.
- iv) If $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ is the prime factorization of n , then

$$\phi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right).$$

Proof (i) and (ii) are trivial from Definition 5.11.

iii) Since $\phi(1) = 1$, the equation $\phi(mn) = \phi(m)\phi(n)$ holds when either $m = 1$ or $n = 1$. So suppose $m > 1$ and $n > 1$. For $\gcd(m, n) = 1$, consider the array

$$\begin{array}{cccccc} 0 & 1 & 2 & \cdots & m-1 & \\ m & m+1 & m+2 & \cdots & m+(m-1) & \\ \cdot & \cdot & \cdot & \cdots & \cdot & \\ \cdot & \cdot & \cdot & \cdots & \cdot & \\ (n-1)m & (n-1)m+1 & (n-1)m+2 & \cdots & (n-1)m+(m-1) & \end{array} \quad (6.2.1)$$

On the one hand, (6.2.1) consists of mn consecutive integers, so it is all the numbers modulo mn and therefore contains $\phi(mn)$ elements prime to mn .

On the other hand, observe (6.2.1). The first row is all the numbers modulo m , and all the elements in any column are congruent modulo m . So there are $\phi(m)$ columns consisting entirely of integers prime to m . Let $b, m+b, 2m+b, \dots, (n-1)m+b$ be any such column of n elements. With $\gcd(m, n) = 1$, by Theorem 6.6, such a column is a complete residue system modulo n . So in each such column there are $\phi(n)$ elements prime to n . To this end we know that in (6.2.1) there are $\phi(m)\phi(n)$ elements prime to both m and n . Further notice that any element prime to both m and to n if and only if it is prime to mn .

Combining the results of the above two paragraphs, we have derived $\phi(mn) = \phi(m)\phi(n)$.

iv) For any prime p , in $1, 2, \dots, p^e$, the elements which are not prime to p^e are the multiples of p , i.e., $p, 2p, \dots, p^{e-1}p$. Clearly, there are exactly p^{e-1} such numbers. So

$$\phi(p^e) = p^e - p^{e-1} = p^e \left(1 - \frac{1}{p}\right).$$

This holds for each prime power $p^e | n$ with $p^{e+1} \nmid n$. Noticing that different such prime powers of n are relatively prime to each other, the targeted result follows from (iii). \square

In §4.8 we considered the problem **Square Freeness**: answering whether a given odd composite integer N is square free. There we used $\phi(N)$ to serve an auxiliary input to show that the problem **Square Freeness** is in \mathcal{NP} . Now from Property (vi) of Lemma 6.1 we know that for any prime $p > 1$, if $p^2 | N$ then $p | \phi(N)$. This is why we use $\gcd(N, \phi(N)) = 1$ to confirm N being square free. The reader may consider the case $\gcd(N, \phi(N)) > 1$ (be careful of the case, e.g., $N = PQ$ with $P | \phi(Q)$).

Euler's phi function has the following elegant property.

Theorem 6.9: For integer $n > 0$, $\sum_{d|n} \phi(d) = n$.

Proof Let $S_d = \{x | 1 \leq x \leq n, \gcd(x, n) = d\}$. It is clear that set $S = \{1, 2, \dots, n\}$ is partitioned into disjoint subsets S_d for each $d|n$. Hence

$$\bigcup_{d|n} S_d = S.$$

Notice that for each $d|n$, $\#S_d = \phi(n/d)$, therefore

$$\sum_{d|n} \phi(n/d) = n.$$

However, for any $d|n$, we have $(n/d)|n$, therefore

$$\sum_{d|n} \phi(n/d) = \sum_{(n/d)|n} \phi(n/d) = \sum_{d|n} \phi(d).$$

\square

Example 6.3: For $n = 12$, the possible values of $d|12$ are 1, 2, 3, 4, 6, and 12. We have $\phi(1) + \phi(2) + \phi(3) + \phi(4) + \phi(6) + \phi(12) = 1 + 1 + 2 + 2 + 2 + 4 = 12$. \square

6.3 The Theorems of Fermat, Euler and Lagrange

We have introduced the Fermat's Little Theorem in Chapter 3 (congruence (4.6.5)) and have since used it for a few time but without having proved it. Now we prove the Fermat's Little Theorem by showing that it is a special case of another famous theorem in number theory: the Euler's Theorem.

Theorem 6.10: (Fermat's Little Theorem) If p is prime and $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$.

Since $\phi(p) = p - 1$ for p being prime, Fermat's Little Theorem is a special case of the following theorem.

Theorem 6.11: (Euler's Theorem) *If $\gcd(a, n) = 1$ then $a^{\phi(n)} \equiv 1 \pmod{n}$.*

Proof For $\gcd(a, n) = 1$, we know $a \pmod{n} \in \mathbb{Z}_n^*$. Also $\#\mathbb{Z}_n^* = \phi(n)$. By Corollary 5.2, we have $\text{ord}(a \pmod{n}) = \text{ord}_n(a) \mid \#\mathbb{Z}_n^*$ which implies $a^{\phi(n)} \equiv 1 \pmod{n}$. \square

Since Corollary 5.2 is a direct application of the Lagrange's Theorem (Theorem 5.1), we can now say that the Fermat's Little Theorem and the Euler's Theorem are special cases of the beautiful Theorem of Lagrange.

6.4 Quadratic Residues

Quadratic residues play important roles in number theory. For example, integer factorization algorithms invariantly involve using quadratic residues. They also have frequent uses in encryption and interesting cryptographic protocols.

Definition 6.1: Quadratic Residue *Let integer $n > 1$. For $a \in \mathbb{Z}_n^*$, a is called a quadratic residue modulo n if $x^2 \equiv a \pmod{n}$ for some $x \in \mathbb{Z}_n$; otherwise, a is called a quadratic non-residue modulo n . The set of quadratic residues modulo n is denoted by QR_n , and the set of quadratic non-residues modulo n is denoted by QNR_n .*

Example 6.4: Let us compute QR_{11} , the set of all quadratic residues modulo 11. $\text{QR}_{11} = \{1^2, 2^2, 3^2, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 10^2\} \pmod{11} = \{1, 3, 4, 5, 9\}$. \square

In this example, we have computed QR_{11} by exhaustively squaring elements in \mathbb{Z}_{11}^* . However, this is not necessary. In fact, the reader may check

$$\text{QR}_{11} = \{1^2, 2^2, 3^2, 4^2, 5^2\} \pmod{11},$$

i.e., exhaustively squaring elements up to half the magnitude of the modulus suffices. The following theorem claims so for any prime modulus.

Theorem 6.12: *Let p be a prime number. Then*

- i) $\text{QR}_p = \{x^2 \pmod{p} \mid 0 < x \leq (p-1)/2\}$;
- ii) *There are precisely $(p-1)/2$ quadratic residues and $(p-1)/2$ quadratic non-residues modulo p , that is, \mathbb{Z}_p^* is partitioned into two equal-size subsets QR_p and QNR_p .*

Proof (i) Clearly, set $S = \{ x^2 \pmod{p} \mid 0 < x \leq (p-1)/2 \} \subseteq \text{QR}_p$. To show $\text{QR}_p = S$ we only need to prove $\text{QR}_p \subseteq S$.

Let any $a \in \text{QR}_p$. Then $x^2 \equiv a \pmod{p}$ for some $x < p$. If $x \leq (p-1)/2$ then $a \in S$. Suppose $x > (p-1)/2$. Then $y = p - x \leq (p-1)/2$ and $y^2 \equiv (p-x)^2 \equiv p^2 - 2px + x^2 \equiv x^2 \equiv a \pmod{p}$. So $\text{QR}_p \subseteq S$.

ii) To show $\#\text{QR}_p = (p-1)/2$ it suffices to show that for $0 < x < y \leq (p-1)/2$, $x^2 \not\equiv y^2 \pmod{p}$. Suppose on the contrary, $x^2 - y^2 \equiv (x+y)(x-y) \equiv 0 \pmod{p}$. Then $p \mid x+y$ or $p \mid x-y$. Only the latter case is possible since $x+y < p$. Hence $x = y$, a contradiction.

Then $\#\text{QNR}_p = (p-1)/2$ since $\text{QNR}_p = \mathbb{Z}_p^* \setminus \text{QR}_p$ and $\#\mathbb{Z}_p^* = p-1$. \square

In the proof of Theorem 6.12(i) we have actually shown the following:

Corollary 6.2: *Let p be a prime number. Then for any $a \in \text{QR}_p$, there are exactly two square roots of a modulo p . Denoting by x one of them, then the other is $-x$ ($= p - x$).* \square

6.4.1 Quadratic Residuosity

Often we need to decide if a number is a quadratic residue element modulo a given modulus. This is the so-called **quadratic residuosity problem**.

Theorem 6.13: (Euler's Criterion) *Let p be a prime number. Then for any $x \in \mathbb{Z}_p^*$, $x \in \text{QR}_p$ if and only if*

$$x^{(p-1)/2} \equiv 1 \pmod{p}. \quad (6.4.1)$$

Proof (\Rightarrow) For $x \in \text{QR}_p$, there exists $y \in \mathbb{Z}_p^*$ such that $y^2 \equiv x \pmod{p}$. So $x^{(p-1)/2} \equiv y^{p-1} \equiv 1 \pmod{p}$ follows from Fermat's Theorem (Theorem 6.10).

(\Leftarrow) Let $x^{(p-1)/2} \equiv 1 \pmod{p}$. Then x is a root of polynomial $y^{(p-1)/2} - 1 \equiv 0 \pmod{p}$. Notice that \mathbb{Z}_p is a field, by Theorem 5.9(iii), every element in the field is a root of the polynomial $y^p - y \equiv 0 \pmod{p}$. In other words, every non-zero element of the field, i.e., every element in the group \mathbb{Z}_p^* , is a root of

$$y^{p-1} - 1 \equiv (y^{(p-1)/2} - 1)(y^{(p-1)/2} + 1) \equiv 0 \pmod{p}.$$

These roots are all distinct since this degree- $(p-1)$ polynomial can have at most $p-1$ roots. Consequently, the $(p-1)/2$ roots of polynomial $y^{(p-1)/2} - 1 \equiv 0 \pmod{p}$ must all be distinct. We have shown in Theorem 6.12 that QR_p contains exactly $(p-1)/2$ elements, and they all satisfy $y^{(p-1)/2} - 1 \equiv 0 \pmod{p}$. Any other element in \mathbb{Z}_p^* must satisfy $y^{(p-1)/2} + 1 \equiv 0 \pmod{p}$. Therefore $x \in \text{QR}_p$. \square

In the proof of Theorem 6.13 we have shown that if the criterion is not met for $x \in \mathbb{Z}_p$, then

$$x^{(p-1)/2} \equiv -1 \pmod{p}. \quad (6.4.2)$$

Euler's Criterion provides a criterion to test whether or not an element in \mathbb{Z}_p^* is a quadratic residue: if congruence (6.4.1) is satisfied, then $x \in \text{QR}_p$; otherwise (6.4.2) is satisfied and $x \in \text{QNR}_p$.

Let n be a composite natural number with its prime factorization as

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}. \quad (6.4.3)$$

Then by Theorem 6.8, \mathbb{Z}_n is isomorphic to $\mathbb{Z}_{p_1^{e_1}} \times \mathbb{Z}_{p_2^{e_2}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}}$. Since isomorphism preserves arithmetic, we have:

Theorem 6.14: *Let n be a composite integer with complete factorization in (6.4.3). Then $x \in \text{QR}_n$ if and only if $x \pmod{p_i^{e_i}} \in \text{QR}_{p_i^{e_i}}$ and hence if and only if $x \pmod{p_i} \in \text{QR}_{p_i}$ for prime p_i with $i = 1, 2, \dots, k$. \square*

Therefore, if the factorization of n is known, given $x \in \mathbb{Z}_n^*$, the quadratic residuosity of x modulo n can be decided by deciding the residuosity of $x \pmod{p}$ for each prime $p|n$. The latter task can be done by testing the Euler's criterion.

However, if the factorization of n is unknown, deciding quadratic residuosity modulo n is a non-trivial task.

Definition 6.2: Quadratic Residuosity Problem (QRP)

INPUT n : a composite number;
 $x \in \mathbb{Z}_n^*$;

OUTPUT YES if $x \in \text{QR}_n$.

The QRP is a well-known hard problem in number theory and is one of the main four algorithmic problems discussed by Gauss in his “Disquisitiones Arithmeticae” [Gau01]. An efficient solution for it would imply an efficient solution to some other open problems in number theory. One example is deciding whether a composite integer n is the product of two or three distinct primes (review **indistinguishable experiments** $E_{2\text{-Prime}}$ and $E_{3\text{-Prime}}$ we constructed in §4.10). In Chapter 13 we will study a well-known **public-key cryptosystem** named the **Goldwasser-Micali cryptosystem**; that cryptosystem has its security based on deciding the QRP.

Combining Theorem 6.12 and Theorem 6.14 we can obtain:

Theorem 6.15: *Let n be a composite integer with $k > 1$ distinct prime factors. Then exactly $\frac{1}{2^k}$ fraction of elements in \mathbb{Z}_n^* are quadratic residues modulo n . \square*

Thus, for a composite number n , an efficient algorithm for deciding quadratic residuosity modulo n will provide an efficient statistic for testing the proportion of quadratic residues in \mathbb{Z}_n^* , and hence by Theorem 6.15, provide an efficient algorithm for answering the question whether n has two or three distinct prime factors. By Theorem 6.15, in the former case, exactly a quarter of elements in \mathbb{Z}_n^* are quadratic residues, and in the latter case, exactly one-eighth of them are.

To date, for a composite n of unknown factorization, no algorithm is known to be able to decide quadratic residuosity modulo n in time polynomial in the size of n .

6.4.2 Legendre-Jacobi Symbols

Testing quadratic residuosity using Euler's criterion (6.4.1) involves evaluating modulo exponentiation which is quite computation intensive. However, quadratic residuosity can be tested by a much faster algorithm. Such an algorithm is based on the notion of Legendre-Jacobi symbol.

Definition 6.3: Legendre-Jacobi Symbol *For each prime number p and for any $x \in \mathbb{Z}_p^*$, let*

$$\left(\frac{x}{p}\right) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x \in \text{QR}_p \\ -1 & \text{if } x \in \text{QNR}_p. \end{cases}$$

$\left(\frac{x}{p}\right)$ is called Legendre symbol of x modulo p .

Let $n = p_1 p_2 \cdots p_k$ be the prime factorization of n (some of these prime factors may repeat). Then

$$\left(\frac{x}{n}\right) \stackrel{\text{def}}{=} \left(\frac{x}{p_1}\right) \left(\frac{x}{p_2}\right) \cdots \left(\frac{x}{p_k}\right)$$

is called Jacobi symbol of x modulo n .

In the rest of this book $\left(\frac{a}{b}\right)$ will always be referred to as Jacobi symbol whether or not b is prime.

For p being prime, comparing (6.4.1), (6.4.2) with Definition 6.3, we know

$$\left(\frac{x}{p}\right) = x^{(p-1)/2} \pmod{p}. \quad (6.4.4)$$

Moreover, Jacobi symbol has the following properties.

Theorem 6.16: *Jacobi symbol has the following properties:*

$$i) \quad \left(\frac{1}{n}\right) = 1;$$

$$ii) \left(\frac{xy}{n}\right) = \left(\frac{x}{n}\right) \left(\frac{y}{n}\right);$$

$$iii) \left(\frac{x}{mn}\right) = \left(\frac{x}{m}\right) \left(\frac{x}{n}\right);$$

$$iv) \text{ if } x \equiv y \pmod{n} \text{ then } \left(\frac{x}{n}\right) = \left(\frac{y}{n}\right);$$

(below m, n are odd numbers)

$$v) \left(\frac{-1}{n}\right) = (-1)^{(n-1)/2};$$

$$vi) \left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8};$$

$$vii) \text{ if } \gcd(m, n) = 1 \text{ and } m, n > 2 \text{ then } \left(\frac{m}{n}\right) \left(\frac{n}{m}\right) = (-1)^{(m-1)(n-1)/4}.$$

In Theorem 6.16, (i–iv) are immediate following the definition of Jacobi symbol. A proof for (v–vii) uses no special technique either. However, due to the lengthiness and lack of immediate relevance to the topic of this book, we shall not include a proof but refer the reader to the standard textbooks for number theory (e.g., [Kra86], [LeV77]).

Theorem 6.16(vii) is known as the Gauss' Law of Quadratic Reciprocity. Thanks to this law, it is not hard to see that the evaluation of $\left(\frac{x}{n}\right)$ for $\gcd(x, n) = 1$ has a fashion and hence the same computational complexity of computing the greatest common divisor.

Remark 6.1: When we evaluate Jacobi symbol by applying Theorem 6.16, the evaluation of the right-hand sides of (v–vii) must not be done via exponentiations. Since $\text{ord}(-1) = 2$ (in multiplication), all we need is the parity of these exponents. In the following algorithm we realize the evaluation by testing whether 2 divides these exponents. \square

Algorithm 6.2 provides a recursive specification of the properties of Jacobi symbol listed in Theorem 6.16.

In Algorithm 6.2, each recursive call of the function $\text{Jacobi}(\cdot)$ will cause either the first input value being divided by 2, or the second input value being reduced modulo the first. Therefore there can be at most $\log_2 n$ calls and the first input value is reduced to 1, reaching the terminating condition. So rigorously expressed, because each modulo operation costs $O_B((\log n)^2)$ time, Algorithm 6.2 computes $\left(\frac{x}{n}\right)$ can be computed in $O_B((\log n)^3)$ time.

However we should notice that, in order to present the algorithm with ease of understanding, we have again chosen to sacrifice the efficiency!

Algorithm 6.2: Legendre/Jacobi SymbolINPUT odd integer $n > 2$, integer $x \in \mathbb{Z}_n^*$;OUTPUT $\left(\frac{x}{n}\right)$.Jacobi(x, n)

1. if ($x == 1$) return (1);
2. if ($2 \mid x$)
 - (a) if ($2 \mid (n^2 - 1)/8$) return (Jacobi($x/2, n$));
 - (b) return($-\text{Jacobi}(x/2, n)$);
- (* now x is odd *)
3. if ($2 \mid (x - 1)(n - 1)/4$) return(Jacobi($n \bmod x, x$));
4. return($-\text{Jacobi}(n \bmod x, x)$).

Figure 6.2.

Instead of bounding each modulo operation with $O_B((\log n)^2)$, via a careful realization, *total* modulo operations in steps 3, 4 can be bounded by $O_B((\log n)^2)$. This situation is exactly the same as that for computing greatest common divisor with a carefully designed algorithm: to exploit the fact expressed in (4.5.11). Consequently, for $x \in \mathbb{Z}_n^*$, $\left(\frac{x}{n}\right)$ can be computed in $O_B((\log n)^2)$ time. A careful realization of the counterpart for Algorithm 6.2 can be found in Chapter 1 of [Coh96].

Comparing with the complexity of evaluating the Euler's criterion (5.3.5), which is $O_B((\log p)^3)$ due to modulo exponentiation, testing quadratic residuosity modulo prime p using Algorithm 6.2 is $\log p$ times faster.

Example 6.5: Let us show that $384 \in \text{QNR}_{443}$.

Going through Algorithm 6.2 step by step, we have

$$\begin{aligned}
 \text{Jacobi}(384, 443) &= -\text{Jacobi}(192, 443) \\
 &= \text{Jacobi}(96, 443) \\
 &= -\text{Jacobi}(48, 443) \\
 &= \text{Jacobi}(24, 443) \\
 &= -\text{Jacobi}(12, 443)
 \end{aligned}$$

$$\begin{aligned}
&= \text{Jacobi}(6, 443) \\
&= -\text{Jacobi}(3, 443) \\
&= \text{Jacobi}(2, 3) \\
&= -\text{Jacobi}(1, 3) \\
&= -1.
\end{aligned}$$

Therefore $384 \in \text{QNR}_{443}$. □

Finally, we should point out that the role of Jacobi symbol does not stop at evaluating Legendre symbol: it also allows us to evaluate $\left(\frac{x}{n}\right)$ without need of knowing the factorization of n . In the rest of this book we shall see a wide use of Jacobi symbol (and Algorithm 6.2) in cryptographic protocols.

6.5 Computing Square Roots Modulo an Integer

In Example 6.2 we have had an experience of “computing a square root modulo an integer”. However the “algorithm” used there should not qualify as an algorithm because we were lucky to have managed to map, using the isomorphism in Theorem 6.8, a seemingly difficult task to two trivially easy ones: computing square roots of 1 and 4, which happen to be square numbers in \mathbb{Z} and the “rooting algorithm” is a knowledge for primary school pupils. In general, the isomorphism in Theorem 6.8 will not be so kind to us: for overwhelming cases the image should not be a square number in \mathbb{Z} .

Now we introduce algorithmic methods for computing square roots of a quadratic residue element modulo a positive integer. We start by considering prime modulus. By Corollary 6.2, the two roots of a quadratic residue complements to one another modulo the prime modulus; so it suffices for us to consider computing one square root of a quadratic residue element.

For most of the odd prime numbers, the task is very easy. These cases include primes p such that $p \equiv 3, 5, 7 \pmod{8}$.

Computing Square Roots Modulo a Prime $p \equiv 3, 5, 7 \pmod{8}$

Case $p \equiv 3, 7 \pmod{8}$

In this case, $p + 1$ is divisible by 4. For $a \in \text{QR}_p$, let

$$x \stackrel{\text{def}}{=} a^{(p+1)/4} \pmod{p}.$$

Then because $a^{(p-1)/2} \equiv 1 \pmod{p}$, we have

$$x^2 \equiv a^{(p+1)/2} \equiv a^{(p-1)/2} a \equiv a \pmod{p},$$

So indeed, x is a square root of a modulo p .

Case $p \equiv 5 \pmod{8}$

In this case, $p+3$ is divisible by 8; also because $(p-1)/2$ is even, -1 meets the Euler's criterion as a quadratic residue. For $a \in \text{QR}_p$, let

$$x \stackrel{\text{def}}{=} a^{(p+3)/8} \pmod{p}. \quad (6.5.1)$$

From $a^{(p-1)/2} \equiv 1 \pmod{p}$ we know $a^{(p-1)/4} \equiv \pm 1 \pmod{p}$; this is because in field \mathbb{Z}_p^* , 1 has only two square roots: 1 and -1 . Consequently

$$x^2 \equiv a^{(p+3)/4} \equiv a^{(p-1)/4} a \equiv \pm a \pmod{p}.$$

That is, we have found that x computed in (6.5.1) is a square root of either a or $-a$. If the sign is $+$ we have done. If the sign is $-$, then we have

$$-x^2 \equiv (\sqrt{-1}x)^2 \equiv a \pmod{p}.$$

Therefore

$$x \stackrel{\text{def}}{=} \sqrt{-1} a^{(p+3)/8} \pmod{p} \quad (6.5.2)$$

will be the solution. So the task boils down to computing $\sqrt{-1} \pmod{p}$. Let b be any quadratic non-residue mod p . Then by the Euler's criterion

$$(b^{(p-1)/4})^2 \equiv b^{(p-1)/2} \equiv -1 \pmod{p},$$

so $b^{(p-1)/4} \pmod{p}$ can be used in place of $\sqrt{-1}$. By the way, since

$$p^2 - 1 = (p+1)(p-1) = (8k+6)(8k+4) = 8(4k'+3)(2k''+1),$$

and the right-hand side is 8 times an odd number; so by Theorem 6.16(vi) $2 \in \text{QNR}_p$. That is, for this case of p we can use $2^{(p-1)/4}$ in place of $\sqrt{-1}$. Then, one may check that (6.5.2) becomes

$$2^{(p-1)/4} a^{(p+3)/8} \equiv (4a)^{(p+3)/8} / 2 \pmod{p}. \quad (6.5.3)$$

We can save one modulo exponentiation by using the right-hand-side of (6.5.3).

The time complexity of Algorithm 6.3 is $O_B((\log p)^3)$.

Computing Square Roots Modulo a Prime in General Case

The method described here is due to Shanks (see §1.5.1 of [Coh96]).

For general case of prime p , we can write

$$p-1 = 2^e q$$

Algorithm 6.3: Square Root Modulo $p \equiv 3, 5, 7 \pmod{8}$.

INPUT prime p satisfying $p \equiv 3, 5, 7 \pmod{8}$; integer $a \in \text{QR}_p$;

OUTPUT a square root of a modulo p .

1. if ($p \equiv 3, 7 \pmod{8}$) return($a^{(p+1)/4} \pmod{p}$);
 (* below $p \equiv 5 \pmod{8}$ *)
2. if ($a^{(p-1)/4} \equiv 1 \pmod{p}$) return($a^{(p+3)/8} \pmod{p}$);
3. return($(4a)^{(p+3)/8} / 2$).

Figure 6.3.

with q odd and $e \geq 1$. By Theorem 5.2, cyclic group \mathbb{Z}_p^* has a unique cyclic subgroup G of order 2^e . Clearly, quadratic residues in G have orders as powers of 2 since they divide 2^{e-1} . For $a \in \text{QR}_p$, since

$$a^{(p-1)/2} \equiv (a^q)^{2^{e-1}} \equiv 1 \pmod{p},$$

so $a^q \pmod{p}$ is in G and is of course a quadratic residue. So there exists an even integer k with $0 \leq k < 2^e$ such that

$$a^q g^k \equiv 1 \pmod{p}, \tag{6.5.4}$$

where g is a generator of G . Suppose that we have found the generator g and the even integer k . Then setting

$$x \stackrel{\text{def}}{=} a^{(q+1)/2} g^{k/2},$$

it is easy to check that $x^2 \equiv a \pmod{p}$.

Thus, the task boils down to two sub-tasks: (i) finding a generator g of group G , and (ii) finding the least non-negative even integer k , such that (6.5.4) is satisfied.

Sub-task (i) is rather easy. For any $f \in \text{QNR}_p$, because q is odd, $f^q \in \text{QNR}_p$ and $\text{ord}_p(f^q) = 2^e$; hence f^q is a generator of G . Finding f is rather easy: picking a random element $f \in \mathbb{Z}_p^*$ and testing $\left(\frac{f}{p}\right) = -1$ (using Algorithm 6.2). Since half the elements in \mathbb{Z}_p^* are quadratic non-residues, the probability of finding a correct f in one go is one-half.

Sub-task (ii) is not too difficult either. The search of k from (6.5.4) is fast by utilizing the fact that non-unity quadratic-residue elements in G have orders as

powers of 2. Thus, letting initially

$$b \stackrel{\text{def}}{=} a^q \equiv a^q g^{2^e} \pmod{p}, \quad (6.5.5)$$

then $b \in G$. We can search the least integer m for $0 \leq m < e$ such that

$$b^{2^m} \equiv 1 \pmod{p} \quad (6.5.6)$$

and then modify b into

$$b \leftarrow bg^{2^{e-m}} \equiv a^q g^{2^{e-m}} \pmod{p}. \quad (6.5.7)$$

Notice that b , after the modification in (6.5.7), has its order been reduced from that in (6.5.5) while remaining to be a quadratic residue in G and so the reduced order should remain to be a power of 2. Therefore, the reduction must be in terms of a power of 2, and consequently, repeating (6.5.6) and (6.5.7), m in (6.5.6) will strictly decrease. Upon $m = 0$, (6.5.6) shows $b = 1$, and thereby (6.5.7) becomes (6.5.4) and so k can be found by accumulating 2^m in each loop of repetition. The search will terminate in at most e loops.

It is now straightforward to put our descriptions into the following algorithm.

Algorithm 6.4: Square Root Modulo Prime

INPUT prime p ; integer $a \in \text{QR}_p$;

OUTPUT a square root of a modulo p .

1. (* initialize *)
set $p - 1 = 2^e q$ with q odd; $b \leftarrow a^q \pmod{p}$; $r \leftarrow e$; $k \leftarrow 0$;
2. (* sub-task (i), using Algorithm 6.2 *)
find $f \in \text{QNR}_p$; $g \leftarrow f^q \pmod{p}$;
3. (* sub-task (ii), searching even exponent k *)
while ($b \neq 1$) do
 - 3.1 find the least non-negative integer m such that $b^{2^m} \equiv 1 \pmod{p}$;
 - 3.2 $b \leftarrow bg^{2^{r-m}} \pmod{p}$; $k \leftarrow k + 2^{r-m}$; $r \leftarrow m$;
4. return($a^{(q+1)/2} g^{k/2} \pmod{p}$).

Figure 6.4.

Since $e < \log_2 p$, the time complexity of Algorithm 6.4 is $O_B((\log p)^4)$.

Remark 6.2: For the purpose of better exposition, we have presented Algorithm 6.4 by following our explanation on the working principle of Shanks' algorithm; in particular, we have followed precisely the explanation on Sub-task (ii) for searching the even exponent k . In so doing, our presentation of Shanks' algorithm sacrifices a little bit of efficiency: explicitly finding k , while is unnecessary since $g^{k/2}$ can be obtained as a byproduct in step 3, costs an additional modulo exponentiation in step 4. For the optimized version of Shanks' algorithm, see Algorithm 1.5.1 in [Coh96].

□

Finally we should point out that Algorithm 6.4 contains Algorithm 6.3 as three special cases.

Computing Square Roots Modulo Composite

Thanks to Theorem 6.8, we know that, for $n = pq$ with p, q primes, \mathbb{Z}_n^* is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Since isomorphism preserves the arithmetic relation

$$x^2 \equiv y \pmod{n}$$

holds if and only if it holds modulo both p and q . Therefore, if the factorization of n is given, square rooting modulo n can be computed using the following algorithm.

Algorithm 6.5: Square Roots Modulo Composite $n = pq$

INPUT primes p, q with $n = pq$; integer $y \in \text{QR}_n$;

OUTPUT a square root of y modulo n .

1. $x_p \leftarrow \sqrt{y \pmod{p}}$;
 $x_q \leftarrow \sqrt{y \pmod{q}}$; (* applying Algorithms 6.3 or 6.4 *)
2. return($\bar{1}_p x_p + \bar{1}_q x_q \pmod{n}$).

Figure 6.5.

Clearly, the time complexity of Algorithm 6.5 is $O_B((\log n)^4)$.

By Corollary 6.2, $y \pmod{p}$ has two distinct square roots, which we denote by x_p and $p - x_p$, respectively. So does $y \pmod{q}$, which we denote by x_q and $q - x_q$, respectively. By the isomorphic relationship between \mathbb{Z}_n^* and $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$ (Theorem 6.8), we know that $y \in \text{QR}_n$ has exactly four square roots in \mathbb{Z}_n^* . By Algorithm 6.5, these

four roots are

$$\left. \begin{aligned} x_1 &\equiv \bar{1}_p x_p & + & \bar{1}_q x_q \\ x_2 &\equiv \bar{1}_p x_p & + & \bar{1}_q (q - x_q) \\ x_3 &\equiv \bar{1}_p (p - x_p) & + & \bar{1}_q x_q \\ x_4 &\equiv \bar{1}_p (p - x_p) & + & \bar{1}_q (q - x_q) \end{aligned} \right\} \pmod{n} \quad (6.5.8)$$

For an exercise, we ask: if $n = pqr$ with p, q, r distinct prime numbers, how many square roots for each $y \in \text{QR}_n$?

We now know that if the factorization of n is known, then computing square roots of any given element in QR_n can be done efficiently. Now, what can we say about square rooting modulo n without knowing the factorization of n ? The third part of the following theorem answers this question.

Theorem 6.17: *Let $n = pq$ with p, q being distinct odd primes and let $y \in \text{QR}_n$. Then the four square roots of y constructed in (6.5.8) have the following properties:*

- i) *they are distinct one another;*
- ii) $x_1 + x_4 = x_2 + x_3 = n$;
- iii) $\gcd(x_1 + x_2, n) = \gcd(x_3 + x_4, n) = q$, $\gcd(x_1 + x_3, n) = \gcd(x_2 + x_4, n) = p$.

Proof

- i) Noticing the meaning of $\bar{1}_p$ and $\bar{1}_q$ defined by (6.1.15) and (6.1.16), we have, e.g., $x_1 \pmod{q} = x_q$ and $x_2 \pmod{q} = q - x_q$. Remember, x_q and $q - x_q$ are two distinct square roots of $y \pmod{q}$. So $x_1 \not\equiv x_2 \pmod{q}$ implies $x_1 \not\equiv x_2 \pmod{n}$, i.e., x_1 and x_2 are distinct. Other cases can be shown analogously.

- ii) From (6.5.8) we have

$$x_1 + x_4 = x_2 + x_3 = \bar{1}_p p + \bar{1}_q q.$$

The right-hand side value is congruent to 0 modulo p and modulo q . From these roots' membership in \mathbb{Z}_n^* , we have $0 < x_1 + x_4 = x_2 + x_3 < 2n$. Clearly, n is the only value in the interval $(0, 2n)$ to be congruent to 0 modulo p and q . So $x_1 = n - x_4$ and $x_2 = n - x_3$.

- iii) We only study the case $x_1 + x_2$; other cases are analogous. Observing (6.5.8) we have

$$x_1 + x_2 = 2 \cdot \bar{1}_p x_p + \bar{1}_q q.$$

Therefore $x_1 + x_2 \pmod{p} \equiv 2x_p \not\equiv 0$ and $x_1 + x_2 \equiv 0 \pmod{q}$. Namely, $x_1 + x_2$ is a non-zero multiple of q , but not a multiple of p . This implies $\gcd(x_1 + x_2, n) = q$. \square

Suppose there existing an efficient algorithm A , which, on input (y, n) for $y \in \text{QR}_n$, outputs x such that $x^2 \equiv y \pmod{n}$. Then we can run $A(x^2, n)$ to obtain a square root of x^2 which we denote by x' . By Theorem 6.17(iii), the probability for $1 < \gcd(x + x', n) < n$ is exactly one half. That is, the algorithm A is an efficient algorithm for factoring n .

Combining Algorithm 6.5 and Theorem 6.17(iii), we have

Corollary 6.3: *Let $n = pq$ with p and q being distinct odd primes. Then factoring n is computationally equivalent to computing square root modulo n .* \square

Also from Theorem 6.17(ii) and the fact that n is odd, we have

Corollary 6.4: *Let $n = pq$ with p and q being distinct odd primes. Then for any $y \in \text{QR}_n$, two square roots of y are less than $n/2$, and the other two roots are larger than $n/2$.* \square

6.6 Blum Integers

Blum integers have wide applications in public-key cryptography.

Definition 6.4: Blum Integer *A composite integer n is called a Blum integer if $n = pq$ where p and q are distinct prime numbers satisfying $p \equiv q \equiv 3 \pmod{4}$.*

A Blum integer has many interesting properties. The following are some of them which are very useful in public-key cryptography and cryptographic protocols.

Theorem 6.18: *Let n be a Blum integer. Then the following properties hold for n :*

- i) $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$ (hence $\left(\frac{-1}{n}\right) = 1$);
- ii) For $y \in \mathbb{Z}_n^*$, if $\left(\frac{y}{n}\right) = 1$ then either $y \in \text{QR}_n$ or $-y = n - y \in \text{QR}_n$;
- iii) Any $y \in \text{QR}_n$ has four square roots $u, -u, v, -v$ and they satisfy (w.l.o.g.)
 - a) $\left(\frac{u}{p}\right) = 1, \left(\frac{u}{q}\right) = 1$, i.e., $u \in \text{QR}_n$;
 - b) $\left(\frac{-u}{p}\right) = -1, \left(\frac{-u}{q}\right) = -1$;
 - c) $\left(\frac{v}{p}\right) = -1, \left(\frac{v}{q}\right) = 1$;

$$d) \left(\frac{-v}{p} \right) = 1, \left(\frac{-v}{q} \right) = -1;$$

iv) Function $f(x) = x^2 \pmod{n}$ is a permutation over \mathbb{QR}_n ;

v) For any $y \in \mathbb{QR}_n$, exactly one square root of y with Jacobi symbol 1 is less than $n/2$;

vi) \mathbb{Z}_n^* is partitioned into four equivalence classes: one multiplicative group \mathbb{QR}_n , and three cosets $(-1)\mathbb{QR}_n$, $\xi\mathbb{QR}_n$, $(-\xi)\mathbb{QR}_n$; here ξ is a square root of 1 with Jacobi symbol -1 .

Proof

i) Notice that $p \equiv 3 \pmod{4}$ implies $\frac{p-1}{2} = 2k+1$. Then by Euler's Criterion (6.4.1), we have

$$\left(\frac{-1}{p} \right) = (-1)^{\frac{p-1}{2}} = (-1)^{2k+1} = -1.$$

Analogously, $\left(\frac{-1}{q} \right) = -1$.

ii) $\left(\frac{y}{n} \right) = 1$ implies either $\left(\frac{y}{p} \right) = \left(\frac{y}{q} \right) = 1$, or $\left(\frac{y}{p} \right) = \left(\frac{y}{q} \right) = -1$. For the first case, $y \in \mathbb{QR}_n$ due to the definition of Legendre symbol (Definition 6.3) and Theorem 6.14. For the second case, (i) implies $\left(\frac{-y}{p} \right) = \left(\frac{-y}{q} \right) = 1$. Hence $-y \in \mathbb{QR}_n$.

iii) First of all, by Theorem 6.17(ii), we can indeed denote the four distinct square roots of x by u , $-u (= n-u)$, v and $-v$.

Next, from $u^2 \equiv v^2 \pmod{n}$, we have $(u+v)(u-v) \equiv 0 \pmod{p}$, that is, $u \equiv \pm v \pmod{p}$. Similarly, $u \equiv \pm v \pmod{q}$. However, by Theorem 6.17(i), $u \not\equiv \pm v \pmod{n}$, so only the following two cases are possible:

$$u \equiv v \pmod{p} \text{ and } u \equiv -v \pmod{q},$$

or

$$u \equiv -v \pmod{p} \text{ and } u \equiv v \pmod{q}.$$

These two cases plus (i) imply $\left(\frac{u}{n} \right) = -\left(\frac{v}{n} \right)$.

Thus, if $\left(\frac{u}{n} \right) = 1$ then $\left(\frac{v}{n} \right) = -1$ and if $\left(\frac{u}{n} \right) = -1$ then $\left(\frac{v}{n} \right) = 1$. Without loss of generality, the four distinct Legendre-symbol characterizations in (a)-(d) follow the multiplicative property of Legendre-Jacobi symbol and (i).

- iv) For any $y \in \text{QR}_n$, by (iii) there exists a unique $x \in \text{QR}_n$ satisfying $f(x) = y$. Thus, $f(x)$ is a 1-1 and onto mapping, i.e., a permutation, over QR_n .
- v) By (iii), the square root with Jacobi symbol 1 is either u or $n - u$. Only one of them can be less than $n/2$ since n is odd. (So, exactly one square root with Jacobi symbol -1 is less than $n/2$; the other two roots are larger than $n/2$ and have the opposite Jacobi symbols.)
- vi) It is trivial to check that QR_n forms a group under multiplication modulo n with 1 as the identity. Now by (iii), the four distinct square roots of 1 have the four distinct Legendre-symbol characterizations in (a), (b), (c), and (d), respectively. Therefore the four sets QR_n , $(-1)\text{QR}_n$, ξQR_n , $(-\xi)\text{QR}_n$ are pair-wise disjoint. These four sets make up \mathbb{Z}_n^* because by Theorem 6.15, $\#\text{QR}_n = \frac{\#\mathbb{Z}_n^*}{4}$. □

6.7 Chapter Summary

In this chapter we have conducted a study in the following topics of elementary number theory:

- linear congruences;
- Chinese remainder theorem;
- Lagrange's, Euler's and Fermat's theorems;
- quadratic residues and Legendre-Jacobi symbols;
- square roots modulo integers;
- Blum integers and their properties.

In addition to introduction to knowledge and facts, we construct several important algorithms (Chinese remainder, Jacobi symbol, square-rooting), explain their working principles and analyze their time complexities. In so doing, we consider that these algorithms not only have theoretic importance, but also have practical importance: they are frequently used algorithms in cryptography and cryptographic protocols.

In the rest of this book we will frequently apply knowledge, facts and skills which we have learnt in this chapter.

Part III

BASIC CRYPTOGRAPHIC TECHNIQUES

This part contains three chapters which introduce the most basic cryptographic techniques for confidentiality and data integrity. Chapter 7 introduces symmetric encryption techniques, Chapter 8 introduces asymmetric encryption techniques, Chapter 9 introduces basic techniques for data integrity.

The basic cryptographic algorithms and schemes introduced in this part, especially the encryption algorithms, can be considered as “textbook versions” since they can be found in many textbooks on cryptography. In these chapters we shall frequently expose the weakness of these “textbook version” algorithms and schemes, even though we will not, in fact cannot, fix these weakness in this part. However, this book will not stop at the “textbook version” level of introduction to cryptography. “Fit for application versions” of the encryption algorithms and data-integrity mechanisms will be introduced in later chapters, and most of them are results of enhancing the “textbook versions”.

ENCRYPTION — SYMMETRIC TECHNIQUES

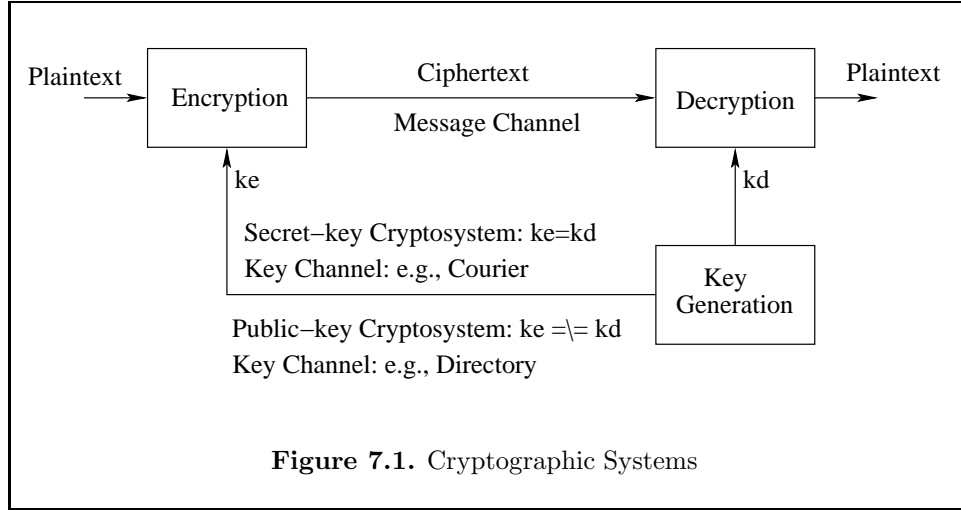
7.1 Introduction

Secrecy is at the heart of cryptography. Encryption is a practical means to achieve information secrecy. Modern encryption techniques are mathematical transformations (algorithms) which transform meaningful messages into unintelligible ones. In order to restore information, such a transformation must be reversible and the reversing transformation is called decryption. Conventionally, encryption and decryption algorithms are parameterized by cryptographic keys. Informally speaking, an encryption algorithm and a decryption algorithm plus the description on the format of messages and keys form a cryptographic system or a cryptosystem. Figure 7.1 provides an illustration of cryptosystems.

Syntactically, a cryptosystem can be defined follows.

Definition 7.1: (Cryptographic System) *A cryptographic system consists of the following:*

- a plaintext message space \mathcal{M} : a set of strings over some alphabet
- a ciphertext message space \mathcal{C} : a set of possible ciphertext messages
- an encryption key space \mathcal{K} : a set of possible encryption keys, and a decryption key space \mathcal{K}' : a set of possible decryption keys
- an efficient key generation algorithm $\mathcal{G} : \mathbb{N} \mapsto \mathcal{K} \times \mathcal{K}'$
- an efficient encryption algorithm $\mathcal{E} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{C}$
- an efficient decryption algorithm $\mathcal{D} : \mathcal{C} \times \mathcal{K}' \mapsto \mathcal{M}$.



For $ke \in \mathcal{K}$ and $m \in \mathcal{M}$, we denote by

$$c = \mathcal{E}_{ke}(m)$$

the encryption transformation and read it as “ c is an encryption of m under key ke ”, and we denote by

$$m = \mathcal{D}_{kd}(c)$$

the decryption transformation and read it as “ m is the decryption of c under key kd which matches the encryption key ke ”. It is necessary that for all $m \in \mathcal{M}$ and all $ke \in \mathcal{K}$, there exists $kd \in \mathcal{K}'$:

$$\mathcal{D}_{kd}(\mathcal{E}_{ke}(m)) = m. \quad (7.1.1)$$

Definition 7.1 applies to cryptosystems which use secret keys as well as public keys (public-key cryptosystems will be introduced in the next chapter). In a secret-key cryptosystem, encryption and decryption use the same key, that is, for every key $ke \in \mathcal{K}$, $kd = ke = k$. The principal who encrypts a message must share the encryption key with the principal who will be decrypting the ciphertext. For this reason, secret-key cryptosystems are also called **symmetric cryptosystems**, or **shared-key cryptosystems**.

Notice that the integer input to the key generation algorithm G provides the size of the output encryption/decryption keys. The integer should be unary encoded (see Definition 4.7).

Semantically, Shannon characterized a desired property or requirement for a cryptosystem as follows: the ciphertext message space \mathcal{C} is the space of all messages while the plaintext message space \mathcal{M} is a region inside \mathcal{C} , in which messages have

a certain fairly simple statistical structure, i.e., they are meaningful; a (good) encryption algorithm E is a mixing-transformation which distributes the meaningful messages from the region \mathcal{M} fairly uniformly over the entire message space \mathcal{C} (pages 711-712 of [Sha49]). Shannon also illustrated this mixing property by

$$\lim_{n \rightarrow \infty} \bigcup_n \mathcal{E}_{ke}^n(\mathcal{M}) = \mathcal{C}. \quad (7.1.2)$$

Although nowadays, in particular after the invention of public-key cryptography, the case $\mathcal{M} \subseteq \mathcal{C}$ is no longer necessarily true (this is still true for most cryptosystems, secret-key or public-key), Shannon's semantical characterization for encryption as a mixing-transformation remains to be very meaningful. A contemporary definition for **semantic security** of an encryption algorithm will be defined in §13.3.

In 1883, Kerchoffs wrote a list of requirements for the design cryptosystems ([MvOV97] page 14). One of the items in Kerchoffs' list has evolved into a widely-accepted principle known as the **Kerchoffs' principle**:

Knowledge of the algorithm and key size as well as the availability of known plaintext, are standard assumptions in modern cryptanalysis. Since an adversary may obtain this information eventually, it is preferable not to rely on its secrecy when assessing cryptographic strength.

Combining Shannon's characterization on cryptosystem and the Kerchoffs' principle, we can provide a summary for a good cryptosystem as follows:

- Algorithms \mathcal{E} and \mathcal{D} contain no component or design part which is secret;
- \mathcal{E} distributes meaningful messages fairly uniformly over the entire ciphertext message space; it may even be possible that the random distribution is due to some internal random operation of \mathcal{E} ;
- With the correct cryptographic key, \mathcal{E} and \mathcal{D} are *practically* efficient;
- Without the correct key, the task for recovering from a ciphertext the correspondent plaintext is a problem with the difficulty determined solely by the size of the key parameter, which usually takes a size such that solving the problem requires a polynomially unbounded power of computation.

7.1.1 Chapter Outline

In this chapter we will introduce several well-known cryptosystems and encryption methods. We begin with introducing several classical ciphers (§7.2, §7.3). We will make explicit the importance of the classical cipher techniques by showing their widespread roles in modern ciphers and in cryptographic protocols (§7.4). After classical ciphers, two important modern block ciphers will be described: the Data

Encryption Standard (the DES, §7.5) and the Advanced Encryption Standard (the AES, §7.6), and their design strategies will be explained. We will also provide a brief discussion on the AES' positive impact on the applied cryptography (§7.6.5). The part for symmetric techniques will also include various standard modes of operations for using block ciphers which achieve the probabilistic encryption (7.7). We end our introduction to symmetric encryption techniques by posing the classical problem of key channel establishment (§7.8).

7.2 Substitution Ciphers

In a **substitution ciphers**, the encryption algorithm $\mathcal{E}_k(m)$ is a substitution function which replaces each $m \in \mathcal{M}$ with a corresponding $c \in \mathcal{C}$. The substitution function is parameterized by a secret key k . The decryption algorithm $\mathcal{D}_k(c)$ is merely the reverse substitution. In general, the substitution can be given by a mapping $\pi : \mathcal{M} \mapsto \mathcal{C}$, and the reverse substitution is just the corresponding inverse mapping $\pi^{-1} : \mathcal{C} \mapsto \mathcal{M}$.

7.2.1 Simple Substitution Ciphers

Example 7.1: (A Simple Substitution Cipher) Let $\mathcal{M} = \mathcal{C} = \mathbb{Z}_{26}$ and interpret $A = 0, B = 1, \dots, Z = 25$. Define encryption algorithm $\mathcal{E}_k(m)$ as the following permutation over \mathbb{Z}_{26}

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 21 & 12 & 25 & 17 & 24 & 23 & 19 & 15 & 22 & 13 & 18 & 3 & 9 \end{pmatrix} \\ \begin{pmatrix} 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 5 & 10 & 2 & 8 & 16 & 11 & 14 & 7 & 1 & 4 & 20 & 0 & 6 \end{pmatrix}.$$

Then the corresponding decryption algorithm $\mathcal{D}_K(C)$ is given by

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 24 & 21 & 15 & 11 & 22 & 13 & 25 & 20 & 16 & 12 & 14 & 18 & 1 \end{pmatrix} \\ \begin{pmatrix} 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 9 & 19 & 7 & 17 & 3 & 10 & 6 & 23 & 0 & 8 & 5 & 4 & 2 \end{pmatrix}.$$

Plaintext messages

proceed meeting as agreed

will be encrypted into the following ciphertext messages (spaces are not transformed)

cqkzyyr jyyowft vl vtqyyr

□

In this simple substitution cipher example, the message spaces \mathcal{M} and \mathcal{C} coincide with the alphabet \mathbb{Z}_{26} . In other words, a plaintext or ciphertext message is a single character in the alphabet. For this reason, the plaintext message string **proceedmeetingasagreed** is not a single message, but comprises of 22 messages; likewise, the ciphertext message string **cqkzyyrjyyowftvlvtqyyr** comprises of 22 messages. The key space of this cipher has the size $26! > 4 \times 10^{26}$, which is huge in comparison with the size of the message space. However, this cipher is in fact very weak: each plaintext character is encrypted to a unique ciphertext character. This weakness renders this cipher extremely vulnerable to a **cryptanalysis** technique called **frequency analysis** which exploits the fact that natural languages contain high volume of redundancy (see §3.7). We will further discuss the security of simple substitution cipher in §7.4.

Several special cases of simple substitution ciphers appeared in the history. The simplest and the most well-known case is called **Shift Ciphers**. In shift ciphers, $\mathcal{K} = \mathcal{M} = \mathcal{C}$; let $N = \#\mathcal{M}$, the encryption and decryption mappings are defined by

$$\begin{cases} \mathcal{E}_k(m) \stackrel{\text{def}}{=} m + k \pmod{N} \\ \mathcal{D}_k(c) \stackrel{\text{def}}{=} c - k \pmod{N} \end{cases} \quad (7.2.1)$$

with $m, c, k \in \mathbb{Z}_N$. For the case of \mathcal{M} being the capital letters of the Latin alphabet, i.e., $\mathcal{M} = \mathbb{Z}_{26}$, the shift cipher is also known as **Caesar cipher**, because Julius Caesar used it with the case of $k = 3$ (§2.2 of [Den82]).

By Theorem 6.6 we know that if $\gcd(k, N) = 1$, then for every $m < N$:

$$km \pmod{N}$$

forms a permutation over the message space. Therefore for such k and for $m, c < N$

$$\begin{cases} \mathcal{E}_k(m) \stackrel{\text{def}}{=} km \pmod{N} \\ \mathcal{D}_k(c) \stackrel{\text{def}}{=} k^{-1}c \pmod{N} \end{cases} \quad (7.2.2)$$

provides a simple substitution cipher. Similarly,

$$k_1m + k_2 \pmod{N}$$

can also define a simple substitution cipher the (called **affine cipher**):

$$\begin{cases} \mathcal{E}_k(m) \stackrel{\text{def}}{=} k_1m + k_2 \pmod{N} \\ \mathcal{D}_k(c) \stackrel{\text{def}}{=} k_1^{-1}(c - k_2) \pmod{N} \end{cases} \quad (7.2.3)$$

It is not difficult to see that using various arithmetic operations between keys in \mathcal{K} and messages in \mathcal{M} , various cases of simple substitution ciphers can be designed.

These ciphers are called **monoalphabetic ciphers**: for a given encryption key, each plaintext message will be substituted into a unique ciphertext message. As we have mentioned earlier, these simple substitution ciphers are extremely vulnerable to frequency analysis attacks.

However, due to their simplicity, simple substitution ciphers have been widely used in modern secret-key encryption algorithms. We will see the kernel role that simple substitution ciphers play in the Data Encryption Standard (DES) (§7.5) and in the Advanced Encryption Standard (AES) (§7.6). Simple substitution ciphers are also widely used in cryptographic protocols; we will illustrate a protocol's application of a simple substitution cipher in §7.4 and see many further such examples in the rest of the book.

7.2.2 Polyalphabetic Ciphers

A substitution cipher is called a **polyalphabetic cipher** if a plaintext message in \mathcal{P} may be substituted into many, possibly any, ciphertext message in \mathcal{C} .

We shall use the **Vigenère cipher** to exemplify a polyalphabetic cipher since the Vigenère cipher is the best known among polyalphabetic ciphers.

The Vigenère cipher is a string-based substitution cipher: a key is a string comprises of a plural number of characters. Let m be the key length. Then a plaintext string is divided into sections of m characters, that is, each section is a string of m characters with possibly an exception that the final section of string may have fewer characters. The encryption algorithm operates that of the shift cipher between the key string and a plaintext string, each plaintext string at a time with the key string being repeated. The decryption follows the same manner, using the decryption operation of the shift cipher.

Example 7.2: (Vigenère Cipher) Let the key string be **gold**. Using the encoding rule $A = 0, B = 1, \dots, Z = 25$, the numerical representation of this key string is (6, 14, 11, 3). The Vigenère encryption of the plaintext string

proceed meeting as agreed

has the following operation which is character-wise addition modulo 26:

15	17	14	2	4	4	3	12	4	4	19
6	14	11	3	6	14	11	3	6	14	11
<hr/>										
21	5	25	5	10	18	14	15	10	18	4
8	13	6	0	18	0	6	17	4	4	3
3	6	14	11	3	6	14	11	3	6	14
<hr/>										
11	19	20	11	21	6	20	2	7	10	17

Thus, the ciphertext string is

vfzfkso pkseltu lv guchkr

□

Other well-known polyalphabetic ciphers are the **book cipher** (also called the **Beale cipher**) where the key string is an agreed text in a book and the **Hill cipher**. For these ciphers, see, e.g., §2.2 of [Den82] or §1.1 of [Sti95], for the description of these substitution ciphers.

We should point out that the substitution ciphers cover the so-called **transposition ciphers** (also called **permutation ciphers**) as the special case. In the latter ciphers, encryption is via moving the position of characters in the plaintext message string. For example, let m be some fixed positive integer, let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$, and let \mathcal{K} be all permutations of $(1, 2, \dots, m)$. Then for a key $\pi = (\pi(1), \pi(2), \dots, \pi(m)) \in \mathcal{K}$ and for plaintext string $(x_1, x_2, \dots, x_m) \in \mathcal{P}$, the encryption algorithm of a transposition cipher is

$$e_\pi(x_1, x_2, \dots, x_m) = (x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(m)}).$$

Let π^{-1} denote the inverse of π , i.e., $\pi^{-1}(\pi(i)) = i$ for $i = 1, 2, \dots, m$. Then the corresponding decryption algorithm is

$$d_\pi(y_1, y_2, \dots, y_m) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, \dots, y_{\pi^{-1}(m)}).$$

7.3 The Vernam Cipher and the One-Time Pad

The **Vernam cipher** is one of the simplest cryptosystems. If we assume that the message is a string of n binary bits

$$m = b_1 b_2 \dots b_n \in \{0, 1\}^n$$

then the key is also a string of n binary bits

$$k = k_1 k_2 \dots k_n \in_U \{0, 1\}^n,$$

(notice here the symbol “ \in_U ”: k is chosen at uniformly random). Encryption takes place one bit at a time and the ciphertext string $c = c_1 c_2 \dots c_n$ is found by the bit operation XOR (exclusive or) each message bit with the corresponding key bit

$$c_i = b_i \oplus k_i$$

for $1 \leq i \leq n$, where the operation \oplus is defined by

\oplus	0	1
0	0	1
1	1	0

Decryption is the same as encryption, since \oplus is addition modulo 2, and thereby subtraction is identical to addition.

Considering $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^*$, the Vernam cipher can be viewed as a substitution cipher. Under such a view, if the key string is used for one time only, then the Vernam cipher satisfies the two conditions for secure substitution ciphers that we have summarized in §7.4. Therefore, the confidentiality service provided by the one-time-key Vernam cipher is also secure in the information-theoretic sense. Another simple way to prove the security is the following: a ciphertext message string c provides (an interceptor) no information whatsoever about the plaintext message string m since any m could have yield c if the key k is equal to $c \oplus m$ (bit by bit).

The one-time-key Vernam cipher is also called the **one-time pad cipher**. In principle, as long as the usage of encryption key satisfies the two conditions for secure substitution ciphers which we have listed in §7.4, then any substitution cipher is a one-time pad cipher. Conventionally however, only the cipher using the bitwise XOR operation is called the one-time pad cipher.

In comparison with other substitution ciphers (e.g., the shift cipher using addition modulo 26), the bitwise XOR operation (which is addition modulo 2) can easily be realized by an electronic circuit. Maybe because of this reason, the bitwise XOR operation is a widely used ingredient in the design of modern secret-key encryption algorithms (see the next two sections).

The one-time pad style of encryption is also widely used in cryptographic protocols.

7.4 Classical Ciphers: Usefulness and Security

Consider character-based substitution ciphers. Because the plaintext message space coincides with the alphabet, each message is a character in the alphabet and encryption is to substitute character-by-character each plaintext character with a ciphertext character and the substitution is according to a secret key. If a key is fixed for encrypting a long string of characters, then the same character in the plaintext messages will be encrypted to a fixed character in the ciphertext messages.

It is well known that letters in a natural language have stable frequencies (review §3.7). The knowledge of the frequency distribution of letters in a natural language provides clue for cryptanalysis, a technique aiming at finding information about the plaintext or the encryption key from given ciphertext messages. This phenomenon is shown in Example 7.1, where the ciphertext messages show a high frequent appearance of the letter y, and suggest that a fixed letter must appear in the corresponding plaintext messages with the same frequency (in fact the letter is e which appears in English with a high frequency). Simple substitution ciphers are not secure for hiding natural-language-based information. For details of the crypt-

analysis technique based on studying the frequencies of letters, see any standard texts in cryptography, e.g., §2.2 of [Den82], or §7.3.5 of [MvOV97].)

Polyalphabetic ciphers are stronger than simple substitution ciphers. However, if the key is short and the message is long, then various cryptanalysis techniques can be used to break such ciphers.

However, classical ciphers, even simple substitution ciphers can be secure in a *very strong sense* if the use of cryptographic keys follows certain conditions. In fact, with the proper key usages, simple substitution ciphers are widely used in cryptographic systems and protocols.

Usefulness

Let us now look at an example of the shift cipher (i.e., the simplest substitution cipher) being securely used in a cryptographic protocol. After showing the example, we will summarize two important conditions for secure use of classical ciphers.

Suppose we have a function $C(x) : \mathbb{Z}_N \mapsto \mathbb{Z}_N$ with the following two properties:

One-way: given any $x \in \mathbb{Z}_N$, evaluation of $C(x)$ can be done efficiently (review §4.7 for the meaning of efficient computation) while for almost all $y \in \mathbb{Z}_N$ and for any efficient algorithms A , $\text{Prob}[x \leftarrow A(y) \wedge C(x) = y]$ is a negligible quantity (review 4.9 for the meaning of negligible quantity);

Homomorphic: for all $x_1, x_2 \in \mathbb{Z}_N$, $C(x_1 + x_2) = C(x_1)C(x_2)$.

There are many functions which apparently satisfy these two properties; we shall see some of them later in this chapter.

Using this function, we can construct a so-called “zero-knowledge proof of knowledge” protocol, which allows a **prover** (let it be Alice) can show a **verifier** (let it be Bob) that she knows the pre-image of $C(x)$ without disclosing to the latter the pre-image. This can be achieved by the following simple protocol which uses the shift cipher.

We now provide an intuitive analysis on the properties of Protocol 7.1.

First, it is easy to see that if Alice knows the pre-image of $C(x)$ then the protocol has described a method for her to correctly respond Bob’s challenge.

Bob’s checking step (Step 4) depends on his random choice of b which takes place after Alice has sent $C(k)$. The consistency of Bob’s checking shows that, if Alice does not know the pre-image of $C(x)$ then she can only have $1/2$ odds to have responded Bob a correct answer. If the iteration of m checkings results in no rejection, the probability for Alice’s successful cheating will be $1/2^m$. Bob will be sufficiently confident that Alice’s proof is valid if $1/2^m$ is sufficiently small.

Protocol 7.1: A Zero-knowledge Proof Protocol Using Shift Cipher

COMMON INPUT positive integer N ;
 a one-way and homomorphic function $C()$ over \mathbb{Z}_N ;
 value $C = C(x)$;
 Alice's INPUT $x < N$; (* prover's private input *)
 OUTPUT to Bob Alice knows $x \in \mathbb{Z}_N$ such that $C = C(x)$.

Repeat the following steps m times:

1. Alice picks $k \in_U \mathbb{Z}_N$ and sends to Bob: $C(k)$;
2. Bob picks $b \in_U \{0, 1\}$ and sends to Alice: b ;
3. Alice sends to Bob: $w = bx + k \pmod{N}$; (* shift cipher when $b = 1$ *)
4. Bob checks $C(w) \stackrel{?}{=} \begin{cases} C(k) & \text{if } b = 0 \\ C(x)C(k) & \text{if } b = 1 \end{cases}$
 he rejects if the checking shows error.

Figure 7.2.**Security**

Let us now argue the quality of the confidentiality service that the shift-cipher encryption offers in Protocol 7.1. We claim that the confidentiality service is *perfect*, that is, after running this protocol Bob gets absolutely *no* new information about x beyond what he may have already obtained from the common input $C(x)$ (the common input only provides *a-priori* information).

We should notice that the shift cipher encryption $w = x + k \pmod{N}$ forms a permutation over \mathbb{Z}_N . With $k \in_U \mathbb{Z}_N = \mathcal{K} = \mathcal{M}$, the permutation renders $w \in_U \mathbb{Z}_N$ since a permutation maps the uniform distribution to the uniform distribution. This means that for a given ciphertext w , *any* key in \mathbb{Z}_N could have been used with the same probability in the creation of w . This is equivalent to say that *any* $x \in \mathbb{Z}_N$ is equally likely to have been encrypted inside w . So the plaintext x is independent from the ciphertext w , or the ciphertext leaks no information whatsoever about the plaintext.

If a cipher achieves independence between the distributions of its plaintext and ciphertext, then we say that the cipher is secure in an **information-theoretic** sense. In contrast to a security in the complexity-theoretic sense which we have established in Chapter 4, the security in the information-theoretic sense is *unconditional* and is immune to any method of cryptanalysis. In Protocol 7.1, this sense of

security means that runs of the protocol will not provide Bob with any knowledge regarding Alice's private input x , except the conviction that Alice has the correct private input.

The notion of information-theoretic-based cryptographic security is developed by Shannon [Sha49]. According to Shannon's theory, we can summarize two conditions for secure use of classical ciphers:

Conditions for Secure Use of Classical Ciphers

- i) $\#\mathcal{K} \geq \#\mathcal{M}$;
- ii) $k \in_U \mathcal{K}$ and is used once in each encryption only.

So if a classical cipher (whether it is a simple substitution cipher in character-based or string-based, a polyalphabetic cipher, or the Vernam cipher) encrypts a message string of length ℓ , then in order for the encryption to be secure, the length of a key string should be at least ℓ , and the key string should be used once only. While this requirement may not be very practical for applications which involve encryption of bulk volumes of messages, it is certainly practical for encrypting small data, such as a nonce (see §2.5.4) or a session key (see §2.4). Protocol 7.1 is just such an example.

In the rest of this book we will meet numerous cryptographic systems and protocols which apply various forms of substitution ciphers such as shift ciphers (as in Protocol 7.1), multiplication ciphers (defined in (7.2.2)), affine ciphers (defined in (7.2.3)), and substitution ciphers under the general form of permutations (as in Example 7.1). Most of such applications follow the two conditions for secure use of classical ciphers.

We should finally point out that the basic working principle of the classical ciphers: substitution, is still the most important kernel technique in the construction of modern symmetric encryption algorithms. We will clearly see such usages in the following two sections where we introduce two important modern encryption schemes.

7.5 The Data Encryption Standard (DES)

Without doubt the first and the most significant modern symmetric encryption algorithm is that contained in the Data Encryption Standard (DES) [NBS77]. The DES was published by the United States' National Bureau of Standards in January 1977 as an algorithm to be used for unclassified data (information not concerned with national security). The algorithm has been in wide international use, a primary example being its employment by banks for funds transfer security. Originally approved for a five year period, the standard stood the test of time and was subsequently approved for three further five year periods.

7.5.1 A Description of the DES

The DES is a **block cipher** in which messages are divided into data blocks of a fixed length and each block is treated as one message either in \mathcal{M} or in \mathcal{C} . In the DES, we have $\mathcal{M} = \mathcal{C} = \{0, 1\}^{64}$ and $\mathcal{K} = \{0, 1\}^{56}$; namely, the DES encryption and decryption algorithms take as input a 64-bit plaintext or ciphertext message and a 56-bit key, and output a 64-bit ciphertext or plaintext message.

The operation of the DES can be described in the following three steps:

1. Apply a fixed “initial permutation” IP to the input block. We can write this initial permutation as

$$(L_0, R_0) \leftarrow \text{IP}(\text{Input Block}). \quad (7.5.1)$$

Here L_0 and R_0 are called “(left, right)-half blocks”, each is a 32-bit block. Notice that IP is a fixed function (i.e., is not parameterized by the input key) and is publicly known; therefore this initial permutation has no apparent cryptographic significance.

2. Iterate the following 16 **rounds** of operations (for $i = 1, 2, \dots, 16$)

$$L_i \leftarrow R_{i-1} \quad (7.5.2)$$

$$R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, k_i), \quad (7.5.3)$$

Here k_i is called “round key” which is a 48-bit substring of the 56-bit input key; f is called “S-box Function” (“S” for substitution; we will provide a brief description on this function in §7.5.2). This operation features swapping two half blocks, that is, the left half block input to a round is the right half block output from the previous round. This aims to achieve a big degree of “message diffusion”, essentially the mixing property modelled by Shannon in (7.1.2).

3. The result from round 16, (L_{16}, R_{16}) , is input to the inverse of IP to cancel the effect of the initial permutation. The output from this step is the output of the DES algorithm. We can write this final step as

$$\text{Output Block} \leftarrow \text{IP}^{-1}(R_{16}, L_{16}). \quad (7.5.4)$$

Please pay a particular attention to the input to IP^{-1} : the two half blocks output from round 16 take an additional swap before being input to IP^{-1} .

These three steps are shared by the encryption and the decryption algorithms, with the only difference in that, if the round keys used by one algorithm are k_1, k_2, \dots, k_{16} , then those used by the other algorithm should be $k_{16}, k_{15}, \dots, k_1$. This way of arranging round keys is called “key schedule”, and can be denoted by

$$(k'_1, k'_2, \dots, k'_{16}) = (k_{16}, k_{15}, \dots, k_1). \quad (7.5.5)$$

Example 7.3: Let a plaintext message m have been encrypted to a ciphertext message c under an encryption key k . Let us go through the DES algorithm to confirm the proper working of the decryption function, i.e., decryption of c under k will output m .

The decryption algorithm starts by inputting the ciphertext c as “Input Block”. By (7.5.1) we have

$$(L'_0, R'_0) \leftarrow \text{IP}(c).$$

But since c is actually “Output Block” from the final step of the encryption algorithm, by (7.5.4) we have

$$(L'_0, R'_0) = (R_{16}, L_{16}). \quad (7.5.6)$$

In round 1, following (7.5.2), (7.5.3) and (7.5.6), we have

$$\begin{aligned} L'_1 &\leftarrow R'_0 = L_{16}, \\ R'_1 &\leftarrow L'_0 \oplus f(R'_0, k'_1) = R_{16} \oplus f(L_{16}, k'_1). \end{aligned}$$

In the right-hand sides of these two assignments, L_{16} should be replaced with R_{15} due to (7.5.2), R_{16} should be replaced with $L_{15} \oplus f(R_{15}, k_{16})$ due to (7.5.3), and $k'_1 = k_{16}$ due to the key schedule (7.5.5). Thus, the above two assignments are in fact the following two:

$$\begin{aligned} L'_1 &\leftarrow R_{15}, \\ R'_1 &\leftarrow [L_{15} \oplus f(R_{15}, k_{16})] \oplus f(R_{15}, k_{16}) = L_{15}. \end{aligned}$$

So, after round 1 of decryption, we obtain

$$(L'_1, R'_1) = (R_{15}, L_{15}).$$

Therefore, at the beginning of round 2, the two half blocks are (R_{15}, L_{15}) .

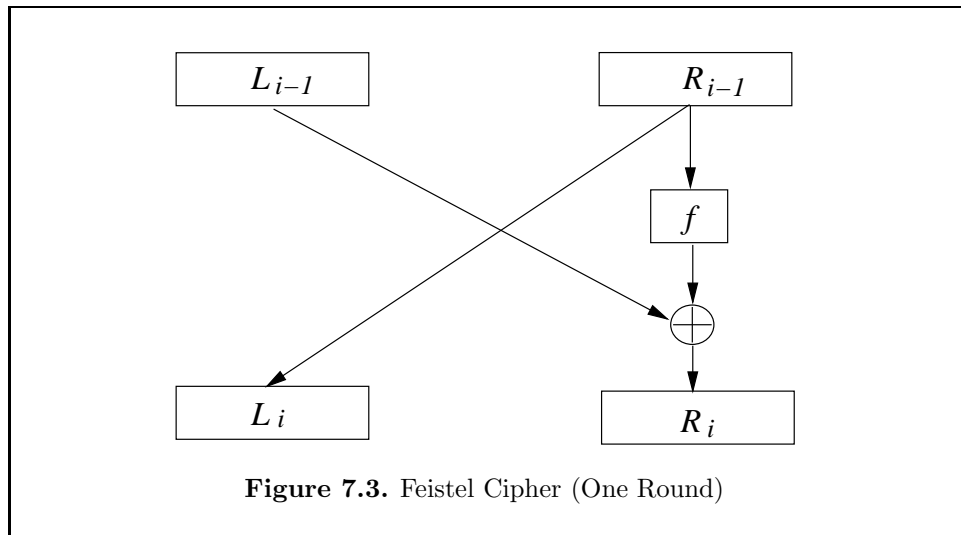
It is routine to check that, in the subsequent 15 rounds, we will obtain

$$(L'_2, R'_2) = (R_{14}, L_{14}), \dots, (L'_{16}, R'_{16}) = (R_0, L_0).$$

The two final half blocks from round 16, (L'_{16}, R'_{16}) are swapped to $(R'_{16}, L'_{16}) = (L_0, R_0)$ and are input to IP^{-1} (notice (7.5.4) for the additional swapping) to cancel the effect of the IP in (7.5.1). Indeed, the output from the decryption function is the original plaintext block m . \square

We have shown that the DES encryption and decryption algorithms do keep equation (7.1.1) to hold for all $m \in \mathcal{M}$ and all $k \in \mathcal{K}$. It is clear that these algorithms work with no regard of the internal details of the “S-box Function” and the key schedule function.

The DES iterations which use (7.5.2) and (7.5.3) to process two half blocks in a swapping fashion is called the **Feistel cipher**. Figure 7.3 illustrates the swapping structure of one round Feistel cipher. Feistel proposed this cipher originally [Fei74]. As we have mentioned earlier, the swapping feature aims to achieve a big degree of data diffusion. Feistel cipher also has an important application in public-key cryptography: a structure named **Optimal Asymmetric Encryption Padding (OAEP)** is essentially a two-round Feistel cipher. We will study OAEP in §14.2.



7.5.2 The Kernel Functionality of the DES: Random and Non-linear Distribution of Message

The kernel part of the DES is inside the “S-box Function” f . This is where the DES realizes a random and non-linear distribution of plaintext messages over the ciphertext message space.

In the i -th round, $f(R_{i-1}, k_i)$ does the following two sub-operations:

- i) add the round key k_i , via bitwise XOR, to the half block R_{i-1} ; this provides the randomness needed in message distribution;
- ii) substitute the result of (i) under a fixed permutation which consists of eight “substitution boxes” (S-boxes), each S-box is a non-linear permutation function; this provides the non-linearity needed in message distribution.

The non-linearity of the S-boxes is very important to the security of the DES. We notice that the general case of the substitution cipher (e.g., Example 7.1 with

random key) is non-linear while the shift cipher and the affine cipher are linear sub-cases. These linear sub-cases not only drastically reduce the size of the key space from that of the general case, but also render the resultant ciphertext vulnerable to a **differential cryptanalysis** (DC) technique [BS91]. DC attacks a cipher by exploiting the linear difference between two plaintext messages and that between two ciphertext messages. Let us look at such an attack using the affine cipher (7.2.3) for example. Suppose an attacker somehow knows the difference $m - m'$ but he does not know m nor m' . Given the corresponding ciphertexts $c = k_1m + k_2 \pmod{N}$, $c' = k_1m' + k_2 \pmod{N}$, the attacker can calculate

$$k_1 = (c - c') / (m - m') \pmod{N}.$$

With k_1 , it becomes much easier for the attacker to further find k_2 , e.g., k_2 can be found if the attacker has a known plaintext-ciphertext pair. Subsequent to its discovery in 1990, DC has shown to be very powerful against many known block ciphers. However, it is not very successful against the DES. It turned out that the designer of the DES had anticipated DC 15 years earlier [Cop94] through the non-linearity design of the S-boxes.

An interesting feature of the DES (in fact, the Feistel cipher) is that the S-boxes in function $f(R_{i-1}, k_i)$ need not be invertible. This is shown in Example 7.3 as encryption and decryption working for arbitrary $f(R_{i-1}, k_i)$. This feature saves space for the hardware realization of the DES.

We shall omit the description of the internal details of the S-boxes, of the key-schedule function and of the initial-permutation function. These details are out of the scope of the book. The interested reader is referred to §2.6.2 of [Den82] for these details.

7.5.3 The Security of the DES

Debates on the security of the DES started soon when the DES was proposed to be the encryption standard. Detailed discussions and historical accounts can be found in various cryptographic texts, e.g., §2 of [SB92], §3.3 of [Sti95], and §7.4.3 of [MvOV97]. Later, it became more and more clear that these debates reached a single main critique: the DES has a relatively short key length. This is regarded as the only most serious weakness of the DES. Attacks related to this weakness involve exhaustively testing keys, using a known pair of plaintext and ciphertext messages, until the correct key is found. This is the so-called **brute-force key search attack**.

However, we should not regard a brute-force key search attack as a real attack. This is because the cipher designers not only have anticipated it, but also exactly have hoped this to be the only means for an adversary. Therefore, given the computation technology of the 1970s, the DES is a very successful cipher.

One solution to overcome the short-key limitation is to run the DES algorithm

a multiple number of times using different keys. One of such proposals is called encryption-decryption-encryption-triple DES scheme [Tuc79]. Encryption under this scheme can be denoted by

$$c = \mathcal{E}_{k_1}(\mathcal{D}_{k_2}(\mathcal{E}_{k_1}(m))),$$

and decryption by

$$m = \mathcal{D}_{k_1}(\mathcal{E}_{k_2}(\mathcal{D}_{k_1}(c))).$$

In addition to achieving an effect of enlarging the key space, this scheme also achieves an easy compatibility with the single-key DES, if $k_1 = k_2$ is used. The triple DES can also use three different keys, but then is not compatible to the single-key DES.

The short-key weakness of the DES became evident in the 1990s. In 1993, Wiener argued that a special-purpose VLSI DES key search machine can be built at the cost of US\$1,000,000. Given a pair of plaintext-ciphertext messages, this machine can find the key in expected 3.5 hours [Wie94]. On July 15, 1998, a coalition of Cryptography Research, Advanced Wireless Technologies and Electronic Frontier Foundation announced a record-breaking the DES key search attack: they built a key search machine, called the DES Cracker, with cost under US\$250,000, and successfully found the key of the RSA's DES Challenge after searching for 56 hours [Fou98]. This result demonstrates that a 56-bit key is too short for a secure secret-key cipher for the late 1990s computation technology.

7.6 The Advanced Encryption Standard (AES)

On January 2, 1997, the United States' National Institute of Standards and Technology (NIST) announced the initiation of a new symmetric-key block cipher algorithm as the new encryption standard to replace the DES. The new algorithm would be named the Advanced Encryption Standard (AES). Unlike the closed design process for the DES, an open call for the AES algorithms was formally made on September 12, 1997. The call stipulated that the AES would specify an unclassified, publicly disclosed symmetric-key encryption algorithm(s); the algorithm(s) must support (at a minimum) block sizes of 128-bits, key sizes of 128-, 192-, and 256-bits, and should have a strength at the level of the triple DES, but should be more efficient than the triple DES. In addition, the algorithm(s), if selected, must be available royalty-free, worldwide.

On August 20, 1998, NIST announced a group of fifteen AES candidate algorithms. These algorithms had been submitted by members of the cryptographic community from around the world. Public comments on the fifteen candidates were solicited as the initial review of these algorithms (the period for the initial public comments was also called the Round 1). The Round 1 closed on April 15, 1999. Using the analyses and comments received, NIST selected five algorithms

from the fifteen. The five AES finalist candidate algorithms were MARS [BCD⁺], RC6 [RRY98], Rijndael [DR98], Serpent [ABK], and Twofish [SKW⁺98]. These finalist algorithms received further analysis during a second, more in-depth review period (the Round 2). In the Round 2, comments and analysis were sought on any aspect of the candidate algorithms, including, - but not limited to, - the following topics: cryptanalysis, intellectual property, cross-cutting analyses of all of the AES finalists, overall recommendations and implementation issues. After the close of the Round 2 public analysis period on May 15, 2000, NIST studied all available information in order to make a selection for the AES. On October 2, 2000, NIST announced that it has selected Rijndael to propose for the AES.

Rijndael is designed by two Belgium cryptographers: Daemen and Rijmen.

7.6.1 An Overview of the Rijndael Cipher

Rijndael is a block cipher with a variable block size and variable key size. The key size and the block size can be independently specified to 128, 192 or 256 bits. For simplicity we will only describe the minimum case of the 128-bit key size and the same block size. Our confined description will not cause any loss of generality to the working principle of the Rijndael cipher.

In this case, a 128-bit message (plaintext, ciphertext) block is segmented into 16 bytes (a byte is a unit of 8 binary bits, so $128 = 16 \times 8$):

$$\text{InputBlock} = m_0, m_1, \dots, m_{15}.$$

So is a key block:

$$\text{InputKey} = k_0, k_1, \dots, k_{15}.$$

The data structure for their internal representation is a 4×4 matrix:

$$\text{InputBlock} = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix},$$

$$\text{InputKey} = \begin{pmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{pmatrix}.$$

Like the DES (and most modern symmetric-key block ciphers), the Rijndael algorithm comprises a plural number of iterations of a basic unit of transformation: “round”. In the minimum case of 128-bit message-block and key-block size, the number of rounds is 10. For larger message sizes and key sizes, the number of rounds should be increased accordingly and is given in Figure 5 of [NIS01b].

A round transformation in Rijndael is denoted by

$$\text{Round}(\text{State}, \text{RoundKey}).$$

Here *State* is a round-message matrix and is treated as both input and output; *RoundKey* is a round-key matrix and is derived from the input key via key schedule. The execution of a round will cause the elements of *State* to change value (i.e., to change its state). For encryption (respectively, decryption), *State* input to the first round is Input Block which is the plaintext (respectively, ciphertext) message matrix, and *State* output from the final round is the ciphertext (respectively, plaintext) message matrix.

The round (other than the final round) transformation is composed of four different transformations which are internal functions to be described in a moment:

```
Round(State, RoundKey) {
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State, RoundKey);
}
```

The final round, denoted by

$$\text{FinalRound}(\text{State}, \text{RoundKey}),$$

is slightly different: it is equal to $\text{Round}(\text{State}, \text{RoundKey})$ with the `MixColumns` function removed. This is analogous to the situation of the final round in the DES where an additional swap between the output half data blocks is applied.

The round transformations are invertible for the purpose of decryption. The respective reverse round transformations should be denoted by

$$\text{Round}^{-1}(\text{State}, \text{RoundKey}), \text{ and}$$

$$\text{FinalRound}^{-1}(\text{State}, \text{RoundKey}),$$

respectively. We shall see below that the four internal functions are all invertible.

7.6.2 The Internal Functions of the Rijndael Cipher

Let us now describe the four internal functions of the Rijndael cipher. We shall only describe the functions for the encryption direction. Because each of the four

internal functions is invertible, decryption in Rijndael merely applies their respective inversions in the reverse direction.

The internal functions of the Rijndael cipher work in the finite field \mathbb{F}_{2^8} . The field is realized as all polynomials modulo the irreducible polynomial

$$f(X) = X^8 + X^4 + X^3 + X + 1$$

over \mathbb{F}_2 . That is, in specific, the Rijndael field is $\mathbb{F}_2[X]_{X^8+X^4+X^3+X+1}$. Let us name this field the “Rijndael field”. We have studied the Rijndael field in Chapter 5, Examples 5.14, 5.15 and 5.16, where we demonstrated the following operations:

- Mapping between an integer byte and a field element (Example 5.14);
- Addition between two field elements (Example 5.15);
- Multiplication between two field elements (Example 5.16).

Our study there can now help us to describe the Rijndael internal functions.

First of all, as we have already described, a block of message (a state) and a block of key in the Rijndael cipher are segmented into bytes. From the simple 1-1 mapping scheme described in Example 5.14, these bytes will be viewed as field elements and will be processed by several Rijndael internal functions which we now describe.

7.6.2.1 Internal function SubBytes(*State*)

This function provides a non-linear substitution on each byte (i.e., x) of *State*. Any non-zero byte $x \in (\mathbb{F}_{2^8})^*$ is substituted by the following transformation:

$$y = Ax^{-1} + b. \quad (7.6.1)$$

where

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

If x is the zero byte, then $y = b$ is the SubBytes transformation result.

We should notice that the non-linearity of the transformation in (7.6.1) comes from the inversion x^{-1} only. Should the transformation be applied on x directly, the affine equation in (7.6.1) would then be *absolutely linear*!

Since the 8×8 constant matrix A is an invertible one (i.e., its rows are linearly independent in \mathbb{F}_{2^8}), the transformation in (7.6.1) is invertible. Hence, function $\text{SubBytes}(State)$ is invertible.

7.6.2.2 Internal function $\text{ShiftRows}(State)$

This function operates on each row of $State$. For the case of 128-bit block size, it is the following transformation:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}. \quad (7.6.2)$$

Since this transformation is a positional one (not a field operation), it is of course mechanically invertible.

7.6.2.3 Internal function $\text{MixColumns}(State)$

This function operates on each column of $State$. So for $State$ of four columns of the right-hand-side matrix in (7.6.2), $\text{MixColumns}(State)$ repeats four iterations. The following description is for one column only. The output of an iteration is still a column.

First, let

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

be a column in the right-hand-side matrix in (7.6.2). Notice that we have omitted the column number for clarity in exposition.

This column is interpreted into a degree-3 polynomial:

$$s(X) = s_3X^3 + s_2X^2 + s_1X + s_0.$$

Notice that because the coefficients of $s(X)$ are bytes, i.e., elements in \mathbb{F}_{2^8} , this polynomial is over \mathbb{F}_{2^8} , and hence is *not* an element in the Rijndael field.

The operation on the column $s(X)$ is defined by multiplying this polynomial with a fixed degree-3 polynomial $c(X)$, modulo $X^4 + 1$:

$$c(X) \cdot s(X) \pmod{X^4 + 1}, \quad (7.6.3)$$

where the fixed polynomial $c(X)$ is

$$c(X) = c_3X^3 + c_2X^2 + c_1X + c_0 = '03'X^3 + '01'X^2 + '01'X + '02'.$$

The coefficients of $c(X)$ are also elements in \mathbb{F}_{2^8} (denoted by the hexadecimal representations of the respective bytes, or field elements).

We should notice that the multiplication in (7.6.3) is *not* a Rijndael field operation: $c(X)$ and $s(X)$ are not even Rijndael field elements. Also because $X^4 + 1$ is reducible in \mathbb{F}_{2^8} (since $X^4 + 1 = (X + 1)^4$ in \mathbb{F}_2), the multiplication in (7.6.3) is not even an operation in any field. The only reason for this multiplication being performed modulo a degree-4 polynomial is in order for the operation to output a degree-3 polynomial, that is, to achieve a transformation from a column (a degree-3 polynomial) to a column (a degree-3 polynomial).

The reader may apply the long division method in Example 5.12 to confirm the following equation over \mathbb{F}_{2^8} (with noticing that subtraction in \mathbb{F}_{2^8} is identical to addition):

$$X^i \pmod{X^4 + 1} = X^{i \pmod{4}}.$$

Therefore, in the product (7.6.3), the coefficient for X^i (for $i = 0, 1, 2, 3$) must be the sum of $c_j s_k$ satisfying $j + k = i \pmod{4}$ (where $j, k = 0, 1, 2, 3$). For example, the coefficient for X^2 in the product is

$$c_2 s_0 + c_1 s_1 + c_0 s_2 + c_3 s_3.$$

The multiplication and addition are in \mathbb{F}_{2^8} . For this reason, it is now easy to check that the polynomial multiplication in (7.6.3) can be achieved by taking the following linear algebraic one:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}. \quad (7.6.4)$$

We further notice that because $c(X)$ is relatively prime to $X^4 + 1$ over \mathbb{F}_{2^8} , the inversion $c(X)^{-1} \pmod{X^4 + 1}$ exists over \mathbb{F}_{2^8} . This is equivalent to saying that the matrix, and hence the transformation, in (7.6.4) are invertible.

7.6.2.4 Internal function $\text{AddRoundKey}(\text{State}, \text{RoundKey})$

This function merely adds, byte by byte and bit by bit, the elements of RoundKey to those of State . Here “add” is addition in \mathbb{F}_2 (i.e., bitwise XOR) and is trivially invertible; the inversion is “add” itself.

The RoundKey bits have been “scheduled”, i.e., the key bits for different rounds are different, and are derived from the key using a fixed (non-secret) “key schedule” scheme. For details for “key schedule” see Figure 12 of [NIS01b].

To this end we have completed the description of the Rijndael internal functions and hence the encryption operation.

7.6.2.5 Decryption operation

As we have seen that each of the four internal functions are invertible, the decryption is merely to invert the encryption in the reverse direction, i.e., applying

$\text{AddRoundKey}(\text{State}, \text{RoundKey})^{-1};$

$\text{MixColumns}(\text{State})^{-1};$

$\text{ShiftRows}(\text{State})^{-1};$

$\text{SubBytes}(\text{State})^{-1}.$

We should notice that, unlike in the case of a Feistel cipher where encryption and decryption use the same circuit (hardware) or code (software), the Rijndael cipher must implement different circuits and codes for encryption and decryption, respectively.

7.6.3 Summary of the Roles of the Rijndael Internal Functions

At the end of our description of the Rijndael cipher let us provide a summary on the roles of the four internal functions.

- **SubBytes** is intended to achieve a non-linear substitution cipher. As we have discussed in §7.5.2, non-linearity is an important property for a block cipher to prevent differential cryptanalysis.
- **ShiftRows** and **MixColumns** are intended to achieve a mixture of the bytes positioned in different places of a plaintext message block. Typically, plaintext messages have a low-entropy distribution in the message space due to the high redundancy contained in natural languages and business data (that is, typical plaintexts concentrate in a small subspace of the whole message space). A mixture of the bytes in different positions of a message block causes a wider distribution of messages in the whole message space. This is essentially the mixing property modelled by Shannon in 7.1.2.
- **AddRoundKey** provides the necessary secret randomness to the message distribution.

These functions repeat a plural number of times (minimum 10 for the case of 128-bit key and data size), and the result is the Rijndael cipher.

7.6.4 Fast and Secure Implementation

We have seen that the Rijndael internal functions are very simple and operate in trivially small algebraic spaces. As a result, implementations of these internal functions can be done with extremely good efficiency. From our descriptions in the

Rijndael internal functions, we see that only **SubBytes** and **MixColumns** have non-trivial algebraic operations and hence worthy of fast implementation considerations.

First, in **SubBytes**, the calculation of x^{-1} can be efficiently done using a “table-lookup” method: a small table of $2^8 = 256$ pairs of bytes can be built once and use forever (i.e., the table can be “hardwired” into hardware or software implementations). In this table of pairs, the zero byte is paired with the zero byte; the rest 255 entries in the table are the 255 cases of the pair (x, x^{-1}) where inversion is performed in the field $\mathbb{F}_2[X]_{X^8+X^4+X^3+X+1}$. The “table-lookup” method not only is efficient, but also prevents a **timing analysis attack** which is based on observing the operation time difference for different data which may suggest whether an operation is performed on bit 0 or bit 1.

Because the matrix A and the vector b in (7.6.1) are constants, the “table lookup” method can actually include the whole transformation (7.6.1) altogether, that is, the table of 256 entries are the pairs (x, y) in $\mathbb{F}_2[X]_{X^8+X^4+X^3+X+1}$, except for the pair $(0, b)$.

Clearly, inversion is merely to use the table in the opposite direction!

Next, in **MixColumns**, multiplication between elements in $\mathbb{F}_2[X]_{X^8+X^4+X^3+X+1}$, i.e., that between coefficients of $c(X)$ and $s(X)$, or equivalently, that between an element of the fixed matrix and that in a column vector in (7.6.4), can also be realized via a “table lookup” method. This means that an implementation (either in software or hardware) builds up a table of $256 \times 256 = 65536$ entries: $z = x \cdot y$ (field multiplication), just like every primary school pupil recites a much smaller multiplication table. This realization not only is fast, but also decreases the risk of the timing analysis attack.

The linear algebraic operation in (7.6.4) and its inversion also have a fast “hard-wired” implementation method. The interested reader is referred to [DR02].

7.6.5 Positive Impact of the AES on Applied Cryptography

The introduction of the AES will in turn introduce a few a positive changes in applied cryptography.

First, multiple encryption, such as triple-DES, will become unnecessary with the AES: the enlarged and variable key and data-block sizes of 128, 192 and 256 can accommodate a wide-spectrum of security strengths for various application needs. Since multiple encryption uses a plural number of keys, the avoidance of using multiple encryption will mean a reduction on the number of cryptographic keys that an application has to manage, and hence will simplify the design of security protocols and systems.

Secondly, wide use of the AES will lead to the emergence of new hash functions of compatible security strengths. In several ways, block cipher encryption algorithms are closely related to hash functions (see §9.2.1). For instance, it has

been a standard practice that block cipher encryption algorithms are often used to play the role of one-way hash functions. The logging-in authentication protocol of the UNIX^a operating system [MT79] is a well-known example: the system stores the ciphertext of the DES encryption of a user's identity (UID) where the user's password is used as the encryption key; here the DES encryption function actually plays the role of a one-way hash function. Another example of using block cipher encryption algorithms to realize (keyed) one-way hash functions can be seen in §9.2.3. In practice, hash functions are also commonly used as pseudo-random number functions for generating keys for block cipher algorithms. With the AES' variable and enlarged key and data-block sizes, hash functions of compatible sizes will be needed. However, due to the square-root attack (the birthday attack, see §3.5 and §9.2.1), a hash function should have a size which doubles the size of a block cipher's key or data-block size. Thus, matching the AES' sizes of 128, 192 and 256, new hash functions of output sizes of 256, 384 and 512 are needed. The ISO/IEC are currently in the process of standardizing hash functions SHA-256, SHA-384 and SHA-512 [ISO01].

7.7 Confidentiality Modes of Operation

A block cipher processes (encrypts or decrypts) messages as data blocks. Usually, the size of a bulk message (i.e., a message string) is larger than the size of the message block of a block cipher, the long message is divided into a series of sequentially listed message blocks, and the cipher processes these blocks one at a time.

A number of different modes of operation have been devised on top of an underlying block cipher algorithm. These modes of operation (except a trivial case of them) provide several desirable properties to the ciphertext blocks, such as adding nondeterminism (randomness) to a block cipher algorithm, padding plaintext messages to an arbitrary length (so that the length of a ciphertext needn't be related to that of the corresponding plaintext), control of error propagation, generation of key stream for a stream cipher, etc. We may consider that the use of these modes of operations (again, except a trivial case of them, to become clear in a moment) turns a "textbook version" block cipher into a "fit-for-application" version.

We describe here five usual modes of operation. They are **electronic code-book (ECB)** mode, **cipher block chaining (CBC)** mode, **output feedback (OFB)** mode, **cipher feedback (CFB)** mode, and **counter (CTR)** mode. Our description follow the most recent NIST recommendation [NIS01a].

In our description, we will use the following notation:

- $\mathcal{E}()$: the encryption algorithm of the underlying block cipher;
- $\mathcal{D}()$: the decryption algorithm of the underlying block cipher;

^aUNIX is a trademark of Bell Laboratories.

- n : the binary size of the message block of the underlying block cipher algorithm (in all block ciphers we consider, the plaintext and ciphertext message spaces coincide, and so n is the block size of both input and output of the block cipher algorithm);
- P_1, P_2, \dots, P_m : m successive segments of plaintext messages input to a mode of operation;
 - the m -th segment may have a smaller size than the other segments and in that case a padding is applied to make the m -th segment have the same size as the other segments;
 - the size of a message segment is equal to n (the block size) in some modes of operation, and is any positive number less than or equal to n in other modes of operation;
- C_1, C_2, \dots, C_m : m successive segments of ciphertext messages output from a mode of operation;
- $\text{LSB}_u(B)$, $\text{MSB}_v(B)$: the least u , and the most v , significant bits of the block B , respectively; for example

$$\text{LSB}_2(1010011) = 11, \quad \text{MSB}_5(1010011) = 10100;$$

- $A \parallel B$: concatenation of the data blocks A and B ; for example,

$$\text{LSB}_2(1010011) \parallel \text{MSB}_5(1010011) = 11 \parallel 10100 = 1110100.$$

7.7.1 The Electronic Codebook Mode (ECB)

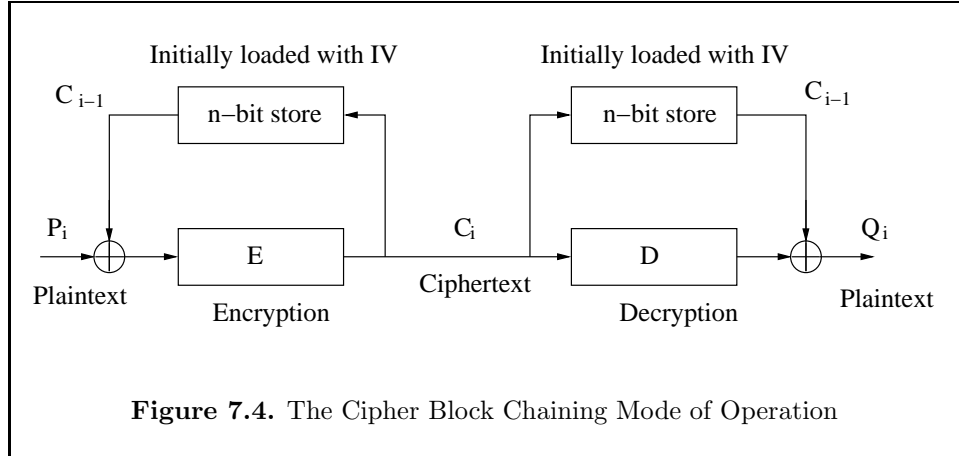
The most straightforward way of encrypting (or decrypting) a series of sequentially listed message segments is just to encrypt (or decrypt) them one another separately. This is what we have mentioned the trivial case which does not turn a “textbook” block cipher into a “fit-for-application” version.

In this case, a message segment is just a message block. Analogous to the assignment of code words in a codebook, this natural and simple method gets an official name: electronic codebook mode of operation (ECB). The ECB mode is defined as follows:

ECB Encryption $C_i \stackrel{\text{def}}{=} \mathcal{E}(P_i)$, $i = 1, 2, \dots, m$;

ECB Decryption $P_i \stackrel{\text{def}}{=} \mathcal{E}(C_i)$, $i = 1, 2, \dots, m$.

The ECB mode is deterministic, that is, if P_1, P_2, \dots, P_m are encrypted twice under the same key, the output ciphertext blocks will be the same. Therefore if this deterministic property is undesirable in an application (in general we do not wish a cipher to have such a deterministic behavior) then the ECB mode should not be used.



7.7.2 The Cipher Block Chaining Mode (CBC)

The cipher block chaining (CBC) mode of operation is a common block-cipher algorithm for encryption of general data. Working with the CBC mode, the output is a sequence of n -bit cipher blocks which are chained together so that each cipher block is dependent, not just on the plaintext block from which it immediately came, but on all the previous data blocks. The CBC mode has the following operations:

CBC Encryption $C_i \stackrel{\text{def}}{=} \mathcal{E}(P_i \oplus C_{i-1}), i = 1, 2, \dots, m;$

CBC Decryption $P_i \stackrel{\text{def}}{=} \mathcal{D}(C_i) \oplus C_{i-1}, i = 1, 2, \dots, m.$

The computation of the first ciphertext block C_1 needs a special input block C_0 which is conventionally called the “initial vector” (IV). An IV is a random n -bit block. In each session of encryption a new and random IV should be used. Since an IV is treated as a ciphertext block, it need not be secret, but it must be unpredictable. From the encryption procedure we know that the first ciphertext block C_1 is randomized by the IV; and in the same way and in turn, a subsequent output ciphertext block is randomized by the immediate preceding ciphertext block. Hence, the CBC mode outputs randomized ciphertext blocks. The ciphertext messages sent to the receiver should include the IV. Thus, for m blocks of plaintext, the CBC mode outputs $m + 1$ ciphertext blocks.

Let Q_1, Q_2, \dots, Q_m be the data blocks output from decryption of the ciphertext blocks $C_0, C_1, C_2, \dots, C_m$. Then since

$$Q_i = \mathcal{D}(C_i) \oplus C_{i-1} = (P_i \oplus C_{i-1}) \oplus C_{i-1} = P_i,$$

indeed, the decryption works properly. Figure 7.4 provides an illustration of the CBC mode.

It may appear that, since the data blocks are chained together, the CBC mode

may provide protection against unauthorized data modification attacks such as deletion and insertion. This is in fact utterly false. However, several authentication protocols in two early draft international standard documents [ISO92], [ISO93] follow this wrong idea (general guideline for these protocols to use CBC is documented in [ISO96], [ISO91a]), and of course, these protocols are fatally flawed [MB93], [MB94]. We shall demonstrate the flaw in §15.2.1.2 by analyzing an authentication protocol in which the use of encryption follows the standard CBC implementation; the protocol is designed to expect that the use of CBC should provide a data-integrity protection on the ciphers, however, the protocol is flawed precisely due to missing of this service.

To randomize output ciphertext appears to be the only security service that the CBC mode offers.

7.7.3 The Cipher Feedback Mode (CFB)

The cipher feedback (CFB) mode of operation features feeding the successive cipher segments which are output from the mode back as input to the underlying block cipher algorithm. A message (plaintext or ciphertext) segment has a size s such that $1 \leq s \leq n$. The CFB mode requires an IV as the initial random n -bit input block. The IV need not be secret, but it must be unpredictable. The CFB mode has the following operations:

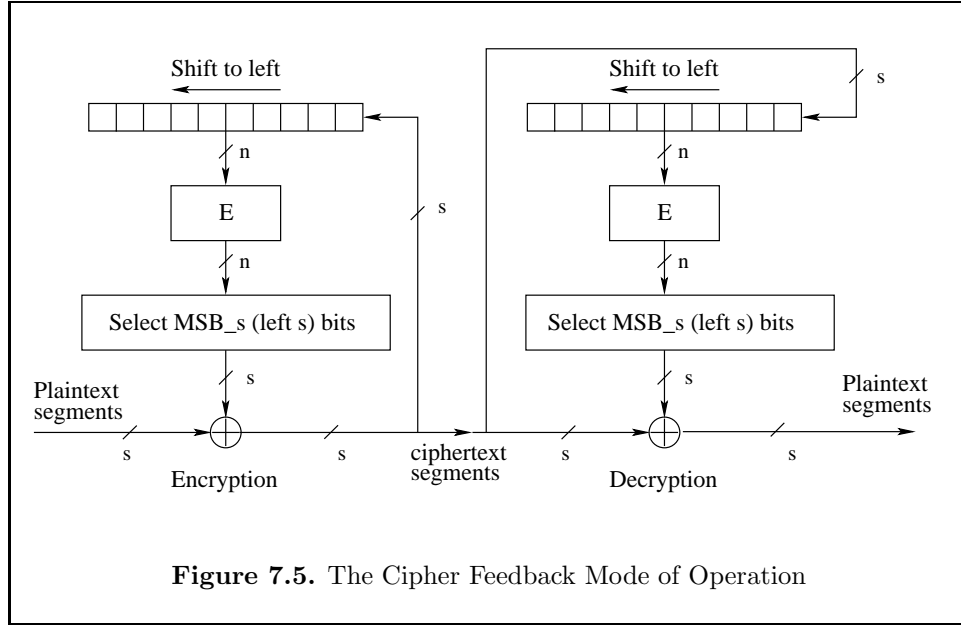
CFB Encryption Input: IV, P_1, \dots, P_m ; Output: IV, C_1, \dots, C_m ;

$$\begin{aligned} I_1 &\stackrel{\text{def}}{=} IV; \\ I_i &\stackrel{\text{def}}{=} \text{LSB}_{n-s}(I_{i-1}) \parallel C_{i-1} \quad i = 2, \dots, m; \\ O_i &\stackrel{\text{def}}{=} \mathcal{E}(I_i) \quad i = 1, 2, \dots, m; \\ C_i &\stackrel{\text{def}}{=} P_i \oplus \text{MSB}_s(O_i) \quad i = 1, 2, \dots, m; \end{aligned}$$

CFB Decryption Input: IV, C_1, \dots, C_m ; Output: P_1, \dots, P_m ;

$$\begin{aligned} I_1 &\stackrel{\text{def}}{=} IV; \\ I_i &\stackrel{\text{def}}{=} \text{LSB}_{n-s}(I_{i-1}) \parallel C_{i-1} \quad i = 2, \dots, m; \\ O_i &\stackrel{\text{def}}{=} \mathcal{E}(I_i) \quad i = 1, 2, \dots, m; \\ P_i &\stackrel{\text{def}}{=} C_i \oplus \text{MSB}_s(O_i) \quad i = 1, 2, \dots, m; \end{aligned}$$

Observe that, in the CFB mode, the encryption function of the underlying block cipher is used in both ends of the encryption and the decryption. As a result, the underlying cipher function E can be any (keyed) one-way transformation, such as a one-way hash function. The CFB mode can be considered as a key stream generator for a stream cipher with the encryption being the Vernam cipher between the key stream and the message segments. Similar to the CBC mode, a ciphertext segment



is a function of all preceding plaintext segment and the IV. Figure 7.5 provides an illustration of the CFB mode.

7.7.4 The Output Feedback Mode (OFB)

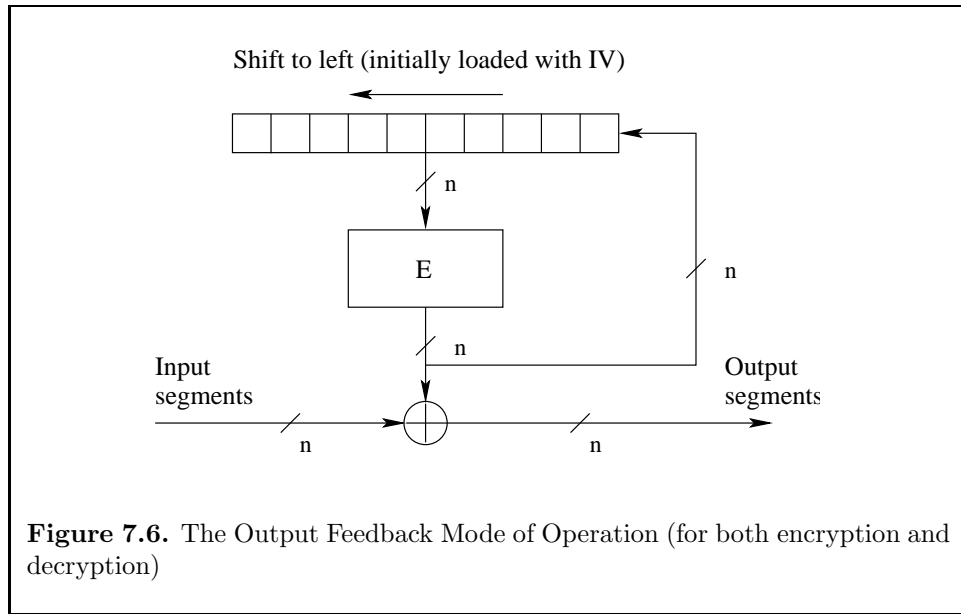
The output feedback (OFB) mode of operation features feeding the successive output blocks from the underlying block cipher back to it. These feedback blocks form a string of bits which is used as the key stream of the Vernam cipher, that is, the key stream is XOR-ed with the plaintext blocks. The OFB mode requires an IV as the initial random n -bit input block. The IV need not be secret, but it must be unpredictable. The OFB mode has the following operations:

OFB Encryption Input: IV, P_1, \dots, P_m ; Output: IV, C_1, \dots, C_m ;

$$\begin{aligned}
 I_1 &\stackrel{\text{def}}{=} IV; \\
 I_i &\stackrel{\text{def}}{=} O_{i-1} \quad i = 2, \dots, m; \\
 O_i &\stackrel{\text{def}}{=} \mathcal{E}(I_i) \quad i = 1, 2, \dots, m; \\
 C_i &\stackrel{\text{def}}{=} P_i \oplus O_i \quad i = 1, 2, \dots, m;
 \end{aligned}$$

OFB Decryption Input: IV, C_1, \dots, C_m ; Output: P_1, \dots, P_m ;

$$\begin{aligned} I_1 &\stackrel{\text{def}}{=} IV; \\ I_i &\stackrel{\text{def}}{=} O_{i-1} \quad i = 2, \dots, m; \\ O_i &\stackrel{\text{def}}{=} \mathcal{E}(I_i) \quad i = 1, 2, \dots, m; \\ P_i &\stackrel{\text{def}}{=} C_i \oplus O_i \quad i = 1, 2, \dots, m; \end{aligned}$$



In the OFB mode, the encryption and the decryption are identical: XORing the input message blocks with the key stream which is generated by the feedback circuit. The feedback circuit actually forms a finite state machine with the state solely determined by the encryption key for the underlying block cipher algorithm and the IV. Thus, if a transmission error occurred to a cipher block, then only the plaintext block in the corresponding position can be garbled. Therefore, the OFB mode is suitable for encryption messages for which retransmission is not possible, like radio signal. Similar to the CFB mode, the underlying block cipher algorithm can be replaced with a keyed one-way hash function. Figure 7.6 provides an illustration of the CFB mode.

7.7.5 The Counter Mode (CTR)

The counter (CTR) mode features feeding the underlying block cipher algorithm with a counter value which counts up from an initial value. With the counter

counting up, the underlying block cipher algorithm outputs successive blocks to form a string of bits. This string of bits is used as the key stream of the Vernam cipher, that is, the key stream is XOR-ed with the plaintext blocks. The CTR mode has the following operations (where Ctr_1 is an initial non-secret value of the counter):

CTR Encryption Input: Ctr_1, P_1, \dots, P_m ; Output: Ctr_1, C_1, \dots, C_m ;

$$C_i \stackrel{\text{def}}{=} P_i \oplus \mathcal{E}(Ctr_i) \quad i = 1, 2, \dots, m;$$

CTR Decryption Input: Ctr_1, C_1, \dots, C_m ; Output: P_1, \dots, P_m ;

$$P_i \stackrel{\text{def}}{=} C_i \oplus \mathcal{E}(Ctr_i) \quad i = 1, 2, \dots, m.$$

Without feedback, the CTR mode encryption and decryption can be performed in parallel. This is the advantage that the CTR mode has over the CFB and OFB modes. Due to its simplicity, we omit the illustration for the CTR mode.

7.8 Key Channel Establishment for Symmetric Cryptosystems

Before two principals can start confidential communications by using symmetric cryptosystems, they must first establish correct cryptographic keys shared between them. Here, “correct” not only means that a key established is bit-by-bit correct, i.e., not corrupted, but also means that both parties must be assured that the key is exclusively shared with the intended communication partner. A communication channel that causes a key to be correctly established is called a **key channel** (see Figure 7.1). A key channel is a separate channel from a message channel. The difference between them is that a key channel is a protected one, while a communication channel is an unprotected one. In symmetric cryptosystems, since the encryption key is equal to the decryption key, the key channel must preserve both the confidentiality and the authenticity of the key.

A key channel for a symmetric cryptosystem can be established by two means: conventional techniques and public-key techniques.

Conventional Techniques In the system setting-up time, a physically secure means, e.g., a courier delivery service, can be employed to make two users to exclusively share an initial key. Once an initial key is shared between two principals, a long-term key channel is setup between them. The two users can run various authentication protocols to maintain the long-term key channel; for example, they can establish further shared keys (long-term keys or session keys) for further secure communications between them. In order to reduce the complexity of key management task, the initial key setting-up, or long-term key channel should be between an end-user and an authentication

server who is trusted by all the end-users in the domain that the server serves (see §2.3 for the notion of this trust). Thus, an end-user does not have to manage many keys as she/he would have to should a long-term key channel be established between any two end-user principals. In Chapter 2 we have seen a few examples of authentication and key establishment protocols which serve for setting-up session keys between any two end-user principals using long-term key channels between end-user principals and an authentication server. We will see more such protocols in Chapter 15 when we discuss the topic of formal analysis of authentication protocols.

Public-key Techniques An important advantage of public-key cryptography is its ease of establishing a key channel between two remote principals. There are a number of public-key techniques for key channel establishment. We shall introduce public-key cryptography in the next chapter. However, with public-key cryptography, there is still a need to initially establish a secure key channel from a user toward the system. Here, “secure” means authentication: a given public key can be identified to be really owned by a claimed principal. Nevertheless, key channel establishment using public-key techniques does not involve handling of any secret. Indeed, the setting-up of a key channel regarding a public key is purely an authentication problem. In Figure 7.1 we have illustrated that a public key channel can be based on a directory service. We will introduce techniques for setting up public key channels in Chapter 12.

7.9 Chapter Summary

In this chapter we have studied the principle of symmetric encryption algorithms and introduced several symmetric encryption schemes.

We start with introducing classical ciphers and considering their conditional security under Shannon’s information theory. We point out that the working principle of the classical ciphers: substitution, is still the most important kernel technique in the construction of modern symmetric encryption algorithms.

Two modern block encryption algorithms, the DES and the AES, are introduced. The DES is introduced for the reasons of its historical position and the still-alive usefulness of its Feistel cipher design structure. The AES, as the newly established encryption standard, is described with detailed explanations on its working principle. We also consider methods for fast and secure realization of the AES, and discuss the positive impact the AES may have on applied cryptography.

We then introduce various standard modes of operation for using block ciphers and discuss issues for the establishment of secure key channels between communication partners who wish to communicate confidential information.

ENCRYPTION — ASYMMETRIC TECHNIQUES

8.1 Introduction

Early ciphers (such as the Caesar cipher) depended on keeping the entire encryption process secret.

Modern ciphers such as the DES and the AES follow the Kerchoffs' principle (see §7.1): the algorithmic details of these ciphers are made public for open scrutiny. In so doing, the designers of these ciphers wish to demonstrate that the security of their cryptosystems reside solely in the choice of the secret encryption keys.

There is further room to practice the Kerchoffs' principle of reducing the secret component in an encryption algorithm. Consider Shannon's semantic property of encryption: a mixing-transformation which distributes meaningful messages from the plaintext region \mathcal{M} fairly uniformly over the entire message space \mathcal{C} (pages 711-712 of [Sha49]). We now know that such a random distribution can be achieved without need of using any secret. Diffie and Hellman first realized this in 1975 [DH76a] (the publication date of this paper was 1976, but the paper was first distributed in December 1975 as a preprint, see [Dif92]). They named their discovery **public-key** cryptography. At that time it was a totally new understanding of cryptography.

In a public-key cryptosystem, encryption uses no secret key; secret key is only needed in decryption time. In [DH76a], Diffie and Hellman sketched several mathematical transformations, which they termed **one-way trapdoor** functions, as possible candidates for realizing public-key cryptography. Informally speaking, a one-way trapdoor function has the following properties:

Property 8.1: (One-way Trapdoor Function) *A one-way trapdoor function, which we denote by $f_t(x) : \mathcal{D} \mapsto \mathcal{R}$, is a one-way function without using (knowing) the trapdoor t : i.e., it is easy to evaluate for all $x \in \mathcal{D}$, and is difficult to invert for*

almost all values in \mathcal{R} . However, if t is used (known) then for all values $y \in \mathcal{R}$, it is easy to compute $x \in \mathcal{D}$ satisfying $y = f_t(x)$.

The notion of one-way trapdoor function forms the enabler for public-key cryptography. Opposing to the notion of secret-key or symmetric cryptosystems, a public-key cryptosystem based on a one-way trapdoor function is also referred to as **asymmetric cryptosystems** due to the asymmetric property of one-way trapdoor functions. Although the several one-way trapdoor functions considered in the first paper of Diffie and Hellman on public-key cryptography (i.e., [DH76a]) were not very plausible due to their poor asymmetry, Diffie and Hellman soon proposed a successful function: modulo exponentiation, and used it to demonstrate the famous cryptographic protocol: the **Diffie-Hellman key exchange protocol** [DH76b] (see §8.2). To this day, this first successful realization of public-key crypto-algorithm is still in wide use and under endless further development.

In 1974, Merkle discovered a mechanism to realize cryptographic key agreement via an apparent asymmetric computation, which is now known as the **Merkle's puzzle** [Mer78]. The asymmetric computation in the Merkle's puzzle means that the computational complexity for legitimate participants of a key agreement protocol and that for an eavesdropper are drastically different: the former is feasible and the latter is not. The Merkle's puzzle was the first effective realization of a one-way trapdoor function. Although the Merkle's puzzle may not be considered suitable for modern cryptographic applications (as the asymmetry is between n and n^2), the insight it revealed was monumental to the discovery of public-key cryptography.

It is now known that Cocks, a British cryptographer, invented the first public-key cryptosystem in 1973 (see e.g., [Sin99]). Cocks' encryption algorithm, named "non-secret key encryption", is based on the difficulty of integer factorization and is essentially the same as the RSA cryptosystem (see §8.3). Unfortunately, Cocks' algorithm was classified. In December 1997, the British government's communications services - Electronics Security Group (CESG), released Cocks' algorithm. Although it appeared to be the case that the discovery of public-key cryptography by the open research community took place after the notion was known in a closed circle, we must point out that it was the open research community that identified the two most important applications of public-key cryptography: (i) digital signatures (see §9.3), and (ii) secret key establishment over public communications channels (see §8.2). These two applications have enabled today's proliferation of secure electronic commerce over the Internet.

In the remainder of this chapter we will introduce the initial work of Diffie and Hellman and several well-known public-key cryptosystems. Along with the introduction we will specify several computational problems and their assumed difficulties which underlie the security of the algorithms introduced. We will also define relevant attacking models and analyze the security of the encryption algorithms against the attacks under these models.

8.1.1 Insecurity of Textbook Encryption Algorithms

However, we should notice that quite a few encryption algorithms to be introduced in this chapter should be labelled “textbook” versions (in particular, all of the public-key algorithms in this chapter are). The name is given as a result of most textbooks in cryptography introducing such algorithms even though such algorithms are actually not suitable for use in real-world applications. The textbook encryption algorithms do not fall into the category of good cryptosystems. In general, a textbook encryption algorithm has the following confidentiality property:

Property 8.2: “Security” of Textbook Encryption Algorithms *Within the scope of this chapter, security (confidentiality) for a cryptosystem is considered in the following two senses:*

- i) **All-or-nothing secrecy.** *For a given ciphertext output from a given encryption algorithm, the attacker’s task is to retrieve the whole plaintext block; or for a given pair of plaintext and ciphertext under a given encryption algorithm, the attacker’s task is to uncover the whole block of the underlying secret key. The attacker either succeeds with obtaining the whole block of the targeted secret, or fails with nothing. We should pay particular attention to the meaning of “nothing”: it means that the attacker does not have any knowledge about the targeted secret before or after its attacking attempt.*
- ii) **Passive attacker.** *The attacker does not manipulate or modify ciphertexts using data she/he has in possession, and does not ask a key owner for providing encryption or decryption services.* □

This notion of security is extremely weak, in fact, is uselessly weak and therefore should be better called insecurity. Regarding (i), in applications, plaintext data are likely to have some non-secret information which can be known to an attacker. For example, some data are always in a small range: a usual salary should be less than one million which is a small number in cryptographic sense; a usual password is a bit-string up to eight characters. Often, the known partial information will permit an attacker to succeed and obtains all, rather than “fail with nothing”. Regarding (ii), we should never expect an attacker to be so nice and remain in passive. The typical behavior of an attacker is to try all means available to it.

By stating Property 8.2, we make it explicit that within the scope of this chapter, we will not consider a stronger notion of security for encryption algorithms, and consequently, for some textbook encryption algorithms to be introduced here, we will not hope that they are secure in a strong sense. On the contrary, we will demonstrate but not try to fix a number of confidentiality flaws with some textbook encryption algorithms where the flaws are about partial information leakage or a result of an active attack.

Definitions for various more stringent notions of security against stronger (i.e., more real) attackers will be introduced in Chapter 13. Stronger and fit-for-application counterparts of the textbook encryption algorithms will be followed up in Chapter 14.

8.1.2 Chapter Outline

We begin with introducing several well-known public-key cryptographic functions. These are: the Diffie-Hellman key exchange protocol (§8.2), the textbook RSA (§8.3), Rabin (§8.4) and ElGamal (§8.5) cryptosystems. These basic public-key cryptographic functions will be introduced together with formal and complexity-theoretic based statements on the respective underlying intractability assumptions: the Diffie-Hellman and the discrete logarithm problems (§8.2.2), the RSA problem and the integer factorization problem (§8.3.2). We will also begin in this chapter to develop formal notions for describing various attacking models against public-key cryptosystems (§8.3.1). We will consider the need for a stronger security notion for public-key encryption (§8.6). Having introduced both symmetric and asymmetric cryptosystems, we shall introduce their combination: hybrid encryption schemes (§8.7). Finally, we examine the hardness of the basic public-key cryptographic functions by examining their bit security: the difficulty to find one bit (e.g., the least significant bit) of the plaintext given a ciphertext.

8.2 The Diffie-Hellman Key Exchange Protocol

With a symmetric cryptosystem it is necessary to transfer a secret key to both communicating parties before secure communication can begin. Prior to the birth of public-key cryptography, the establishment of a shared secret key between communication parties had always been a difficult problem because the task needed a secure confidential channel, and often such a channel meant physical delivery of keys by a special courier. An important advantage that public key cryptography provides over symmetric cryptography is the achievement of exchanging a secret key between remote communication parties with no need of a secure confidential channel. The first practical scheme to achieve this was proposed by Diffie and Hellman, known as the Diffie-Hellman exponential key exchange protocol [DH76b].

To begin with, users Alice and Bob are assumed to have agreed on a finite field \mathbb{F}_p and an element $g \in \mathbb{F}_p$ which generates a group of a large order. For simplicity, we consider the case of \mathbb{F}_p being a prime field, i.e., p is a prime number. The two parties may test the primality of p using Algorithm 4.5 where they have constructed p such that they know the complete factorization of $p - 1$; and then they may find a generator g (e.g., of the group \mathbb{F}_p^*) using Algorithm 5.1. By Theorem 5.11, each number in $[1, p)$ can be expressed as $g^x \pmod{p}$ for some x . Now p and g are the common input to the following (basic version) protocol.

Protocol 8.1: Diffie-Hellman Key Exchange

COMMON INPUT Pair (p, g) with p a prime, g a generator element in \mathbb{Z}_p^* ;

OUTPUT An element in \mathbb{Z}_p^* shared between Alice and Bob.

1. Alice picks $a \in_U (1, p-1)$ and sends to Bob: $g_a \stackrel{\text{def}}{=} g^a \pmod{p}$;
2. Bob picks $b \in_U (1, p-1)$ and sends to Alice: $g_b \stackrel{\text{def}}{=} g^b \pmod{p}$;
3. Alice computes $k \stackrel{\text{def}}{=} g_b^a \pmod{p}$;
4. Bob computes $k \stackrel{\text{def}}{=} g_a^b \pmod{p}$.

Figure 8.1.

It is easy to see from Protocol 8.1 that for Alice

$$k = g^{ba} \pmod{p},$$

and for Bob

$$k = g^{ab} \pmod{p}.$$

We note that since $ab \equiv ba \pmod{p-1}$, the two parties have computed the same value. This is how the Diffie-Hellman key exchange protocol achieves a shared key between two communication parties.

A system-wide users may share the common public parameters p and g .

Example 8.1: Let $p = 43$. Applying Algorithm 5.1 we find that 3 is a primitive root modulo 43. Let Alice and Bob share the public material elements $(p, g) = (43, 3)$.

For Alice and Bob to agree a secret key, Alice picks her random secret exponent 8, and sends to Bob $3^8 \equiv 25 \pmod{43}$. Bob picks his random secret 37, and sends to Alice $3^{37} \equiv 20 \pmod{43}$. The secret key agreed between them is

$$9 \equiv 20^8 \equiv 25^{37} \pmod{43}. \quad \square$$

We should add a few cautionary details to the implementation and the use of the Diffie-Hellman key exchange protocol.

- The common input p should be such a prime (or a prime power) that $p-1$ has a sufficiently large prime factor p' ; here “sufficiently large” means $p' > 2^{160}$. The need for p to have this property will be discussed in §8.2.2.

- The common input g need not be a generator of the group \mathbb{F}_p^* ; for ease of use (see the next bullet point), g should be an element of the order p' .
- Alice (respectively, Bob) should check $g_b \neq 1$ (respectively, $g_a \neq 1$); for g of the order p' , if they choose their exponents from $(1, p')$, then this checking step will guarantee that the shared key g^{ab} will be one chosen from the subgroup of \mathbb{F}_p of the order p' , that is, from a sufficiently large subgroup. For this purpose, p' should be part of the common input to the protocol.
- Alice (respectively, Bob) should erase her exponent a (respectively, his exponent b) upon termination of the protocol. In so doing, they will have a **forward secrecy** property on the exchanged key g^{ab} if they also properly dispose the exchanged key after their session communication ends. We will further discuss the “forward secrecy property” in §8.7.

8.2.1 The Man-in-the-Middle Attack

It should be noted that the Diffie-Hellman key exchange protocol does not support the authenticity of the key agreed. An active adversary in the middle of the communications between Alice and Bob can manipulate the protocol messages to succeed an attack called **man-in-the-middle attack**. Example 8.2 illustrates such an attack.

In an attack to a run of the protocol, Malice (the bad guy) intercepts and blocks Alice’s first message to Bob, g_a , and he masquerades as Alice and sends to Bob

“Alice” (Malice) sends to Bob: $g_m (\stackrel{\text{def}}{=} g^m) \pmod{p}$;

(The reader may recall our convention agreed in §2.5.2 for denoting Malice’s action of masquerading as other principals.) Bob will follow the protocol by replying g_b to “Alice” (Malice). This means that the value transmitted is again intercepted and blocked by Malice. Now Malice and Bob have agreed a key $g^{bm} \pmod{p}$ which Bob thinks to be shared with Alice.

Analogously, Malice can masquerade as Bob and agree another key with Alice (e.g., $g^{am} \pmod{p}$). After this, Malice can use these two keys to read and relay “confidential” communications between Alice and Bob, or to impersonate one of them to the other.

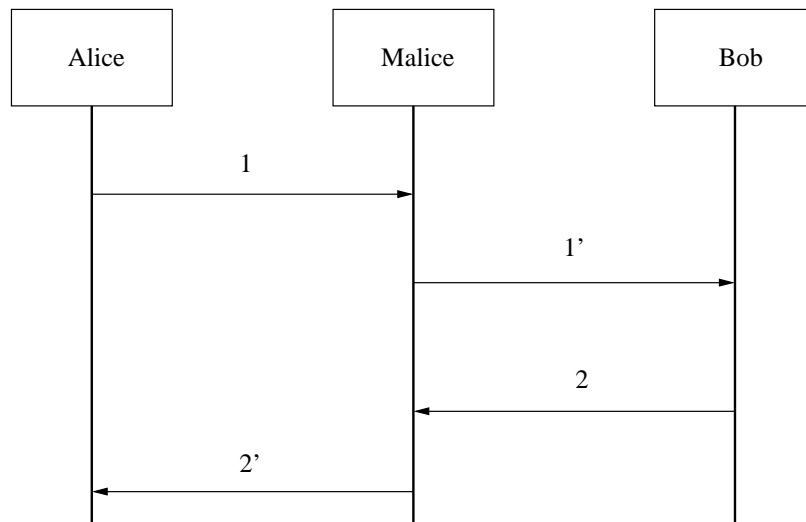
The man-in-the-middle attack on the Diffie-Hellman key exchange protocol is possible because the protocol does not provide an authentication service on the source of the protocol messages. In order to agree on a key which is exclusively shared between Alice and Bob, these principals must make sure that the messages they receive in a protocol run are indeed from the intended principals. In Chapter 10

Example 8.2: Man-in-the-Middle Attack on the Diffie-Hellman Key Exchange Protocol

PREMISE Same as Protocol 8.1;

RESULT OF ATTACK

Malice shares a key with Alice and another with Bob.



- 1 Alice picks $a \in_U [1, p)$ and sends to "Bob" (Malice): $g_a \stackrel{\text{def}}{=} g^a \pmod{p}$;
- 1' "Alice" (Malice) picks $m \in [1, p)$ and sends to Bob: $g_m \stackrel{\text{def}}{=} g^m \pmod{p}$;
- 2 Bob picks $b \in_U [1, p)$ and sends to "Alice" (Malice): $g_b \stackrel{\text{def}}{=} g^b \pmod{p}$;
- 2' "Bob" (Malice) sends to Alice: g_m ;
- 3 Alice computes $k_1 = g_m^a \pmod{p}$;
(* key agreed between Alice and Malice *)
- 4 Bob computes $k_2 = g_m^b \pmod{p}$.
(* key agreed between Bob and Malice *)

Figure 8.2.

we will study authentication techniques; there (§??) we will introduce methods for securely applying the Diffie-Hellman key exchange protocol.

8.2.2 The Diffie-Hellman Problem and the Discrete Logarithm Problem

The secrecy of the agreed shared key from the Diffie-Hellman key exchange protocol is exactly the problem of computing $g^{ab} \pmod{p}$ given g_a and g_b . This problem is called **the computational Diffie-Hellman problem** (CDHP).

Definition 8.1: (Computational Diffie-Hellman Problem, CDHP)

INPUT $\text{desc}(G)$: the description of an abelian group G ;
 $g \in G$: a group generator, i.e., $\langle g \rangle = G$;
 g^a, g^b for some integers $a, b < \#G$;

OUTPUT g^{ab} .

We have formulated the problem in a general form working in an abelian group. The Diffie-Hellman key exchange protocol in §8.2 uses a special case where $\text{desc}(G)$ is: $G = \mathbb{Z}_p^*$ for p being a prime number. We will frequently use this special case.

If the CDHP is easy, then $g^{ab} \pmod{p}$ can be computed from the values p, g, g_a, g_b , which are transmitted as part of the protocol messages. According to our assumptions on the ability of our adversary (see §2.2), these values are available to an adversary.

The CDHP lies, in turn, on the difficulty of **the discrete logarithm problem** (DLP).

Definition 8.2: (Discrete Logarithm Problem, DLP)

INPUT $\text{desc}(G)$: the description of an abelian group;
 $g \in G$: a group generator, i.e., $\langle g \rangle = G$;
 $h \in G$;

OUTPUT the unique integer $a < \#G$ such that $h = g^a$.

We denote the integer a by $\log_g h$.

The DLP looks similar to taking ordinary logarithms in the reals. But unlike logarithms in the reals where we only need approximated “solutions”, the DLP is defined in a discrete domain where a solution must be *exact*.

We have discussed in Chapter 4 that the security theory of modern public-key cryptography is established on a complexity-theoretic foundation. Upon this foundation, the security of a public-key cryptosystem is *conditional* on some assumptions that certain problems are intractable. The CDHP and the DLP are two assumed intractable problems. Intuitively we can immediately see that the difficulties of these problems depend on the size of the problems (here, it is the size of the group G), as well as on the choice of the parameters (here, it is the choice of the public parameter g and the private parameters a, b). Clearly, these problems need not be difficult for small instances. In a moment we will further see that these problems need not be difficult for poorly chosen instances. Thus, a precise description of the difficulty must formulate properly both the problem size and the choice of the instances. With the complexity-theoretic foundations that we have established in Chapter 4, we can now describe precisely the assumptions on the intractabilities of these two problems. The reader may review Chapter 4 to refresh several notions to be used in the following formulations (such as “ 1^k ”, “probabilistic polynomial time”, and “negligible quantity in k ”).

Assumption 8.1: (Computational Diffie-Hellman Assumption, CDHA) A CDHP solver is a PPT algorithm \mathcal{A} such that with an advantage $\epsilon > 0$:

$$\epsilon = \text{Prob} [g^{ab} \leftarrow \mathcal{A}(\text{desc}(G), g, g^a, g^b)]$$

where the input to \mathcal{A} is defined in Definition 8.1.

Let \mathcal{IG} be a group instance generator that on input 1^k , runs in time polynomial in k , and outputs (i) $\text{desc}(G)$ (the description of an abelian group G) with $|\#G| = k$, (ii) a group generator $g \in G$.

We say that \mathcal{IG} satisfies the computational Diffie-Hellman assumption (CDHA) if there exists no CDHP solver for $\mathcal{IG}(1^k)$ with advantage $\epsilon > 0$ non-negligible in k .

Assumption 8.2: (Discrete Logarithm Assumption, DLA) A DLP solver is a PPT algorithm \mathcal{A} such that with an advantage $\epsilon > 0$:

$$\epsilon = \text{Prob} [\log_g h \leftarrow \mathcal{A}(\text{desc}(G), g, h)]$$

where the input to \mathcal{A} is defined in Definition 8.2.

Let \mathcal{IG} be a group instance generator that on input 1^k , runs in time polynomial in k , and outputs (i) $\text{desc}(G)$ with G abelian and $|\#G| = k$, (ii) a group generator $g \in G$, (iii) $h \in G$.

We say that \mathcal{IG} satisfies the discrete logarithm assumption (DLA) if there exists no DLP solver for $\mathcal{IG}(1^k)$ with advantage $\epsilon > 0$ non-negligible in k .

In a nutshell, these two assumptions state that in abelian groups for all sufficiently large instances, there exists no efficient algorithm to solve the CDHP or the DLP. Few negligible exceptions maybe due to some weak instances.

Remark 8.1:

1. In Assumptions 8.1 and 8.2, the respective probability space should consider (i) the instance space, i.e., arbitrary groups and arbitrary group generators are sampled (the importance of this will be discussed in §8.2.3), and (ii) the space of the random operations in an efficient algorithm. The need for considering (ii) is because by “polynomial-time” or “efficient” algorithm we include randomized algorithms (see Definition 4.6).
2. The number k in the both formulations is called a **security parameter**. $\mathcal{IG}(1^k)$ is a random instance of group G which has the number of elements at the level of 2^k . Our study of the probabilistic prime generation in §4.7 has shown that $\text{desc}(G)$, i.e., the description of G , can be constructed in polynomial time in k . For $\text{desc}(G)$ being $G = \mathbb{Z}_p^*$ with p prime, it is now widely accepted that $k = 1024$ is a proper setting for a high confidence in security. This setting is a result of the time complexity expressed in (4.9.1) for the best known algorithm (the index-calculus algorithm) for solving DLP in \mathbb{Z}_p^* . In the expression (4.9.1), $\log N$ should be replaced with $k/\log_2 e$ (e is the base to the natural logarithm); for $k = 1024$; then the expression yields a quantity at the level of 2^{86} .
3. Holding of the DLA means that the function

$$g^x \pmod{p} : \mathbb{Z}_{p-1} \mapsto \mathbb{Z}_p^* \quad (8.2.1)$$

is one-way. Therefore, holding of the DLA implies the existence of one-way function. It is widely believed that the DLA should actually hold (a case under the belief $\mathcal{P} \neq \mathcal{NP}$, see §4.8), or the function in (8.2.1) should be one-way, or in other words, one-way function should exist.

4. It is not known to date whether or not the function in (8.2.1) is a trapdoor function (see Property 8.1 for the meaning of one-way trapdoor function). That is, no one knows how to embed a trapdoor information inside this function to enable an efficient inversion of the function (i.e., an efficient method to compute x from g^x using a trapdoor information). However, if the function uses a composite modulus (the function remains one-way), then the function becomes a trapdoor where the prime factorization of the modulus forms the trapdoor information. The reader is referred to [PG97], [OU98], [Pai99] for the technical details. \square

Notice that the availability of $a = \log_g g_1$ or $b = \log_g g_2$ will permit the calculation of

$$g^{ab} = g_1^b = g_2^a.$$

That is, an efficient algorithm which solves the DLP will lead to an efficient algorithm to solve the CDHP. Therefore if the DLA does not hold, then we cannot have

the CDHA. We say that the CDHP is weaker than the DLP, or equivalently, the CDHA is stronger an assumption than the DLA. The converse of this statement is an open question:

Can the DLA be true if the CDHA is false?

Maurer and Wolf have given a strong heuristic argument on the relation between these two problems which suggests that they are likely to be equivalent [MW99].

8.2.3 The Importance of Arbitrary Instances in Intractability Assumptions

We should emphasize the importance of arbitrary instances required in the DLA. Let us consider $G = \mathbb{Z}_p^*$ with p a k -bit prime and the problem of extracting a from $h \equiv g^a \pmod{p}$.

We know that a is an element in \mathbb{Z}_{p-1}^* . If $p-1 = q_1 q_2 \cdots q_\ell$ with each factor q_i being small (meaning, $q_i \leq \text{polynomial}(k)$ for $i = 1, 2, \dots, \ell$), then the discrete-logarithm-extraction problem can be turned into extracting $a_i \equiv a \pmod{q_i}$ from $h^{(p-1)/q_i} \pmod{p}$ but now a_i are small and can be extracted in time polynomial in k . After a_1, a_2, \dots, a_ℓ are extracted, a can be constructed by applying the Chinese remainder theorem (Theorem 6.7). This is the idea behind the polynomial-time algorithm of Pohlig and Hellman [PH78] for solving the DLP modulo p if $p-1$ has no large prime factor. Clearly, if every prime factor of $p-1$ is bounded by a polynomial in k , then the Pohlig-Hellman algorithm has a running time in polynomial in k .

A prime number p with $p-1$ containing no large prime factor is called a **smooth prime**. But sometimes we also say “ $p-1$ is smooth” with the same meaning. A standard way to avoid the smooth-prime weak case is to construct the prime p such that $p-1$ is divisible by another large prime p' . By Theorem 5.2(2), the cyclic group \mathbb{Z}_p^* contains the unique subgroup of order p' . If p' is made public, the users of the Diffie-Hellman key exchange protocol can make sure that the protocol is working in this large subgroup; all they need to do is to find an element $g \in \mathbb{Z}_p^*$ such that

$$g^{(p-1)/p'} \not\equiv 1 \pmod{p}.$$

This element g generates the group of the prime order p' . The Diffie-Hellman key exchange protocol should use (p, p', g) so generated as the common input. An accepted value for the size of the prime p' is at least 160 (binary bits), i.e., $p' > 2^{160}$. (Also see our discussion in §9.3.4.1.)

The DLP and the CDHP are also believed to be intractable in a general finite abelian group of a large order, such as a large prime-order subgroup of a finite field \mathbb{F}_q , or a group of points on an elliptic curve defined over a finite field. Thus, the Diffie-Hellman key exchange protocol will also work well in these groups.

There are several exponential-time algorithms which are very effective for extracting the discrete logarithm when the value to be extracted is known to be small. Since the problem of extracting small discrete logarithms has useful applications in some cryptographic protocols, we will describe these algorithms later when we apply them.

Research into the DLP is very active. Odlyzko provided a survey of the area which included an extensive literature on the topic [Od199].

8.3 The RSA Cryptosystem

The best known public-key cryptosystem is the RSA, named after its inventors Rivest, Shamir and Adleman [RSA78]. The RSA is the first practical realization of public-key cryptography based on the notion of one-way trapdoor function which Diffie and Hellman envisioned two years earlier.

The RSA cryptosystem is specified in Algorithm 8.1.

How Does it Work?

We now argue why the decryption procedure will actually return the same plaintext message that Bob has encrypted.

From the definition of the modulo operation (Definition 4.4), (8.3.1) means

$$ed = 1 + k\phi(N)$$

for some integer k . Therefore, the number returned from Alice's decryption procedure is

$$c^d \equiv m^{ed} \equiv m^{1+k\phi(N)} \equiv m \cdot m^{k\phi(N)} \pmod{N}. \quad (8.3.2)$$

We should notice that for $m < N$, it is almost always the case that $m \in \mathbb{Z}_N^*$ (the multiplicative group of integers relatively prime to N). In fact, the cases for $m \notin \mathbb{Z}_N^*$ are $m = up$ or $m = vq$ for some $u < q$ or $v < p$. In such cases, Bob can factor N by computing $\gcd(m, N)$. Assuming that the factoring is difficult (we will formulate the factorization problem and an assumption on its difficulty in a moment), we can assume that any message $m < N$ prepared by Bob satisfies $m \in \mathbb{Z}_N^*$.

For $m \in \mathbb{Z}_N^*$, by Lagrange's Theorem (Corollary 5.2), we have

$$\text{ord}_N(m) \mid \#\mathbb{Z}_N^* = \phi(N).$$

This is true for all $m \in \mathbb{Z}_N^*$. By the definition of the order of a group element (Definition 5.9), this means that for all $m \in \mathbb{Z}_N^*$

$$m^{\phi(N)} \equiv 1 \pmod{N}.$$

Algorithm 8.1: The RSA Cryptosystem**Key Setup**

To set up a user's key material, user Alice performs the following steps:

1. choose two random prime numbers p and q such that $|p| \approx |q|$;
(* This can be done by applying a Monte-Carlo prime number finding algorithm, e.g., Algorithm 4.7 *)
2. compute $N = pq$;
3. compute $\phi(N) = (p - 1)(q - 1)$;
4. choose a random integer $e < \phi(N)$ such that $\gcd(e, \phi(N)) = 1$, and compute the integer d such that

$$ed \equiv 1 \pmod{\phi(N)}; \quad (8.3.1)$$

(* Since $\gcd(e, \phi(N)) = 1$, congruence (8.3.1) does have a solution for d which can be found by applying the Extended Euclid Algorithm (Algorithm 4.2). *)

5. publicize (N, e) as her public key, safely destroy p , q and $\phi(N)$, and keep d as her private key.

Encryption

To send a confidential message $m < N$ to Alice, the sender Bob creates the ciphertext c as follows

$$c \stackrel{\text{def}}{=} m^e \pmod{N}.$$

(* Viewed by Bob, the plaintext message space is the set of all positive numbers less N . *)

Decryption

To decrypt the ciphertext c , Alice computes

$$m \stackrel{\text{def}}{=} c^d \pmod{N}.$$

Figure 8.3.

Obviously, this further implies

$$m^{k\phi(N)} \equiv 1 \pmod{N}$$

for any integer k . Thus, the number in (8.3.2) is, indeed, m .

Example 8.3: Let Alice set $N = 7 \times 13 = 91$ and $e = 5$. Then $\phi(N) = 6 = 72$. Applying Algorithm 4.2 (by inputting $(a, b) = (72, 5)$), Alice obtains the following equation

$$72 \times (-2) + 5 \times 29 = 1,$$

that is, $5 \times 29 \equiv 1 \pmod{72}$. Therefore Alice has computed 29 to be her private decryption exponent. She publicizes $(N, e) = (91, 5)$ as her public key material for the RSA cryptosystem.

Let Bob encrypt a plaintext $m = 3$. Bob performs encryption by computing

$$c = 3^5 = 243 \equiv 61 \pmod{91}.$$

The resultant ciphertext message is 61.

To decrypt the ciphertext message 61, Alice computes

$$61^{29} \equiv 3 \pmod{91}. \quad \square$$

8.3.1 Cryptanalysis Against Public-key Cryptosystems

It makes sense to say “Cryptosystem X is secure against attack Y but is insecure against attack Z”, that is, the security of a cryptosystem is defined by an attack. Let us now model three usual modes of active attacks. These modes of active attacks will be used in the analysis of the cryptosystems to be introduced in rest of this chapter.

Definition 8.3: (Active Attacks on Cryptosystems)

Chosen plaintext attack (CPA) *An attacker chooses plaintext messages and gets encryption assistance to obtain the corresponding ciphertext messages. The task for the attacker is to weaken the targeted cryptosystem using the obtained plaintext-ciphertext pairs.*

Chosen ciphertext attack (CCA) *An attacker chooses ciphertext messages and gets decryption assistance to obtain the corresponding plaintext messages. The task for the attacker is to weaken the targeted cryptosystem using the obtained plaintext-ciphertext pairs. The attacker is successful if he can retrieve some secret plaintext information from a “target ciphertext” for which and when no decryption assistance will be available.*

Adaptive chosen ciphertext attack (CCA2) *This is a CCA where the decryption assistance for the targeted cryptosystem will be available forever, except for the target ciphertext.*

We may imagine these attacks with the following scenarios. In a CPA, an attacker has in possession of an encryption box. In a CCA, an attacker has a conditional use of a decryption box: the box will be switched off before the target ciphertext is chosen by the attacker. In a CCA2, an attacker has in possession of a decryption box for use as long as he wishes, provided that he does not feed the target ciphertext to the decryption box; this only restriction on CCA2 is reasonable since otherwise there will be no difficult problem for the attacker to solve. In all cases, the attacker should not have in possession of the respective cryptographic keys.

CPA and CCA were originally proposed as active cryptanalysis models against secret-key cryptosystems where the objective of an attacker is to weaken the targeted cryptosystem using the plaintext-ciphertext message pairs he obtains from the attacks (see e.g., §1.2 of [Sti95]). They have been adopted to modelling active cryptanalysis on public-key cryptosystems. We should notice the following three points which are specific to public-key cryptosystems.

- The encryption assistance of a public-key cryptosystem is always available to anybody since given a public key anyone has the *complete* control of the encryption algorithm. In other words, CPA can always be mounted against a public-key cryptosystem. So, we can call an attack against a public-key cryptosystem CPA if the attack does not make use of any decryption assistance. Consequently and obviously, any public-key cryptosystem must resist CPA or else it is not a cryptosystem.
- In general, mathematics underlying most public-key cryptosystems has some nice properties of an algebraic structure underlying these cryptosystems, such as closure, associativity, and homomorphism, etc., (review Chapter 5 for these algebraic properties). An attacker may explore these nice properties and make up a ciphertext via some clever calculations. If the attacker is assisted with a decryption service, then his clever calculations may enable him to obtain some plaintext information, or even the private key of the targeted cryptosystem, which otherwise should be computationally infeasible for him to obtain. Therefore, public-key cryptosystems are particularly vulnerable to CCA and CCA2. We shall see that every public-key cryptosystem to be introduced in this chapter is vulnerable to CCA or CCA2. As a general principle, we have provided in Property 8.2.(ii) an advice that the owner of a public key should always be careful not to allow oneself to provide any decryption assistance to anybody. This advice must be followed for every public-key cryptosystem introduced in this chapter. In Chapter 13 we will introduce stronger public-key

cryptosystems which are purposely designed for the user to be relieved from being kept in an alert state.

- It seems that CCA is unrealistically restrictive since in applications it cannot be very clear when for a user to begin to stop providing decryption assistance for decryption requests. On the other hand, any public-key cryptosystem must be secure against CPA. We therefore consider CCA2 to be *the only* active attack against public-key cryptosystems which should be thwarted.

8.3.2 Insecurity of the Textbook RSA

Against CPA, the security of RSA lies on the difficulty of computing the e -th root of a ciphertext c modulo a composite integer n . This is the so-called **the RSA problem**.

Definition 8.4: (RSA Problem, RSAP)

INPUT $N = pq$ with p, q prime numbers;
 e : an integer such that $\gcd(e, (p-1)(q-1)) = 1$;
 $c \in \mathbb{Z}_N^*$;

OUTPUT the unique integer $m \in \mathbb{Z}_N^*$ satisfying $m^e \equiv c \pmod{N}$.

No difference from all underlying difficult problems for the security of public-key cryptosystems, the RSAP is also only assumed to be difficult under the properly chosen parameters.

Assumption 8.3: (RSA Assumption, RSAA) An RSAP solver is a PPT algorithm \mathcal{A} such that with an advantage $\epsilon > 0$:

$$\epsilon = \text{Prob}[m \leftarrow \mathcal{A}(N, e, m^e \pmod{N})],$$

where the input to \mathcal{A} is defined in Definition 8.4.

Let \mathcal{IG} be an RSA instance generator that on input 1^k , runs in time polynomial in k , and outputs (i) a $2k$ -bit modulus $N = pq$ where p and q are two distinct uniformly random primes, each of k bit in length, (ii) $e \in \mathbb{Z}_{(p-1)(q-1)}^*$.

We say that \mathcal{IG} satisfies the RSA assumption (RSAA) if there exists no RSAP solver for $\mathcal{IG}(1^k)$ with advantage $\epsilon > 0$ non-negligible in k .

Similar to our discussion in Remark 8.1(3), we know that holding of the RSAA implies the existence of one-way function. Also related to our discussion in Remark 8.1(4), the one-way function implied by the RSAA is a trapdoor function: the prime factorization of the modulus enables an efficient inversion procedure.

We should notice that the probability space in this assumption includes the instance space, the plaintext message space and the space of the random operations of a randomized algorithm for solving the RSAP.

We further notice that in the description of the RSAA, the (alleged) algorithm takes the encryption exponent e as part of its input. This precisely describes the target of the problem: breaking the RSAP under a given encryption exponent. There is a stronger version of the RSA assumption called **the strong RSA assumption** (strong RSAA) ([CS99]) in which the target of the problem is: find an encryption exponent $e > 1$ and break the RSAP under that exponent. We will later see some cryptographic protocols which are based on the strong RSAA.

It is clear that for public key (N, e) , if $m < N^{1/e}$ then encryption $c = m^e \pmod{N}$ will take no modulo reduction, and hence m can be found efficiently by extracting the e -th root in integers. This is one of the reasons why the case $e = 3$ should be avoided. In the case of $e = 3$, if one message m is encrypted in three different modulus: $c_i = m^3 \pmod{N_i}$ for $i = 1, 2, 3$, then because the moduli are pair-wise co-prime, the Chinese remainder theorem (Theorem 6.7) can be applied to construct $C = m^3 \pmod{N_1 N_2 N_3}$. Now because $m < N^{1/3}$, decryption of C is to extract the 3rd root in integers and can be efficiently done.

Coppersmith [Cop96] further extends this trivial case to a non-trivial one: for $m' = m + t$ where m is known and t is unknown but $t < N^{1/e}$, given $c = m'^e \pmod{N}$, t can be extracted efficiently. Because in applications, partially known plaintext is not uncommon (we shall see such an application in Chapter 14), it is now widely agreed that RSA encryption should avoid using very small encryption exponent. A widely accepted encryption exponent is $e = 2^{16} + 1 = 65537$ which is a prime number. This exponent makes encryption sufficiently efficient while refuting a small exponent attack.

RSA is also insecure if the decryption exponent d is small. Wiener discovers a method based on continued fraction expansion of e/N to find d if $d < N^{1/4}$ [Wie90]. This result has been improved to $d < N^{0.292}$ [BD99].

For random key instance and random message instance, by Definition 8.4 and Assumption 8.3, an efficient CPA against the RSA cryptosystem means precisely the false of the RSAA. Therefore we have

Theorem 8.1: *The RSA cryptosystem is “all-or-nothing” secure against CPA if and only if the RSAA holds.* \square

Here, the meaning of “all-or-nothing” secure is explained in Property 8.2.(i); while CPA means the attacker remains passive as stipulated in Property 8.2.(ii). A security of this quality is actually not a very useful for the reasons we now explain.

First, let us consider “all-or-nothing” security. The requirement of random message instance in the RSA assumption is usually not met in applications. In applications, a plaintext typically contains some non-secret partial information which is

known to an attack. The textbook RSA does not hide some partial information about a plaintext. For example, if a plaintext is known to be a number less than 1,000,000 (e.g., a secret bid), then given a ciphertext, an attacker can pinpoint the plaintext in less than 1,000,000 trial-and-error encryptions.

In general, for a plaintext $m(< N)$, with a non-negligible probability, only \sqrt{m} number of trials are needed to pinpoint m if \sqrt{m} size of memory is available. This is due to a clever observation made in [BJN00] which exploits the multiplicative property of RSA.

Example 8.4: Let $c = m^e \pmod{N}$ such that Malice knows $m < 2^\ell$. With non-negligible probability m is composite satisfying

$$m = m_1 \cdot m_2 \quad \text{and} \quad m_1, m_2 < 2^{\ell/2}. \quad (8.3.3)$$

With RSA's multiplicative property, we have

$$c = m_1^e \cdot m_2^e \pmod{N}. \quad (8.3.4)$$

Malice can build a database

$$1^e, 2^e, \dots, (2^{\ell/2})^e \pmod{N},$$

and searches through the database sequentially to see if $c/i^e \pmod{N}$ ($i = 1, 2, \dots$) is in the database. Because of (8.3.3) and (8.3.4), a match, denoted by

$$c/i^e \equiv j^e \pmod{N}$$

will be found before $2^{\ell/2}$ steps (modular exponentiations) of search. Since Malice knows the plaintexts i, j , he discovers $m = i \cdot j$. \square

This method for extracting plaintext from an RSA ciphertext is called a “meet-in-the-middle” attack. Now imagine the scenario of using an 1024-bit RSA to encrypt a DES key of 56 bits in the textbook style. If the DES key is a composite, then with a non-negligible probability the discovery of the key can be done using 2^{28} storage and 2^{28} modular exponentiations. This can be done without difficulty on a good personal computer, while direct searching for the key requires 2^{56} steps which can be quite prohibitive.

For cases the encrypted message having sizes ranging from 40-64 bits, the probabilities for the integer to be split to two similar size integers range from 18%-50% (see Table 1 of [BJN00]). Now we know that we *must not* use textbook RSA to encrypt a short key or a password which are around 2^{64} range.

The following example further shows the inadequacy of the CPA notion: the textbook RSA cryptosystem is miserably vulnerable to an active attacker.

Example 8.5: Let Malice be in a conditional control of Alice’s RSA decryption box. The condition is quite “reasonable”: if the decryption result of a ciphertext submitted by Malice is not meaningful (looks random), then Alice should return the plaintext to Malice. We say this condition to be “reasonable” for the following two reasons:

- i) “A random response for a random challenge” is quite a standard mode of operation in many cryptographic protocols, and hence, a user should follow such a “challenge-response” instruction. Indeed, *often* cryptographic protocols have been designed to allow this kind of conditional control of a decryption box by a protocol participant. For example, the Needham-Schroeder public-key authentication protocol (see Protocol 2.5) has exactly such a feature: Alice is instructed to decrypt a ciphertext from Bob.
- ii) Anyway, we would like to hope that a random-looking decryption result should not provide an attacker with any useful information.

Now if Malice wants to know the plaintext of a ciphertext $c \equiv m^e \pmod{N}$ which he had eavesdropped or intercepted from a previous confidential communication between Alice and someone else (not with him!). He picks a random number $r \in_U \mathbb{Z}_N^*$, computes $c' = r^e c \pmod{N}$ and sends his chosen ciphertext c' to Alice. The decryption result by Alice will be

$$c'^d \equiv rm \pmod{N}$$

which can be completely random for Alice since the multiplication of r is a permutation over \mathbb{Z}_N^* . So Alice returns the decryption result rm back to Malice. Alas, Malice has r and thereby can obtain m with a division modulo N . \square

In §15.4 we will see a concrete CCA on the the Needham-Schroeder public-key authentication protocol.

The conventional term to name Alice’s inadvertent decryption assistance as she has offered in Example 8.5 (or e.g., in an unfortunately implemented Needham-Schroeder public-key authentication protocol) is **oracle service**. The term “oracle” appears frequently in the literature of cryptography, usually for naming any unknown algorithm or method which is alleged to be able to solve a difficult problem. According to the RSAA, to decrypt a ciphertext encrypted under Alice’s public key is indeed a difficult problem should Alice not provide an inadvertent decryption service.

Examples 8.4 and 8.5 show that the textbook RSA is very weak. A systematic fix of these weakness will be given Chapters 13 and 14.

8.3.3 The Integer Factorization Problem

The difficulty of the RSA problem depends, in turn, on the difficulty of **the integer factorization problem**.

Definition 8.5: (Integer Factorization Problem, IFP)

INPUT N : odd composite integer with at least two distinct prime factors;

OUTPUT prime p such that $p|N$.

Again, the IFP is only assumed to be difficult under properly chosen parameters.

Assumption 8.4: (Integer Factorization Assumption, IFA) An integer factorizer is a PPT algorithm \mathcal{A} such that with an advantage $\epsilon > 0$:

$$\epsilon = \text{Prob}[\mathcal{A}(N) \text{ divides } N \text{ and } 1 < \mathcal{A}(N) < N]$$

where the input to \mathcal{A} in Definition 8.5.

Let \mathcal{IG} be an integer instance generator that on input 1^k , runs in time polynomial in k , and outputs a $2k$ -bit modulus $N = pq$ where p and q are each a k -bit uniformly random odd prime.

We say that \mathcal{IG} satisfies the integer factorization assumption (IFA) if there exists no integer factorizer for $\mathcal{IG}(1^k)$ with advantage $\epsilon > 0$ non-negligible in k .

Obviously, an algorithm which solves the IFP will solve the RSAP since Alice decrypts an RSA ciphertext exactly by first computing $d \equiv e^{-1} \pmod{(p-1)(q-1)}$, i.e., from the knowledge of the factorization of N . Similar to the relation between the CDHP and the DLP, the converse is also an open question: Can the IFA be true if the RSAA is false?

Similar to the situation of a smooth prime making a weak case for the DLP, a smooth prime factor of N will also make a weak case for the IFP. Such a weak case permits an efficient factorization algorithm known as Pollard's $p-1$ algorithm [Pol74]. The idea behind Pollard's $p-1$ algorithm is the following. Let p be a prime factor of N where the largest prime factor $p-1$ is bounded by $B = \text{Poly}(k)$ where $k = |N|$ and $\text{Poly}(k)$ is a polynomial in k (B is called "the smoothness bound of $p-1$ "). We can construct

$$A = \prod_{\text{primes } r < B} r^{\lfloor \log N / \log r \rfloor}.$$

By this construction, $p-1|A$, and so $a^A \equiv 1 \pmod{p}$ for any a with $\gcd(a, p) = 1$ due to Fermat's little theorem (Theorem 6.10). If $a \not\equiv 1 \pmod{q}$ for some other

prime factor q of N (this is easily satisfiable), then $a^A - 1 \pmod{N} = \ell p$ for some integer ℓ which is not a multiple of q . Thus, $\gcd(a^A - 1 \pmod{N}, N)$ must be a proper prime factor of N , and it must be p if $N = pq$. It remains to show that the size of A is a polynomial in k , and so computing $a^A \pmod{N}$ takes time in a polynomial in k .

By the prime number theorem (see e.g., page 28 of [Kra86]), there are no more than $B/\log B$ prime numbers less than B . So we have

$$A < B^{\lceil \log N \rceil \frac{B}{\log B}} < B^{k \frac{B}{\log B}}$$

that is,

$$|A| < kB \log 2 < k\text{Poly}(k).$$

Clearly, the right-hand side is a polynomial in k . Thus, $a^A \pmod{N}$ can be computed in a number of multiplications modulo N (using Algorithm 4.3) where the number is a polynomial in k . Notice that the explicit construction of A is unnecessary; $a^A \pmod{N}$ can be computed by computing $a^{r^{\lceil \log N / \log r \rceil}} \pmod{N}$ for all prime $r < B$.

It is very easy to construct an RSA modulus $N = pq$ such that the smoothness bound of $p - 1$ and that of $q - 1$ are non-polynomially (in $|N|$) small, and so the modulus would resist this factoring method. One may start by finding large prime p' such that $p = 2p' + 1$ is also a prime; and large prime q' such that $q = 2q' + 1$ is also prime. A prime of this format is called a **safe prime** and an RSA modulus with two safe prime factors is called a **safe-prime RSA modulus**. There is a debate on the need of using safe-prime RSA modulus for the RSA cryptosystems. The point against the use (see e.g., [Sil97]) is that an RSA modulus should be as random as possible, and that for a randomly chosen prime p , probability that $p - 1$ has a large prime factor is overwhelming. We shall however see later that many cryptographic protocols based on the IFP require to use safe-prime RSA moduli.

It is also well-known that partial information of a prime factor of N can produce efficient algorithms to factor N . For instance, for $N = pq$ with p and q primes of roughly equal size, knowledge of up to half the bits of p will suffice to factor N in polynomial time in the size of N , see e.g., [Cop96].

If not using any *a-priori* information about the prime factors of the input composite, then the current best factorization algorithm is the number field sieve method which has time complexity expressed in (4.9.1). Thus, similar to the setting of the security parameter for the DLP modulo a prime, 1024 is a proper setting for the size of an RSA modulus for a high confidence in security.

Recently, the number field sieve method demonstrated an effectiveness of massive parallelization: in early 2000, a coalition of 9,000 workstations worldwide ran a parallel algorithm and factored a 512-bit RSA modulus (the RSA-512 Challenge) after more than four months of running of the parallel algorithm [CDL⁺00].

Research into integer factorization is very active and it is impossible to rule out a decisive advance. Boneh provided a survey on the RSAP [Bon99]. Discussions on the progress in the IFP area with a literature review can be found in Chapter 3 of [MvOV97].

8.4 The Rabin Cryptosystem

Rabin developed a public-key cryptosystem based on the difficulty of computing a square root modulo a composite integer [Rab79]. Rabin's work has a theoretic importance; it provided the first provable security for public-key cryptosystems: the security of the Rabin cryptosystem is exactly the intractability of the IFP. (Recall our discussion for the case of the RSA: it is not known if the RSAP is equivalent to the IFP.) The encryption algorithm in the Rabin cryptosystem is also extremely efficient and hence is very suitable in certain applications such as encryption performed by hand-held devices. The Rabin cryptosystem is specified in Algorithm 8.2.

How Does it Work?

We know from elementary mathematics that the general solution to this equation can be written as

$$m \equiv \frac{-b + \sqrt{\Delta_c}}{2} \pmod{N}, \quad (8.4.3)$$

where

$$\Delta_c \stackrel{\text{def}}{=} b^2 + 4c \pmod{N}. \quad (8.4.4)$$

Since c is formed using $m \in \mathbb{Z}_N^*$, of course the quadratic equation (8.4.2) has solutions in \mathbb{Z}_N^* , and these solutions include m sent from Bob. This implies that Δ_c must be a quadratic residue modulo N , i.e., an element in QR_N .

The decryption computation involves to compute square roots modulo N . From our study of the square-rooting problem in §6.5 we know that the difficulty of this problem is computationally equivalent to that of factoring N (Corollary 6.3). Therefore, the only person who can compute (8.4.3) is Alice since only she knows the factorization of N . Alice can compute $\sqrt{\Delta_c}$ using Algorithm 6.5. In §6.5 we also know that for each ciphertext c sent by Bob, there are four distinct values for $\sqrt{\Delta_c}$ and hence there are four different decryption results. We assume that, in applications, a plaintext message should contain *redundant information* to allow Alice to recognize the correct plaintext from the four decryption results. We will provide in §9.3.1.1 the meaning for “recognizable redundancy” and a common method for a message to be formatted to contain recognizable redundancy.

We notice that if N is a so-called **Blum integer**, that is, $N = pq$ with $p \equiv q \equiv 3 \pmod{4}$, then it is easier to compute square roots modulo N (by computing

Algorithm 8.2: The Rabin Cryptosystem**Key Setup**

To set up a user's key material, user Alice performs the following steps:

1. choose two random prime numbers p and q such that $|p| \approx |q|$
 (* Using a Monte-Carlo prime number finding algorithm, e.g., Algorithm 4.7. *)
2. compute $N = pq$;
3. pick a random integer $b \in_U \mathbb{Z}_N^*$;
4. publicize (N, b) as her public key material, and keep (p, q) as her private key.

Encryption

To send a confidential message $m \in \mathbb{Z}_N^*$ to Alice, the sender Bob creates ciphertext c as follows:

$$c \stackrel{\text{def}}{=} m(m + b) \pmod{N}. \quad (8.4.1)$$

Decryption

To decrypt the ciphertext c , Alice solves the quadratic equation

$$m^2 + bm - c \equiv 0 \pmod{N} \quad (8.4.2)$$

for $m < N$.

Figure 8.4.

square roots modulo p and q using Algorithm 6.3, Case $p \equiv 3, 7 \pmod{8}$ and then constructing the square roots by applying the Chinese remainder theorem). Therefore, in practice, the public modulus in the Rabin cryptosystem is set to be a Blum integer.

The Rabin encryption algorithm only involves one multiplication and one addition and hence is much faster than the RSA encryption.

Example 8.6: Let Alice set $N = 11 \times 19 = 209$ and $b = 183$. She publicize $(N, b) = (209, 183)$ as her public key material for the Rabin cryptosystem.

Let Bob encrypt a plaintext message $m = 31$. Bob performs encryption by applying (8.4.1):

$$c = 31 \times (31 + 183) \equiv 155 \pmod{209}.$$

The resultant ciphertext is 155.

To decrypt the ciphertext 155, Alice first computes Δ_c using (8.4.4):

$$\Delta_c = b^2 + 4c = 183^2 + 4 \times 155 \equiv 42 \pmod{209}.$$

Now applying Algorithm 6.5, Alice finds the four square roots of 42 modulo 209 to be 135, 173, 36, 74. Finally, she can apply equation 8.4.3 and obtains the four decryption results: 185, 204, 31, 50. In real application of the Rabin cryptosystem, the plaintext should contain additional information for the receiver to pinpoint the correct decryption result. \square

8.4.1 Insecurity of the Textbook Rabin

The textbook Rabin has the following variation:

$$c = m^2 \pmod{N}.$$

It is clear that the insecurity of the textbook RSA such as small encryption exponent and “meet-in-the-middle” attack in Example 8.4 will also apply to this variation of the textbook Rabin.

Moreover, we have a more devastating attack against the textbook Rabin.

Theorem 8.2:

- I) *The Rabin cryptosystem is provably “all-or-nothing” secure against CPA if and only if factoring of RSA moduli is hard.*
- II) *The Rabin cryptosystem is completely insecure if it is attacked under CCA.*

Proof (I) Because the specified decryption procedure of the Rabin cryptosystem uses the factorization of an RSA modulus, the security of the Rabin encryption therefore implies the intractability of factoring of RSA moduli. Thus for (I), we only need to prove the statement for the other direction: the intractability of factoring of RSA moduli implies the security of the Rabin cryptosystem.

Suppose that there exists an oracle \mathcal{O} which breaks the Rabin cryptosystem with a non-negligible advantage ϵ , i.e.,

$$\text{Prob} \left[\mathcal{O}(c, N) = \frac{-b + \sqrt{\Delta_c}}{2} \pmod{N} \mid c \in_U \mathbb{Z}_N^* \right] \geq \epsilon.$$

We choose a random message m , computes $c = m(m+b) \pmod{N}$ and call $\mathcal{O}(c, N)$ which will return $m' \equiv \frac{-b+\sqrt{\Delta_c'}}{2} \pmod{N}$ with advantage ϵ . Here $\sqrt{\Delta_c'}$ denotes any one of the four square roots of Δ_c . By Theorem 6.17 we know with probability $1/2$:

$$m' + \frac{b}{2} \equiv \frac{\sqrt{\Delta_c'}}{2} \not\equiv \pm \frac{\sqrt{\Delta_c}}{2} \equiv \pm(m + \frac{b}{2}) \pmod{N}.$$

But because

$$(m' + \frac{b}{2})^2 \equiv \frac{\Delta_c}{4} \equiv [\pm(m + \frac{b}{2})]^2 \pmod{N},$$

so as shown in Theorem 6.17,

$$\gcd(m' + \frac{b}{2} \pm (m + \frac{b}{2}), N) = p \text{ or } q. \quad (8.4.5)$$

That is, N can be factored with the non-negligible advantage $\epsilon/2$. This contradicts the assumed intractability of factoring of RSA moduli (the IFA). We have thus shown (I).

Statement (II) holds trivially true if an attacker can obtain a decryption assistance: the decryption assistance plays exactly the role of the oracle used in the proof of statement (I)! Since the attacker will generate (choose) ciphertext for the decryption oracle to decrypt, such an attack is CCA. \square

Theorem 8.2 tells us two opposite things. First, the Rabin cryptosystem is provably secure, in an “all-or-nothing” sense in Property 8.2.(i), with respect to the difficulty of factorization: if the IFP is indeed intractable, then the alleged oracle \mathcal{O} in the proof of (I) should not exist.

Secondly, it is now clear that, in the Rabin cryptosystem, one should *never* allow oneself to be used as a decryption oracle: CCA is devastating against the Rabin cryptosystem: the consequence of such an attack is not merely finding *some* plaintext information (as in the case of CCA2 against the RSA cryptosystem as illustrated in Example 8.5), it is the discovery of the private key of the key owner, and hence the attacker will be able to read *all* confidential messages encrypted under the targeted public key.

Example 8.7: In Example 8.6 for the Rabin cryptosystem we have seen that for public key material $(N, b) = (209, 183)$, the four decryption results of the ciphertext 31 are 185, 204, 31, 50.

If these numbers are made available to a non-owner of the public key, e.g., via a CCA, they can be used to factor the modulus 209. For example, applying (8.4.5):

$$\gcd(204 - 185, 209) = 19,$$

or

$$\gcd((31 + 183/2) + (50 + 183/2), 209) = \gcd(264, 209) = 11. \quad \square$$

Finally we should notice that since the modulus of the Rabin cryptosystem is the same as that of the RSA cryptosystem, the cautionary measures that we have discussed for the proper choice of the RSA modulus apply to the Rabin modulus.

8.5 The ElGamal Cryptosystem

In 1985, ElGamal developed a public-key cryptosystem based on the CDHP [ELG85]. The cryptosystem is a successful turning of the Diffie-Hellman one-way trapdoor function into a public-key encryption scheme. It immediately attracted great interests in both research and applications which have been remaining high to this day (we will see further development of the cryptosystem in Chapters 12 and 13). One reason for this is that ElGamal's work enabled the use of the widely believed reliable intractability for underlying the security of public-key cryptosystems: the CDHP, which is widely believed to be as hard as the DLP and the latter is considered to be a good alternative to the other widely accepted reliable intractability: the IFP (the basis for the RSA and Rabin).

The ElGamal cryptosystem is specified in Algorithm 8.3.

How Does it Work?

Since

$$c_1^x \equiv (g^k)^x \equiv (g^x)^k \equiv y^k \equiv c_2/m \pmod{p},$$

the decryption calculation (8.5.2) does indeed restore the plaintext m .

The division in the decryption step (8.5.2) needs to use the extended Euclid algorithm (Algorithm 4.2) which is generally more costly than a multiplication. However Alice may avoid the division by computing

$$m \stackrel{\text{def}}{=} c_2 c_1^{-x} \pmod{p}.$$

One may verify that this decryption method works, but notice that $-x$ here means $p - 1 - x$.

Example 8.8: From Example 8.1 we know that 3 is a primitive root modulo 43. Let Alice choose her private key to be 7. She computes her public key as

$$37 \equiv 3^7 \pmod{43}.$$

Alice publicize her public key material $(p, g, y) = (43, 3, 37)$.

Let Bob encrypt a plaintext message $m = 14$. Bob picks a random exponent 26 and computes

$$c_1 = 15 \equiv 3^{26} \pmod{43}, \quad c_2 = 31 \equiv 37^{26} \times 14 \pmod{43}.$$

Algorithm 8.3: The ElGamal Cryptosystem**Key Setup**

To set up a user's key material, user Alice performs the following steps:

1. choose a random prime number p ;
2. compute a random multiplicative generator element g of \mathbb{F}_p^* ;
3. pick a random number $x \in_U \mathbb{Z}_{p-1}$ as her private key;
4. compute her public key by

$$y \stackrel{\text{def}}{=} g^x \pmod{p};$$

5. publicize (p, g, y) as her public key, and keep x as her private key.

(* Similar to the case of the Diffie-Hellman key exchange protocol, a system-wide users may share the common public parameters (p, g) . *)

Encryption

To send a confidential message $m < p$ to Alice, the sender Bob picks $k \in_U \mathbb{Z}_{p-1}$ and computes ciphertext pair (c_1, c_2) as follows:

$$\begin{cases} c_1 \stackrel{\text{def}}{=} g^k \pmod{p}, \\ c_2 \stackrel{\text{def}}{=} y^k m \pmod{p}. \end{cases} \quad (8.5.1)$$

Decryption

To decrypt ciphertext (c_1, c_2) , Alice computes

$$m \stackrel{\text{def}}{=} c_2 / c_1^x \pmod{p}. \quad (8.5.2)$$

Figure 8.5.

The resultant ciphertext message pair is $(15, 31)$.

To decrypt the ciphertext message $(15, 31)$, Alice computes

$$14 = 31/36 \equiv 31/15^7 \pmod{43}.$$

Division requires application of Algorithm 4.2. But Alice can avoid it by computing:

$$14 = 31 \times 15^{42-7} \equiv 31 \times 6 \pmod{43}. \quad \square$$

8.5.1 Insecurity of the Textbook ElGamal

The encryption algorithm (8.5.1) of the ElGamal cryptosystem is probabilistic: it uses a random input $k \in_U \mathbb{Z}_{p-1}$. Suppose that Alice's private key x is relatively prime to $p-1$; then by Theorem 5.2(3), her public key $y \equiv g^x \pmod{p}$ remains to be a generator of the group \mathbb{Z}_p^* (since g is), and thereby $y^k \pmod{p}$ will range over \mathbb{Z}_p^* when k ranges over \mathbb{Z}_{p-1} . Consequently, for any plaintext message $m \in \mathbb{Z}_p^*$, $c_2 \equiv y^k m \pmod{p}$ will range over \mathbb{Z}_p^* when k ranges over \mathbb{Z}_{p-1} (Theorem 6.6). Thus, we have $c_2 \in_U \mathbb{Z}_p^*$ for $k \in_U \mathbb{Z}_{p-1}$. This means that the ElGamal encryption achieves the distribution of the plaintext message *uniformly* over the entire message space. This is the ideal semantic property for a cipher^a.

However, we should not be too optimistic! The ciphertext of the ElGamal encryption is not just the single block c_2 , but the pair (c_1, c_2) , and these two blocks *are* statistically *related*. Therefore, like all other public-key cryptosystems, the security of the ElGamal cryptosystem is conditional under an intractability assumption.

Theorem 8.3: *For a plaintext message uniformly distributed in the plaintext message space, the ElGamal cryptosystem is “all-or-nothing” secure against CPA if and only if the CDHP is hard.*

Proof (\Rightarrow) We need to show that if the ElGamal cryptosystem is secure, then the CDHA holds.

Suppose on the contrary the CDHA does not hold. Then given any ciphertext $(c_1, c_2) \equiv (g^k, y^k m) \pmod{p}$ constructed under the public key $y \equiv g^x \pmod{p}$, a CDH oracle will compute from (p, g, g^x, g^k) to $g^{xk} \equiv y^k \pmod{p}$ with a non-negligible advantage. Then $c_2/y^k \rightarrow m \pmod{p}$ with the same advantage. This contradicts the assumed security of the ElGamal cryptosystem.

(\Leftarrow) We now need to show that if the CDHA holds, then there exists no efficient algorithm that can recover plaintext message encrypted in an ElGamal ciphertext with non-negligible advantage.

Suppose on the contrary there exists an efficient oracle \mathcal{O} against the ElGamal cryptosystem, that is, given any public key (p, g, y) and ciphertext (c_1, c_2)

$$\mathcal{O}(p, g, y, c_1, c_2) \rightarrow m$$

with a non-negligible advantage δ where m satisfies

$$c_2/m \equiv g^{(\log_g y \log_g c_1)} \pmod{p}.$$

^aIn Chapter 13 we shall show that for the ElGamal cryptosystems to be “semantically secure”, g must only generate the group QR_p (the quadratic residues modulo p), and the plaintext messages should also be encoded into elements in QR_p .

Then for an arbitrary CDH instance (p, g, g_1, g_2) , we set (p, g, g_1) as public key and set (g_2, c_2) as ciphertext pair for a random $c_2 \in \mathbb{Z}_p^*$. Then with the advantage δ

$$\mathcal{O}(p, g, g_1, g_2, c_2) \rightarrow m$$

where m satisfies

$$c_2/m \equiv g^{(\log_g g_1 \log_g g_2)} \pmod{p}.$$

This contradicts the holding of the CDHA. \square

Since the CPA security of the ElGamal cryptosystem is equivalent to the CDHP, our discussions for the CDHP and DLP in §8.2.2), such as the cautionary considerations on the settings of the public-key parameters, all apply to the ElGamal cryptosystem. As in the Diffie-Hellman key exchange protocol, the ElGamal cryptosystem can also work in a large prime-order subgroup of \mathbb{Z}_p^* or in the finite field \mathbb{F}_q , or in a large group of points on an elliptic curve defined over a finite field.

The need for random distribution of the plaintext message in Theorem 8.3 is necessary. For small plaintext messages, a “meet-in-the-middle” attack (see Example ??) can also be applied to the textbook ElGamal. Although due to the probabilistic property of the encryption, such attacks are more complex. The interested reader is referred to [BJN00].

Finally, we provide an example on ElGamal’s vulnerability to active attack.

Example 8.9: Let Malice be in a conditional control of Alice’s ElGamal decryption box. Same as in Example 8.5, the condition is a “reasonable” one in that if a decryption of a ciphertext submitted by Malice results in a message which is not meaningful (looks random), then Alice should return the decryption result to Malice.

Let Malice have a ciphertext $(c_1, c_2) \equiv (g^k, y^k m) \pmod{p}$ which he has eavesdropped or intercepted from a previous confidential communication between Alice and someone else (not with Malice!). If Malice wants to know the corresponding plaintext. He picks a random number $r \in_U \mathbb{Z}_p^*$, computes $c'_2 = rc_2 \pmod{p}$ and sends his chosen ciphertext (c_1, c'_2) to Alice. The decryption result by Alice will be

$$rm \pmod{p}$$

which, viewed by Alice, is completely random since the multiplication of r is a permutation over \mathbb{Z}_p^* . So Alice returns the decryption result rm back to Malice. Alas, Malice has r and thereby can obtain m with a division modulo p . \square

8.6 Need for Stronger Security Notions for Public-key Cryptosystems

We have introduced several basic public-key cryptosystems. These basic schemes can be viewed as direct applications of various one-way trapdoor functions. (The

meaning of one-way trapdoor functions has been given in Property 8.1.) Let us call these basic public-key cryptosystems “textbook schemes” because most textbooks in cryptography introduce these schemes rather than their “fit-for-application” counterparts.

Now it is time to provide a summary on the insecurity of these textbook schemes. We should provide a brief discussion here on two aspects of vulnerabilities that a textbook public-key cryptosystem has.

First, as having stated in Property 8.2.(i), within the scope of this chapter we have only considered a very weak notion of security: secrecy in an “all-or-nothing” sense. In most applications of public-key cryptosystems, such a weak notion of secrecy is far from good enough and is also not very useful. In many applications plaintext messages contain *a-priori* information known to an attacker. For example, if a cipher encrypts a vote, then the *a-priori* information can be “YES” or “NO”, or a handful names of the candidates; thus, regardless of how strong a trapdoor function is, an attacker only needs several trial-and-error to pinpoint the correct plaintext. In some other applications, some partial *a-priori* information about the plaintext will provide an attacker an unentitled advantage (we will see such an attack in §13.3.2). In general, a textbook encryption algorithm does not hide such partial information very well. Thus, stronger public-key cryptosystems secure for hiding any *a-priori* information about the plaintext are needed.

Secondly, as having stated in Property 8.2.(ii), within the scope of this chapter we have only considered a very weak mode of attack: “passive attacker”. However, for each textbook scheme introduced in this chapter we have demonstrated an active attack on it (Examples 8.5, 8.7, 8.9). In such an attack, the attacker can prepare a cleverly calculated ciphertext message and submit it to a key owner for an oracle service in the decryption mode (CCA or CCA2). Our attacks show that textbook public-key cryptosystems are in general vulnerable to CCA or CCA2. Although we have provided an advice as a general principle for a user to anticipate an active attacker: a public key owner should always be vigilant not to provide a decryption service, however, considering that it will be impractical to require an innocent user to be always in an alert state, an advice on not to respond to a decryption request is not a correct strategy against an active attacker.

Public-key cryptosystems with stronger notions of security with respect to these two aspects have been proposed by various authors. In Chapter 13 we shall study the course of establishing various stronger confidentiality notions and how to achieve **formally provable security**. In Chapter 14 we shall introduce two “fit-for-application” public-key cryptosystems which are provably secure.

8.7 Combination of Asymmetric and Symmetric Cryptography

Public-key cryptography solves the key distribution problem very nicely. However, in general, public-key cryptographic functions operate in very large algebraic

structures which mean expensive algebraic operations. Comparatively, symmetric cryptographic functions are in general much more efficient. Considering the AES for example, it works in a field of 256 elements; the basic operations such as multiplication and inversion can be conducted by “table lookup” method (review §7.6.4) which is extremely efficient. In general, public-key cryptosystems are comparatively much more computationally intensive than their symmetric-key counterparts.

In applications, in particular in those which need encryption of bulk data, it is now a standard approach that encryption uses a **hybrid scheme**. In such a scheme, public-key cryptography is used to encrypt a so called **ephemeral key** for keying a symmetric cryptosystem; this establishes the shared ephemeral key between a sender and a receiver; the bulk data payload is then encrypted under the shared ephemeral key using a symmetric cryptosystem. Such a combined scheme achieves the best out of the two kinds of cryptosystems: the ease of key distribution from public-key cryptosystems and the efficiency from the symmetric cryptosystems.

A widely used combination of public-key and symmetric-key cryptosystems in cryptographic protocols is a so-called **digital-envelope** technique. This is the combination of the RSA cryptosystem with a symmetric-key cryptosystem such as the DES, the triple-DES or the AES. This common combination (RSA + DES or RSA + triple DES) is the basic mode for the **secure sockets layer protocol** ([Hic95] we will introduce it in Chapter 11) which has been used in popular Web browsers such as Netscape and Internet Explorer and Web servers. In the SSL protocol, the initiator of the protocol (let it be Alice, usually in the position of a Web client) will first download the public-key material of the other communication party (let it be Bob, usually in the position of a Web server); then Alice (in fact, her web-browser software) will generate a random session key, encrypts (“envelopes”) the session key using Bob’s public key and send the “envelope” to Bob. After Bob (in fact, his web-server software) has decrypted the “envelope” and retrieved the session key, the two parties can then use the session key to key a symmetric encryption scheme for their subsequent confidential communications.

In the context of protocols, the simple hybrid encryption scheme is conceptually very simple. But it has two limitations. First, the scheme uses a session key which is created by one party (the message sender or the protocol initiator); the other party (the message receiver or the protocol responder) will have to completely rely her/his own security on the sender’s or the initiator’s competence (or honesty) in key generation. This may not be desirable in some circumstances, for instance, in the SSL protocol’s client-server setting where the client is the sender and is implemented in software which is notoriously weak in generation of randomness.

The second limitation of the simple hybrid encryption scheme is due to its non-evanescent property. In hybrid encryption scheme, an eavesdropper who can coerce the receiver into revealing her/his private key can then recover the full `Payload_Message`. This weakness is often referred to as lack of “forward secrecy property”. The forward secrecy property means an impossibility for an eavesdropper to

recover the plaintext message in a future time using the ciphertext messages sent in the past, either by means of cryptanalysis or even by means of *coercion*.

These two limitations can be overcome if the public-key cryptographic part of a hybrid encryption scheme uses the Diffie-Hellman key exchange protocol.

Let us first look at how the first limitation disappears if a KEM-DEM scheme uses the Diffie-Hellman key exchange protocol. In the Diffie-Hellman key exchange protocol run between Alice and Bob, the shared secret g^{ab} contains randomness input from the both parties: Alice's contribution is from a and Bob's, from b . Given that g generates a prime-order group and that the protocol messages satisfy $g^a \neq 1$ and $g^b \neq 1$ (see the “cautionary details” that we have provided in §8.2), Alice (respectively, Bob) can be sure that the shared secret session key derived from g^{ab} will be random as long as she (respectively, he) has used a random exponent. This is because the mappings $g^b \mapsto (g^b)^a$ and $g^a \mapsto (g^a)^b$ are permutations in the group in question and thereby a uniform exponent (less than the group order) will cause g^a (respectively, g^b) to be mapped to a uniform group element g^{ab} .

Secondly, let us look at how the second limitation is overcome. We note that a hybrid encryption scheme using the Diffie-Hellman key exchange protocol has the forward secrecy property if Alice and Bob run the key exchange protocol in a cautionary manner which we have recommended in §8.2, and if they also properly process the subsequent session communications. To run the Diffie-Hellman key exchange protocol in a cautionary manner, Alice and Bob should exchange the session key g^{ab} and then erase the exponents a and b upon termination of the protocol. To properly process the subsequent session communications, Alice and Bob should destroy the session key after the session ends and should properly dispose the plaintext messages they have communicated. If they follow these rather *standard* procedures, then obviously coercion will not enable an eavesdropper to find out the plaintext messages that Alice and Bob have communicated. Cryptanalysis won't do the job for the eavesdropper either since the forward secrecy property (of the Diffie-Hellman key exchange protocol) is simply due to the difficulty of the CDHP (see §8.2.2).

Finally we point out that a hybrid encryption scheme can be designed to have a **provable security** under a very strong notion of confidentiality. In Chapter 14 we shall conduct an overview of a series of such schemes.

8.8 Bit Security of the Basic Public-key Cryptographic Functions

Although we have seen from several examples that the basic public-key cryptographic functions in general do not hide partial information *about* plaintext messages very well, however, in general these functions do hide any single bit *of* plaintext messages in a very strong sense. Let us now examine the **bit security** of these functions.

8.8.1 The RSA Bit

If an RSA ciphertext encrypts a message which contains no *a-priori* guessable information (for example, when a message is a uniformly random number in \mathbb{Z}_N^*), then it is known that the problem of extracting from a ciphertext one bit of the plaintext is as hard as extracting the whole block of the plaintext [GMT82], [CG85], [Cho85]. Without loss of generality, “one bit of the plaintext” can be the least significant bit, i.e., the parity bit, of the plaintext message. What we are trying to say here is the following statement.

Theorem 8.4: *Let N be an RSA modulus. The following two problems are equally hard (or equally easy):*

- I) *given the RSA encryption of a message, retrieve the message;*
- II) *given the RSA encryption of a message, retrieve the least significant bit of the message.*

If one can solve (I) then obviously one can solve (II). The converse seems not so straightforward. At a first glance, one may think that these two problems can hardly be computationally equivalent: (I) is a computational problem while for uniformly random plaintext message, (II) is a decisional problem and sheer guessing will entitle one to solve half the instances.

Nevertheless, if one can have in possession of an oracle which can answer (II) *reliably*, then one can indeed solve (I) by calling this oracle $\log_2 N$ times, and we shall show such a method. Since $\log_2 N$ is the size of N , such a method “reduces” (I) to (II) in polynomial time in the size of the input, and is therefore called a **polynomial time reduction**. Consequently, (I) can be solved in time polynomial in the size of the input, *on top of the time* for the oracle to solve (II). We view these two problems to have the same time complexity because we do not differentiate complexities which are different up to a polynomial.

Now let us describe a polynomial reduction method from (I) to (II). Let us call the oracle solving (II) “RSA parity oracle” and denote it by PO_N , namely,

$$PO_N(m^e \pmod{N}) \rightarrow m \pmod{2}.$$

In our proof of Theorem 8.4, we denote by $x \in (a, b)$ an integer x in the open interval (a, b) where a and/or b may or may not be integer. Since x is an integer, $x \in (a, b)$ implies that x is in the closed interval $[a], [b]$.

The crux of the proof is a **binary search** technique which is enabled by the following observation.

Lemma 8.1: *Let N be an odd integer and $x \in (0, N)$. Then $2x \pmod{N}$ is even if and only if $x \pmod{N} \in (0, \frac{N}{2})$.*

Proof For all $x \in (0, \frac{N}{2})$, multiplication $2x \pmod{N}$ takes no modulo operation and therefore the result is $2x$ and is an even number in $(0, N)$. Conversely, if $2x \pmod{N}$ is even then it can be divided by 2 and the division takes no modulo operation. Consequently $x \in (0, \frac{N}{2})$. \square

Since all $x \in (0, \frac{N}{2})$ occupy exactly half the integers in $(0, N)$, Lemma 8.1 also says that $2x \pmod{N}$ is odd if and only if $x \in (\frac{N}{2}, N)$.

Now let us prove Theorem 8.4.

Proof (of Theorem 8.4) We only need to show (II) \Rightarrow (I). The proof is constructive. We want to find m from an RSA ciphertext $c = m^e \pmod{N}$, and we do so by constructing a binary-search procedure which makes use of a reliable PO_N .

In the first step of the procedure, we ask PO_N by feeding it $2^e c \pmod{N}$. Noticing $2^e c \equiv (2m)^e \pmod{N}$, so from $PO_N(2^e c)$ we can deduce from Lemma 8.1 whether $m \in (0, \frac{N}{2})$ or $m \in (\frac{N}{2}, N)$. We therefore obtain the interval which contains m . Let us call this interval “current interval” which is of length under $\frac{N}{2}$.

In the second step, suppose the current interval being $(\frac{N}{2}, N)$, and so $N < 2m < 2N$. We feed $4^e c \equiv (4m)^e \pmod{N}$ to PO_N . If $PO_N(4^e c) = 0$, then by Lemma 8.1 we know $2m \pmod{N} \in (0, \frac{N}{2})$. But remember $2m < 2N$; so for $2m \pmod{N} < \frac{N}{2}$ it is only possible for $2m < 2N - \frac{N}{2} = \frac{3N}{2}$, and thereby $m < \frac{3N}{4}$. Combining this result with the result from the first step, we reach $m \in (\frac{N}{2}, \frac{3N}{4})$. Thus, after probing PO_N the second question, the length of the current interval is under $\frac{N}{4}$, i.e., it further shrinks by half the length of the current interval. The reader may check the other three cases analogously (i.e., the current interval being $(\frac{N}{2}, N)$ with $PO_N(4^e c) = 1$, and the current interval being $(0, \frac{N}{2})$ with the two answers of PO_N).

Algorithm 8.4 summarizes the general description of this binary search algorithm.

Clearly, after $i = \lfloor \log_2 N \rfloor + 1$ steps of search, the current interval will shrink to the length 1, i.e., it contains m solely. The search terminates with output m . To this end we have proven Theorem 8.4. \square

Example 8.10: For RSA public key $(N, e) = (15, 3)$, ciphertext $c = 13$, let us ask PO_N 4 questions and pinpoint the secret plaintext m . We feed PO_N the following random-looking ciphertext queries:

$$(2^3 \times 13, 4^3 \times 13, 8^3 \times 13, 16^3 \times 13) \equiv (14, 7, 11, 13) \pmod{15}.$$

PO_N answers: 0, 1, 1, 1. From these answers we deduce:

$$\text{1st answer 0} \Rightarrow m \in (0, 15 - \frac{15}{2}) = (0, \frac{15}{2}), \text{ i.e., } m \in [1, 7];$$

$$\text{2nd answer 1} \Rightarrow m \in (0 + \frac{15}{4}, \frac{15}{2}) = (\frac{15}{4}, \frac{15}{2}), \text{ i.e., } m \in [4, 7];$$

$$\text{3rd answer 1} \Rightarrow m \in (\frac{15}{4} + \frac{15}{8}, \frac{15}{2}) = (\frac{45}{8}, \frac{15}{2}), \text{ i.e., } m \in [6, 7];$$

Algorithm 8.4: Binary Searching RSA Plaintext Using a Parity Oracle

INPUT (N, e) : RSA public-key material;
 $c = m^e \pmod{N}$: an RSA ciphertext;
 PO_N : a parity oracle, on inputting an RSA ciphertext,
 it returns the least significant bit of the corresponding plaintext;
 OUTPUT m .

1. Set $(a, b) = (0, N)$;
 (* The length of the interval (a, b) will be halved in each iteration of
 the following searching steps. *)
2. For $i = 1, 2, \dots, \lfloor \log_2 N \rfloor + 1$ do
 {
 (* The length of (a, b) is always under $\frac{N}{2^{i-1}}$. *)
 2.1 If ($PO_N(2^{ie}c) == 0$) then $b \leftarrow b - \frac{N}{2^i}$;
 (* m is in the lower half of (a, b) *)
 2.2 Else $a \leftarrow a + \frac{N}{2^i}$;
 (* m is in the upper half of (a, b) *)
 }
 }
3. Return($\lfloor b \rfloor$).

Figure 8.6.

4th answer $1 \Rightarrow m \in (\frac{45}{8} + \frac{15}{16}, \frac{15}{2}) = (\frac{105}{16}, \frac{15}{2})$, i.e., $m \in [7, 7]$.

So we have found $m = 7$. Indeed, $7^3 = 13 \pmod{15}$. □

This result suggests that the RSA least significant bit can be as strong as the whole block of the plaintext, provided the holding of the RSAA.

In Example 8.5 we have seen that it is dangerous for a user, as the owner of an RSA public key, to act as a decryption oracle to return a plaintext as a whole data block to a decryption request. Now from the “RSA least significant bit security” result we further know that the user must also not act as an “parity oracle”, or an “ $N/2$ -oracle” (due to Lemma 8.1) to answer any cipher query on the parity bit of the corresponding plaintext (or to answer whether the plaintext is less than $N/2$).

We should warn the reader that an attacker may embed such queries in an innocent-looking protocol. See the following example.

Example 8.11: When Malice and Alice need to agree on a secret session key to be shared exclusively between them, Malice may provide the following reasonable suggestion:

“Alice, how about we send to each other 1,000 ciphertext messages encrypted under our respective public keys? Let the session key be the bit-string from XORing the parity bits of each pair of the exchanged plaintext messages. By the way, to assure you that the session key will be random, let me send my 1,000 blocks to you first!”

Alice not only agrees, she is also grateful for the trust Malice has over her (in making the session key random)! However, the 1,000 ciphertext messages Malice sends to her will be $(2^i)^e c \pmod{N}$ ($i = 1, 2, \dots, 1000$) where c is a ciphertext someone else sent to Alice and was eavesdropped by Malice.

After the protocol, Malice pretends to have erred in the computation of the session key:

“Alice, I’m sorry for having messed up my computation. Would you be so kind and send me the session key encrypted under my public key?”

Poor Alice offers help. Alas, from the session key, Malice can extract the needed parity bits and then applies Algorithm 8.4 to discover the plaintext encrypted inside c ! \square

8.8.2 The Rabin Bit and the Strength of the Blum-Blum-Shub Pseudo Random Bits Generator

Algorithm 8.4 can be modified and applied to the Rabin encryption if the encryption takes the simple form of $c = m^2 \pmod{N}$ (i.e., the case of encryption exponent being $e = 2$, and that is all the “modification”). But there is some complication. We have studied (in §6.5) that any given $c \in \text{QR}_N$ has four distinct square roots modulo N , i.e., the ciphertext c has four different plaintexts. If an parity-oracle somehow answers the parity of a random square root of c (i.e., random among the four), then this oracle is not a reliable one and cannot be used. However, if a square root has certain properties which allow an oracle to do the job deterministically, then the binary-search technique can still be applied to the Rabin encryption.

One example of such a deterministic oracle is one which answers the parity bit of a smaller square root of the positive Jacobi symbol. By Theorem 6.18, we know that if N is a Blum integer, then any quadratic residue c has two roots m , $-m$ of the positive Jacobi symbol. Since N is odd, only one of these two roots is less than $\frac{N}{2}$, and so we can call it the “smaller root of c of the positive Jacobi symbol”. Now if we confine N to be in a more restricted form of Blum integer, such that $\left(\frac{2}{N}\right) = 1$

($N = pq$ with $p \equiv q \pmod{8}$) will do, notice also we have $\left(\frac{-1}{N}\right) = 1$, then a parity oracle which answers the parity bit of a smaller square root of the positive Jacobi symbol will work. Notice that with $\left(\frac{2}{N}\right) = 1$, an i -th query made to this reliable parity oracle will have the plaintext $\frac{m}{2^i} \pmod{N}$ and so will keep the sign of the Jacobi symbol for all i plaintext queries. For the details of a modified binary search algorithm for the Rabin encryption case, see [GMT82].

This result suggests that the Rabin least significant bit is strong if the IFA hold. The strength of the Rabin least significant bit has an important application: **cryptographically strong pseudo-random bits** (CSPRB) generation [BBS86]. The so-called **Blum-Blum-Shub pseudo-random number generator** uses a seed $x_0 \in \text{QR}_N$ where N is a k -bit Blum integer. Then the pseudo-random bits generated from the BBS generator using the seed x_0 are composed of the least significant bit of each number in the following sequence

$$x_0, x_1 = x_0^2, \dots, x_i = x_{i-1}^2, \dots \pmod{N} \quad (8.8.1)$$

It can be shown [BBS86], [GMT82] that, without knowing the seed x_0 , predicting the least significant bits in the sequence in (8.8.1) is computationally equivalent to factoring the Blum integer N .

Remark 8.2: *It is also known [ACGS88], [VV85] that the problem of extracting the simultaneous $\log_2 \log_2 N$ least significant bits from a Rabin ciphertext is equivalent to factoring N .* \square

Blum and Goldwasser applied this result and proposed an efficient cryptosystem which has a strong sense of security called **semantic security**. We shall study semantic security in Chapter 13, there we shall also introduce the semantically secure cryptosystem of Blum and Goldwasser based on the strength of the Rabin bit.

8.8.3 The ElGamal Bit

For the ElGamal cryptosystem given in the form of Algorithm 8.3, since the plaintext message space is \mathbb{Z}_p^* where p is a prime number (hence odd), it is straightforward that the binary search technique can also be applied. To find the plaintext message encrypted under a ciphertext pair (c_1, c_2) , the querying ciphertext messages sent to a parity oracle should be

$$(c_1, 2^i c_2) \pmod{p}, \quad i = 1, 2, \dots, \lfloor \log_2 p \rfloor + 1.$$

If the parity oracle is a human being (this is very likely, see Example 8.11), then in order to avoid suspicion, the attacker can blind the queries, for instance, as

follows:

$$(g^{r_i} c_1, 2^i y^{r_i} c_2) \pmod{p} \text{ with } r_i \in_U \mathbb{Z}_{p-1}, i = 1, 2, \dots, \lfloor \log_2 p \rfloor + 1,$$

where (g, y) are the public key material of the parity oracle. These $\lfloor \log_2 p \rfloor + 1$ pairs of ciphertext messages are completely independent one another, however, they encrypt the related message series $2^i m \pmod{p}$ for $i = 1, 2, \dots, \lfloor \log_2 p \rfloor + 1$:

$$(g^{k+r_i}, y^{k+r_i} 2^i m) \pmod{p}, i = 1, 2, \dots, \lfloor \log_2 p \rfloor + 1.$$

To this end we can conclude that the bit security of the ElGamal cryptosystem is as hard as the block data security. On the other hand, a public key owner should be careful not to be tricked to play the game as in Example 8.11.

8.8.4 The Discrete Logarithm Bit

In §8.2.2 we have discussed that in an abelian group in the general case, the discrete logarithm problem is hard: the function g^x is believed to be one-way and so far it is not known whether the function is a trapdoor. But at any rate, let us assume that there exists some oracles which can answer some bit-level partial information of x upon being fed a pair (g, g^x) .

If the element g has an odd order, then there exists value $\frac{1}{2} \pmod{\text{ord}(g)}$ and this value is available if $\text{ord}(g)$ is not secret (this is the usual case). In this situation, the problem of extracting the discrete logarithm using a parity oracle is in fact, bit-by-bit, the reverse operation of the modulo exponentiation algorithm (see Algorithm 4.3). Since the modulo exponentiation algorithm is also called “squaring-and-multiplying” method, the reverse algorithm should be called “square-rooting-and-dividing” method. Algorithm 8.5 specifies such a method.

Now what happens if g has an even order? For example, if g is a generator element in \mathbb{Z}_P^* with P being a prime number, then $\text{ord}_P(g) = P - 1$ is even and is not relatively prime to 2. Therefore $\frac{1}{2} \pmod{P-1}$ does not exist. So square-rooting of h cannot be done in the form of Step 2.2 in Algorithm 8.5.

However, in this case (i.e., when g is a generator element in \mathbb{Z}_P^*), we can still compute square roots of h modulo P using Algorithm 6.4. For any quadratic residue element $h \in \text{QR}_P$, that square-rooting algorithm will return two square roots of h , which we should denote by $\pm\sqrt{h}$. Since g is a generator of \mathbb{Z}_P^* , it holds $g \in \text{QNR}_P$; but $h \in \text{QR}_P$, therefore $\log_g h$ must be an even number. Thus, without loss of generality, we can write the discrete logarithms of the two square roots of h to the base g as follows:

$$\log_g \sqrt{h} = \frac{\log_g h}{2}, \quad \log_g (-\sqrt{h}) = \frac{P-1}{2} + \frac{\log_g h}{2}. \quad (8.8.2)$$

Notice because addition in (8.8.2) is computed modulo $P-1$, exactly one of the two values in (8.8.2) is less $\frac{P-1}{2}$, the other must be greater than or equal to $\frac{P-1}{2}$.

Algorithm 8.5: Extract Discrete Logarithm Using a Parity Oracle

INPUT (g, h) : g is a group element of an odd order, $h = g^x$;
 $PO_{\text{desc}(g)}$: a parity oracle, $PO_{\text{desc}(g)}(g, h) = \log_g h \pmod{2}$;
OUTPUT integer x .

1. Set $x \leftarrow 0$; $y \leftarrow \frac{1}{2} \pmod{\text{ord}(g)}$;
2. Repeat the following steps until $h = 1$ (* including $h = 1$ *)

$\{$
 - 2.1 If ($PO_{\text{desc}(g)}(g, h) == 1$) then $h \leftarrow h/g$; $x \leftarrow x + 1$;
 (* When $\log_g h$ is odd, do “division and plus 1”, as reverse to
 “multiplication and minus 1” in modulo exponentiation *)
 - 2.2 $h \leftarrow h^y$; $x \leftarrow 2x$;
 (* Now $\log_g h$ is even, do “square rooting and doubling” as reverse
 to “squaring and halving” in modulo exponentiation *) $\}$
3. Return(x).

Figure 8.7.

Clearly, the square root which has the smaller discrete logarithm to the base g is the correct square root. The trouble is, from \sqrt{h} and $-\sqrt{h}$, we cannot see which one of the two square roots has the smaller discrete logarithm to the base g !

Nevertheless, if we have a different “one-bit-information oracle” which, upon being fed $(g, y, -y)$, answers y or $-y$, whichever has the smaller discrete logarithm to the base g , then we can use this oracle (call it “half-order oracle”) to pick the correct square root for us. Algorithm 8.6, which is modified from Algorithm 8.5, does the job using the “half-order oracle”.

Since testing Legendre symbol and computing square roots modulo a prime can be efficiently done, from Algorithm 8.6 we know that being able to decide from (g, h) whether $\log_g h$ is less than $\frac{\text{ord}(g)}{2}$ is equivalent to extracting $\log_g h$ from (g, h) .

Algorithm 8.6 is more general than Algorithm 8.5 in that, it will also work with g of odd order. Let us therefore go through Algorithm 8.6 with a small numerical example.

Example 8.12: Suppose we have a “half-order oracle”. For group \mathbb{Z}_{23}^* , generator

Algorithm 8.6: Extract Discrete Logarithm Using a “Half-order Oracle”

INPUT (g, h, P) : g is a generator of \mathbb{Z}_P^* with P prime; $h = g^x \pmod{P}$;
 $PO_{(P,g)}$: a half-order oracle,
 $PO_{(P,g)}(g, y, -y) = \begin{cases} y & \text{if } \log_g y < \log_g(-y) \\ -y & \text{otherwise} \end{cases}$;
 OUTPUT integer x .

1. Set $x \leftarrow 0$;
2. Repeat the following steps until $h = 1$ (* including $h = 1$ *)
 - {
 - 2.1 If ($h \in \text{QNR}_P$) then $h \leftarrow h/g$; $x \leftarrow x + 1$;
 (* $h \in \text{QNR}_P$ implies that $\log_g h$ is odd; this can be done by testing Legendre symbol $\left(\frac{h}{P}\right) = -1$. *)
 - 2.2 $h \leftarrow PO_{(P,g)}(g, \sqrt{h}, -\sqrt{h})$; $x \leftarrow 2x$;
 (* We can do square-rooting with no difficulty, but we need the oracle to tell us which root is the correct one, i.e., has the smaller discrete logarithm $\frac{\log_g h}{2}$. *)
 - }
3. Return(x).

Figure 8.8.

$g = 5$ and element $h = 9$, let us extract $x = \log_5 9 \pmod{22}$ by calling the “half-order oracle”.

The following is an execution tree of Algorithm 8.6 on input $(5, 9, 23)$. Double arrows stand for the “square-rooting” computation, of which the horizontal ones (\Rightarrow) are those chosen by the “half-order oracle”. Single arrows stand for the “dividing-5” computation. All computations are performed modulo 23.

$$\begin{array}{cccccccc}
 9 & \Rightarrow & 20 & \rightarrow & 4 & \Rightarrow & 2 & \Rightarrow & 5 & \rightarrow & 1 & \Rightarrow & 1 \\
 \Downarrow & & & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\
 3 & & & & 21 & & 18 & & & & 22 & &
 \end{array}$$

At the starting of the algorithm, x is initialized to 0 (step 1). For each \Rightarrow , $x \leftarrow 2x$ will be performed (step 2.2), and for each \rightarrow , $x \leftarrow 1$ will be performed

(step 2.1). Upon termination of the algorithm, the final value for x is 10. Indeed, $9 = 5^{10} \pmod{23}$. \square

These results show that the individual bits of discrete logarithm are in general as hard as the whole block. We now also know that if a generator element is a quadratic residue, then all bits including the least significant bit of a discrete logarithm to this base are hard. This leads to a “semantically secure” version of the ElGamal cryptosystem, which we shall introduce in Chapter 13.

8.9 Key Channel Establishment for Public-key Cryptosystems

The well-known man-in-the-middle attack on the Diffie-Hellman key exchange protocol (see §8.2.1) is general in public-key cryptosystems. In general, to send a confidential message to a recipient by encrypting under her/his public key, the sender must first make sure that the key to be used really belongs to the intended recipient. Likewise, upon receipt a “digital envelope”, the recipient must make sure that the “envelope” is really from the claimed source before engaging in a confidential communications using the symmetric key retrieved from the “envelope”.

Thus, with the fascinating public-key cryptographic techniques, there is still a need for establishing a secure key channel between communication parties. However, in public-key cryptography $ke \neq kd$ (see Figure 7.1). Therefore transporting an encryption key ke to the message sender need not involve handling of any secret. Therefore, the task for establishing a secure key channel is purely an authentication problem, namely, the key channel involves no handling of any secret and should only preserve the authenticity of the encryption key.

Authenticated key channel establishment for public keys will be the topic for Chapter 12. Directory based techniques for public-key channel setting-up will be introduced in §12.2 while some identity based techniques will be introduced in §12.3.

8.10 Chapter Summary

In this chapter we have introduced several well-known and widely-in-use public-key encryption algorithms: Diffie-Hellman key exchange protocol, the RSA, Rabin and ElGamal encryption algorithms. Along with these basic public-key algorithms, we introduce respective hard problems as complexity-theoretic assumptions which are the security underpins for the basic public-key encryption algorithms.

We declare that the quality of security considered in this chapter: all-or-nothing secrecy and passive attacker, is a low one: it is a security notion only suitable for “textbook version” encryption algorithms. All public-key encryption algorithms introduced in this chapter are “textbook versions”. Various attacks on them are demonstrated to show that they are not fit for applications. We discuss the need for

more stringent and “fit-for-application” security notions for public-key encryption algorithms, however, defer their introduction to a later chapter.

We then investigate the hardness of the complexity-theoretic problems for the basic public-key encryption algorithms at each individual bit level. Our investigation results are invariantly positive: for all cases, each individual bit can be as secure as the whole block. These positive results are necessary in order for us to enhance “textbook versions” of the algorithms into “fit-for-application” ones in a later chapter.

DATA INTEGRITY TECHNIQUES

9.1 Introduction

In Chapter 2 we made a realistic and standard assumption on the vulnerability of the open communications network: all communications go through an adversary named Malice who is free to eavesdrop, intercept, relay, modify, forge or inject messages. When Malice injects modified or forged messages, he will try to fool the targeted receivers into believing that the messages are sent from some other principals. To use such a vulnerable communications medium in a secure manner, as is required for secure electronic commerce transactions, cryptographic mechanisms which can provide the security service in terms of message confidentiality (i.e., protection against eavesdropping) are inadequate. We need mechanisms which can enable a message receiver to verify that a message has indeed come from the claimed source and has not been altered in an unauthorized way during the transmission. **Data integrity** is the security service against unauthorized modification of messages.

Data integrity in modern cryptography is closely related to, and evolves from, a classical subject in communications: **error-detection code**. The latter is a procedure for detecting errors which can be introduced into messages due to fault in communications. It is considered that using information which has been modified in a malicious way is at the same risk as using information which contains defects due to errors introduced in communications or data processing. As a result, the working principle of the techniques providing data integrity and that of techniques providing error-detection codes are essentially the same: a transmitter of a message creates a “checking value” by encoding some redundancy into the message to be transmitted and appends the checking value to the message; a receiver of the message then verifies the correctness of the message received using the appended checking value according to a set of rules which are agreed with the transmitter [Sim92]. In error-detection codes, the redundancy is encoded in such a way that the receiver can use a maximum likelihood detector to decide which message he should infer as having

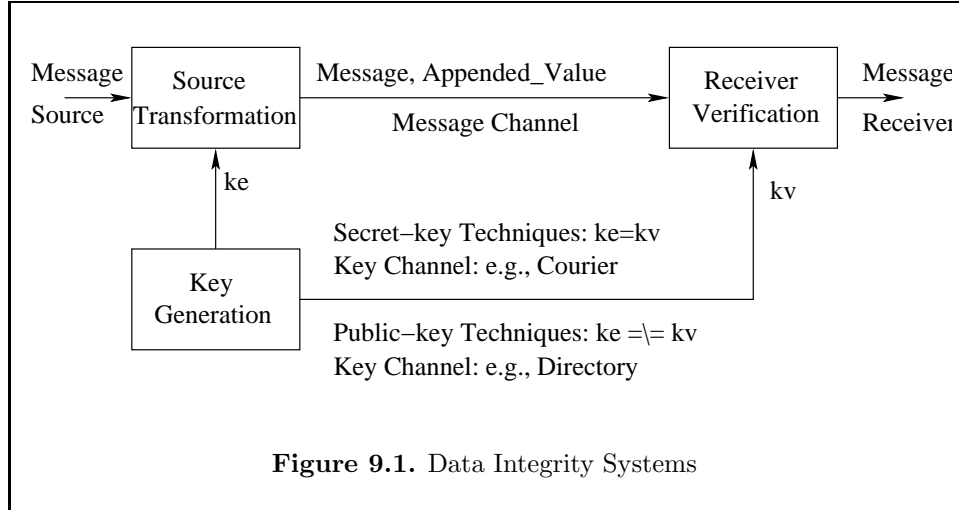


Figure 9.1. Data Integrity Systems

most likely been transmitted from the possibly altered codes that were received. In data integrity, the redundancy is encoded in such a way that the appended checking value will be distributed as uniform as possible to the entire message space of the checking values. The cryptographic transformation for the latter way of adding redundancy is similar to the mixing-transformation for encryption that we have described in §7.1).

Like an encryption algorithm, the cryptographic transformations for achieving data integrity should also be parameterized by keys. Thus, in the usual sense, a correct data-integrity verification result will also provide the verifier with the knowledge of the message source, that is, the principal who had created the data integrity protection. However, recently a notion of “data integrity without source identification” has emerged. This new notion is important in the study of public-key cryptosystems secure against adaptive attackers. In this chapter we will use an example to introduce this notion which serves a preparation for a later chapter when we study the public-key cryptosystems secure against adaptive attackers.

We are now ready to characterize the notion of data integrity.

Definition 9.1: (Data Integrity System) *Let Data be arbitrary information. Data integrity protection on Data is a cryptographic service comprising the following cryptographic operations:*

1. *Manipulation detection code creation:*

$$\text{MDC} = f(K, \text{Data});$$

2. *Manipulation detection code verification:*

$$\text{MDC} = g(KV, \text{Data});$$

where f and g are some cryptographic transformations, and K and KV are some auxiliary input values such that a consistency between **Data** and **MDC** is created and can only be verified by following the given cryptographic operations.

In this definition, **MDC** stands for **manipulation detection code**.

Figure 9.1 provides an illustration of data integrity systems.

We should notice that although in our introductory discussions (and in Figure 9.1) we have used a communications scenario to introduce the notion of data integrity, Definition 9.1 does not confine the notion to communications; for example, the pair (**Data**, **MDC**) can data stored to or retrieved from an insecure data storage.

9.1.1 Chapter Outline

In this chapter we will introduce several well-known primitive cryptographic techniques for providing data integrity service. Our introduction to these techniques will be divided into symmetric ones and asymmetric ones. The former will be introduced in §9.2. The latter techniques are further divided into two categories: data integrity with message source identification which will be introduced in §9.3, and that without source identification; we will exemplify this notion in §9.4.

9.2 Symmetric Techniques

In symmetric techniques for achieving data integrity, the cryptographic transformations f and g (see Definition 9.1) are a symmetric cryptographic algorithm which means $f = g$ and $K = KV$, that is, the creation and the verification of the consistency between **Data** and **MDC** use the identical cryptographic operation.

Due to a close relation between data integrity and **message authentication** (to discuss in Chapter 9), **MDC** created by a symmetric cryptographic technique is often called a **message authentication code** (a **MAC** for short). A **MAC** can be created and verified using a **keyed hash function** technique, or using a block cipher encryption algorithm.

9.2.1 Cryptographic Hash Functions

A common method for realizing a **MAC** is to use a so-called **keyed hash function** technique. We first introduce the notion of cryptographic hash function.

In modern cryptography which is based on a complexity-theoretic approach (meaning: the security is based on some assumed intractabilities of certain computational problems, review Chapter 4), we *assume* the existence of one-way hash functions (hash function for short). Such a function is deterministic and maps an arbitrary length of (binary) string to a **hashed value** which is a (binary) string of a fixed length. Let $h()$ denote such a hash function whose fixed output length is denoted by $|h|$. It is assumed that $h()$ should have the following properties.

- $h()$ is a **good mixing-transformation**: that is, on any input, the output hashed value should be computationally indistinguishable from a uniform binary string in the interval $[0, 2^{|h|})$. For this assumption to be reasonable, it need to be the case that the input of $h()$ is taken from a sufficiently large space.
- $h()$ is **collision resistant**: that is, it should be computationally infeasible to find two inputs x, y with $x \neq y$ such that $h(x) = h(y)$. For this assumption to be reasonable, it need to be the case that the space of the hashed values of $h()$ should be sufficiently large. The least value for $|h|$ is 128 while a typical value is 160.
- $h()$ is **pre-image resistant**: that is, given a hashed value h , it should be computationally infeasible to find an input x such that $h = h(x)$. This assumption also requires the space of the hashed values of $h()$ be sufficiently large.
- $h()$ is **practically efficient**: that is, given input x , the computation of $h(x)$ can be done in time bounded by a small-degree polynomial in the size of x .

In the design of a hash function, the mixing-transformation property and the collision resistant property can be realized by using operations similar to those used in a block cipher algorithm; and the pre-image resistant property can be realized using some non-invertible data compression technique. We shall not describe the design details of any real hash function. The interested reader can find in the literature (e.g. Chapter 9 of [MvOV97]).

9.2.1.1 Random oracle

A hash function's mixing-transformation property together with its deterministic property approximate the behavior of a so-called **random oracle**.

A random oracle is an imaginary function. It deterministically and efficiently maps an input value of any distribution and any length to an output value of the uniform distribution in its finite range space. Notice that this function is deterministic: if a random oracle is fed an input value twice, the same output value will be output. A standard phrase to express a calling of such a *very very powerful*

function is “**query**”: we query a random oracle with a value and it will efficiently reply a uniformly distributed random value. Here efficiently means polynomial-time in the size of the query. We regard this function to be *very very powerful* because of the three properties we have described: deterministic, efficient and the uniform output. These three properties means that a random oracle can relate two independent values efficiently and deterministically, for example, the relation between two independent values $RO(x)$ and $RO(x + 1)$ can be efficiently found by RO as a “plus 1” relation in the domain of RO.

We would really like a real-world hash function to have such a random-oracle property. However, from all computational models we know of, there exists no computing mechanism or machinery which can be so powerful. On the one hand, we can output uniformly distributed random values efficiently, e.g., by flipping a fair coin, but we do not know how to do it in a deterministic fashion (say, a fixed way of coin flipping always produces a fixed random value). On the other hand, we can also relate uniformly independent values deterministically, e.g., sorting a set of such values so that any two of them has a deterministic relation as the distance between them in the sorted list; but this relation cannot be computed in polynomial-time in the size of these random values; in fact, it even needs a memory of a size exponential in the size of these random values!

As a matter of fact, with deterministic, efficient yet uniformly distributed response to a query of any distribution, the response will have an information entropy greater than that of the query (review §3.6 for the meaning of entropy); that is, this deterministic function is an “entropy amplifier”. There is definitely no such thing! A random oracle only exists in an imaginary world.

Moreover, since the mixing-transformation property is only a computational assumption, a real-world hash function should in fact have its output to follow some probability distribution which may or may not be discernible by a polynomially bounded distinguisher. Thus any real-world hash function only emulates the random oracle behavior to a precision where the difference is hopefully a negligible quantity.

Nevertheless, such an emulated random oracle behavior by a hash function plays an important role in cryptography. In essence, to hash a message is to add certain quality redundancy to the message in a deterministically verifiable way. Digital signature schemes and public-key encryption schemes with security paddings and with provable security widely apply hash functions (see §9.3 and §9.4).

9.2.1.2 Birthday attack

Assuming that a hash function $h()$ behaves really as a random oracle, the square-root attack (the birthday attack, see §3.5) suggests that

$$2^{|h|/2} = \sqrt{2^{|h|}}$$

random evaluations of the hash function will suffice an attacker to obtain a collision with a non-negligible probability. To mount a birthday attack, the attacker should generate random message-hash pairs

$$(m_1, h(m_1)), (m_2, h(m_2)), \dots$$

until he ends up with finding two messages m and m' satisfying

$$m \neq m', \quad h(m) = h(m'). \quad (9.2.1)$$

Such a pair of messages is called a collision under the hash function h . Of course, in order for a birthday attack to be useful for the attacker, the collision message m and m' should contain some fixed sub-messages. For example, let a message to be hashed (and to be digitally signed, see §9.3) be a payment authorization statement in the following form

$$M = \text{Price, Goods_Description, } R$$

where R is a random number to randomize the protocol messages (it is always desirable that protocol messages are randomized). Then an interesting birthday attack can be

$$m = \text{Price}_1, \text{ Goods_Description, } r$$

and

$$m' = \text{Price}_2, \text{ Goods_Description, } r'$$

where $\text{Price}_1 \neq \text{Price}_2$ and Goods_Description are fixed message parts, and the collision is on the random numbers $r \neq r'$. Collision finding for such messages has the same complexity as collision finding for random messages as in (9.2.1) since we can view

$$h'(x) \stackrel{\text{def}}{=} h(\text{Price}_1, \text{ Goods_Description, } x)$$

and

$$h''(x) \stackrel{\text{def}}{=} h(\text{Price}_2, \text{ Goods_Description, } x)$$

as two random functions.

It is obvious that fewer evaluations will be needed if a hash function is not a truly random function.

Thus, the size of the output space of a cryptographic hash function must have a lower bound. The current widely used hash functions in applied cryptography are SHA-1 [NIS95] and RIPEMD-160 [BDP97]. Both have the output length $|h| = 160$. Their strength against the square-root attack is therefore 2^{80} . This is compatible to the strength of a block cipher algorithm of the key length up to 80 bits. The previous popular hash function MD5 [Riv92] has the case $|h| = 128$ which was tailored to suit the DES's key length of 56 bits and block length of 64 bits.

With the introduction of the AES-128, AES-192 and AES-256 (the AES of key lengths 128, 192 and 256 bits, respectively, see §??), standard bodies (e.g., the ISO/IEC [ISO01]) are currently standardizing hash functions of compatible output lengths $|h| \in \{256, 384, 512\}$.

9.2.1.3 Hash functions in public-key cryptography

Hash functions with large output spaces also find important applications in public-key cryptography:

- Hash functions are widely used in digital signatures for preparing “message digests” or “message finger-prints”. We will see such applications in §9.3.
- Hash functions are widely used in some encryption algorithms where their usage is to realize message padding schemes to turn a “textbook encryption scheme” into “CCA2-secure” (see §8.6 for the meaning of textbook encryption scheme and see Definition 8.3 for the meaning of CCA2). An example of such a padding scheme for the RSA encryption is given in §9.4. In Chapter 13 we will study some cryptosystems which are secure with respect to CCA2.

9.2.2 MAC Based on a Keyed Hash Function

Cryptographic hash functions naturally form a cryptographic primitive for data integrity. For use in a shared-key scenario, a hash function takes a key as part of its input. The other part of the input is the message to be authenticated. Thus, to authenticate a message M , a transmitter computes

$$\text{MAC} = h(K \parallel M),$$

where K is a secret key shared between the transmitter and a receiver, and “ \parallel ” denotes the bit string concatenation.

From the properties of the hash function listed in §9.2.1, we can assume that in order to create a valid MAC using the hash function with respect to a key K and a message M , a principal must actually have in possession of the correct key and the correct message. The receiver who shares the key K with the transmitter should re-calculate the MAC from the received message M and check that it agrees with the MAC received. If so the message can be believed to be from the claimed transmitter.

Because such a MAC is constructed using a hash function, it is also called an HMAC. It is often a prudent practice that an HMAC is computed in the following format

$$\text{HMAC} = h(k \parallel M \parallel k),$$

that is, the key is pre-fixed and post-fixed to the message to be authenticated [Tru92]. This is in order to prevent an adversary from exploiting a “round-function iteration” structure of some hash functions. Without guarding the both ends of the message with a secret key, such a known structure of certain hash functions may allow an adversary to modify the message by pre-fixing or post-fixing some chosen data to the message without need of knowing the secret key K .

9.2.3 MAC Based on a Block Cipher Encryption Algorithm

A standard method for forming a keyed hash function is to apply the CBC mode of operation using a block cipher algorithm. Conventionally, a keyed hash function so constructed is called a MAC.

Let $\mathcal{E}_K(m)$ denote a block cipher encryption algorithm keyed with the key K on inputting the message m . To authenticate a message M , the transmitter first divide M as

$$M = m_1 m_2 \dots m_\ell$$

where each sub-message block m_i ($i = 1, 2, \dots, \ell$) has the size of the input of the block cipher algorithm. Padding of a random value to the last sub-message block m_ℓ may be necessary if the final block is not of the full block size. Let $C_0 = IV$ be a random initializing vector. Now the transmitter applies the CBC encryption:

$$C_i \stackrel{\text{def}}{=} \mathcal{E}_k(m_i \oplus C_{i-1}), \quad i = 1, 2, \dots, \ell.$$

Then the pair

$$(IV, C_\ell)$$

will be used as the MAC to be appended with M and sent out.

It is obvious that the computation for creating a CBC-MAC involves non-invertible data compression (in essence, a CBC-MAC is a “short digest” of the whole message), and so a CBC-MAC is a one-way transformation. Moreover, the mixing-transformation property of the underlying block cipher encryption algorithm adds a hash feature to this one-way transformation (i.e., distributes a MAC over the MAC space as uniform as the underlying block cipher should do over its ciphertext message space). Thus, we can assume that in order to create a valid CBC-MAC, a principal actually has to have in possession of the key K which keys the underlying block cipher algorithm. The receiver who shares the key K with the transmitter should re-calculate the MAC from the received message and check that it agrees with the version received. If so the message can be believed to be from the claimed transmitter.

We will sometimes denote by $\text{MAC}(K, M)$ a MAC which provides the integrity service on the message M for principals who share the key K . In this denotation we ignore the implementation details such as what underlying one-way transformation has been used for the MAC’s realization.

9.3 Asymmetric Techniques I: Digital Signatures (Textbook Versions)

In public-key cryptography, a principal can use her/his private key to “encrypt” a message and the resultant “ciphertext” can be “decrypted” back to the original

message using the principal's public key. Evidently, the “ciphertext” so created can serve the function of a manipulation detection code (MDC), that is, to provide a data integrity protection for the message “encrypted” in; here, the public-key “decryption” process forms a step of verification of the MDC.

Moreover, while the verification of such an MDC can be performed by anybody (since the public key is available to anybody), it is considered that only the owner of the public key used in the MDC verification could have created the MDC. Thus, this usage of public key cryptography models precisely a signature, a **digital signature**, for showing the authorship of a message, or in other words, showing who has “signed” the message. Diffie and Hellman envisioned the notion of digital signature in 1975 [DH76a] (the publication date of this paper was 1976, but the paper was first distributed in December 1975 as a preprint, see [Dif92]). The ability to provide a digital signature forms a great advantage of public-key cryptography over secret-key cryptography (the other significant advantage of public-key cryptography is the possibility of achieving key distribution among remote parties, see §?? and §8.7). Now that only a single entity is able to create a digital signature on a message which can be verified by anybody, it is easy to settle a dispute on who has created the signature. This allows provision of a security service named **non-repudiation** which means no denial of a connection with a message. Non-repudiation is a necessary security requirement in various applications such as electronic commerce over the Internet.

Syntactically, a digital signature scheme can be defined follows.

Definition 9.2: (Digital Signature) *A digital signature scheme consists of the following:*

- a plaintext message space \mathcal{M} : a set of strings over some alphabet
- a signature space \mathcal{S} : a set of possible signatures
- a signing key space \mathcal{K} : a set of possible keys for signature creation, and a verification key space \mathcal{K}' : a set of possible keys for signature verification
- an efficient key generation algorithm $\text{Gen} : \mathbb{N} \mapsto \mathcal{K} \times \mathcal{K}'$
- an efficient signing algorithm $\text{Sign} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{S}$
- an efficient verification algorithm $\text{Verify} : \mathcal{M} \times \mathcal{S} \times \mathcal{K}' \mapsto \{\text{True}, \text{False}\}$.

For any $K \in \mathcal{K}$ and any $m \in \mathcal{M}$, we denote by

$$s = \text{Sign}_K(m)$$

the signing transformation and read it as “ s is a signature of m under key K ”.

For any $K \in \mathcal{K}$, let KV denote the verification key matching K , and for $m \in \mathcal{M}$, $s \in \mathcal{S}$, it is necessary

$$\text{Verify}_{KV}(m, s) = \begin{cases} \text{True} & \text{if } s = \text{Sign}_K(m) \\ \text{False} & \text{if } s \neq \text{Sign}_K(m) \end{cases}$$

Notice that the integer input to the key generation algorithm **Gen** provides the size of the output signing/verification keys. The integer should be unary encoded (see Definition 4.7)

Semantically, Shannon's mixing-transformation characterization for encryption algorithms (see §7.1) is also sensible for digital signatures: **Sign** should distribute the messages in \mathcal{M} fairly uniformly over the entire signature space \mathcal{S} . However we should point out that the mixing-transformation property, while in the case of encryption aiming to achieve a difficulty for someone to find the plaintext message without using the decryption key, here aims to achieve a difficulty for someone to create a valid signature without using the signing key (difficulty for forging a signature).

Analogous to Property 8.2 as a very weak security notion for the textbook public-key encryption algorithms introduced in Chapter 8, we also have the following remark for a very weak security notion for the digital signature schemes to be introduced in this chapter.

Remark 9.1: *Within the scope of this chapter, we shall only consider a very weak security notion for digital signature schemes: it is computationally infeasible for an attacker to forge (i.e., to create) a valid signature “from scratch”. That is, the attacker is given a message, the description of a signature scheme and a public key, and is required to output a valid message-signature pair. The attacker does not attempt to ease a forgery by trying to make use of available message-signature pairs or an opportunity to interact with a signer for the latter to produce signatures on its chosen messages.* \square

We should notice that this notion of security for digital signatures is inadequate in most applications (in particular, in cryptographic protocols applications) because it assumes that the attacker is unreasonably weak or that environment is extremely harsh to the attacker. A stronger notion of security named **adaptively chosen message forgery**, which corresponds to CCA2 in cryptosystems (see Definition 8.3), will be introduced in Chapter 13. Some practical digital signature schemes which are provably secure with respect to adaptively chosen message forgery will be introduced in a couple of later chapters.

Let us now introduce several well-known digital signature schemes.

Algorithm 9.1: The RSA Signature Scheme**Key Setup**

The key setup procedure is the same as that for the RSA cryptosystems (Algorithm 8.1).

(* Thus, user Alice's public-key material is (N, e) where $N = pq$ with p and q being two large prime numbers of roughly equal size, and e is an integer such that $\gcd(e, \phi(N)) = 1$. She also finds an integer d such that $ed \equiv 1 \pmod{\phi(N)}$. The integer d is Alice's private key. *)

Signature Generation

To create a signature on message $m \in \mathbb{Z}_N^*$, Alice creates

$$s = \text{Sign}_d(m) \stackrel{\text{def}}{=} m^d \pmod{N}.$$

Signature Verification

(* Let Bob be a verifier who knows that the public-key material (N, e) belongs to Alice *)

Given a message-signature pair (m, s) , Bob's verification procedure is

$$\text{Verify}_{(N,e)}(m, s) = \text{True} \text{ iff } m \equiv s^e \pmod{N}.$$

Figure 9.2.**9.3.1 The RSA Signature**

The RSA signature scheme is the first digital signature scheme which is realized by Rivest, Shamir and Adleman [RSA78]. The RSA signature scheme is specified in Algorithm 9.1.

It is easy to see that the RSA digital signature procedures are in the same format as those for the RSA encryption and decryption (see §??), except that now Alice performs “encryption” first using her private key, and Bob (or anybody) performs “decryption” later using Alice's public key. The holding of the verification congruence for a valid signature follows exactly the argument we have made in §?? for the cases of the RSA encryption and decryption.

9.3.1.1 Message recognizability in RSA signature

If the RSA signature scheme is just as simple as we have described, then it is not a difficult problem at all for anybody to forge Alice's signature. For example, Bob

can pick a random number $s \in \mathbb{Z}_N^*$ and compute

$$m \stackrel{\text{def}}{=} s^e \pmod{N}. \quad (9.3.1)$$

Of course, for so prepared “message”-signature pair, the verification will return **True**!

In fact, it is always the case that for any digital signature scheme, given a message-signature pair (m, s) and a public key KV , $\text{Verify}_{KV}(m, s)$ should only return **True** if $m \in \mathcal{M}$ is a **recognizable** message. Here, “recognizable” means that m contains adequate redundancy which makes it “meaningful”. A common and simple method for formatting a message in \mathcal{M} to be recognizable is to have it as a hashed value mapped from a string in $\{0, 1\}^*$ using a cryptographic hash function (see §9.2.1). Let $h()$ be such a hash function mapping from $\{0, 1\}^*$ to \mathcal{M} . Then a message $m \in \mathcal{M}$ is regarded to be recognizable, i.e., meaningful, if there exists a string $M \in \{0, 1\}^*$ such that

$$m = h(M).$$

It is easy to see that under such a notion of message recognizability, to forge an RSA signature should no longer be an easy job. Computing m from $s \in \mathbb{Z}_N^*$ using Alice’s public key as in (9.3.1) does not constitute a real forgery if the attacker cannot come up with an m which is recognizable, e.g., by finding a pre-image of m under a cryptographic hash function. In fact, since the RSA encryption is a good mixing-transformation, m computed from the RSA encryption mode in (9.3.1) should look random (quite uniform in \mathbb{Z}_N^*), and therefore should not be a recognizable message.

At first glance, it seems to be a reasonable thought that the security of the RSA signature scheme (i.e., the strength against various ways of signature forgery) should be the difficulty of finding the e -th root of a given element in \mathbb{Z}_N^* , that is, it is the difficulty of the RSAP (see Definition 8.4). However, we have also seen that a mechanism for formatting a recognizable message plays an important role in the security of the RSA signature scheme. Therefore, a formal argument for the security of the RSA signature scheme should be conducted with a specific mechanism for formatting a recognizable message. We should defer such a formal argument to a later chapter where a specific method for signing in RSA will be introduced.

Similarly, for other signature schemes to be introduced in this chapter, we shall use a mapping $h : \{0, 1\}^* \mapsto \mathcal{M}$ to denote a suitable message formatting mechanism which prepares a recognizable message in \mathcal{M} before signing (and verification). The reader may always (with no harm) consider h to be a cryptographic hash function. Since no specific details about this mapping is given, formal arguments for the security of these schemes cannot be given in this chapter, and will be deferred to a later chapter.

Algorithm 9.2: The Rabin Signature Scheme**Key Setup**

User Alice sets up her public modulus same as an RSA modulus.

(* So her modulus is $N = pq$ with p, q being distinct primes odd primes. N is her public key and p, q form her private key. *)

Signature Generation

To create a signature on message $m \in \mathbb{Z}_N^*$, Alice creates signature

$$s \stackrel{\text{def}}{=} m^{1/2} \pmod{n}.$$

(* For this calculation to be possible, it is necessary for $m \in \text{QR}_N$. From §6.5 we know that for N being an RSA modulus, $\#\text{QR}_N = \#\mathbb{Z}_N^*/4$, i.e., a quarter elements in \mathbb{Z}_N^* are in QR_N . Thus, Alice can employ a suitable message formatting mechanism so that she can make sure $m \in \text{QR}_N$. For such m , Alice can use Algorithm 6.5 to compute a square root of m . *)

Signature Verification

(* Let Bob be a verifier who knows that the public modulus N belongs to Alice *)

Given a message-signature pair (m, s) , Bob's verification procedure is

$$\text{Verify}_N(m, s) = \text{True} \text{ iff } m \equiv s^2 \pmod{B}.$$

Figure 9.3.**9.3.2 The Rabin Signature**

The Rabin signature scheme [Rab79] is very similar to the RSA signature scheme. The difference between the two schemes is that they use different kinds of verification exponents. In the case of the RSA (see “RSA Signature Verification” in §9.3.1), e is an odd integer (since it is required for $\gcd(e, \phi(N)) = 1$ where $\phi(N)$ is an even number), while in the case of the Rabin, $e = 2$. The Rabin signature scheme is specified in Algorithm 9.2.

The Rabin signature has a couple of advantages over RSA. First, forgery is provably as hard as factoring (formal argument deferred). Secondly, verification is faster, and is suitable to use in applications where signature verification uses small computing devices, such as mobile phones.

9.3.2.1 message recognizability and a message formatting mechanism in Rabin signature

Analogous to the case of the RSA signature, if m is not a recognizable message, then it is trivially easy to forge a valid “message”-signature pair for the Rabin signature scheme.

Also we should notice that if Alice issues a signature twice for the same message m , then she must make sure that in both instances the same signature s should be issued. If she issues two different signatures $s \not\equiv \pm s' \pmod{N}$ for the same message m , then her modulus can be factored (see Theorem 8.2). In a later chapter we shall introduce a message formatting mechanism for the Rabin signature scheme, which formats a recognizable message and guarantees that multiple issuances of signatures for the same message will not disclose the factorization of the modulus used.

9.3.3 The ElGamal Signature

The ElGamal’s public-key cryptosystem (see §??) includes a digital signature scheme. The ElGamal signature scheme is specified in Algorithm 9.3.

9.3.3.1 Message recognizability in ElGamal signature

If m is not a recognizable message, then it is not difficult to forge a valid “message”-signature pair in the ElGamal signature. For example, let u, v be any integers less than $p - 1$ such that $\gcd(v, p - 1) = 1$; set

$$r \stackrel{\text{def}}{=} g^u y^v \pmod{p},$$

$$s \stackrel{\text{def}}{=} v^{-1}(-r) \pmod{p - 1},$$

$$m \stackrel{\text{def}}{=} uv^{-1}(-r) \pmod{p - 1};$$

then $(m, (r, s))$ is indeed a valid “message”-signature pair for the ElGamal signature scheme, since

$$\begin{aligned} y^r r^s &\equiv y^r r^{[v^{-1}(-r)]} \\ &\equiv y^r (g^u y^v)^{[v^{-1}(-r)]} \\ &\equiv y^r (g^u)^{[v^{-1}(-r)]} (y^v)^{[v^{-1}(-r)]} \\ &\equiv y^r g^{[uv^{-1}(-r)]} y^{-r} \\ &\equiv g^m \pmod{p}. \end{aligned}$$

However, in this forgery, “message” m is not recognizable. Thus, a proper

Algorithm 9.3: The ElGamal Signature Scheme**Key Setup**

The key setup procedure is the same as that for the ElGamal cryptosystems (see §??).

(* Thus, user Alice's public-key material is a tuple (g, y, p) where p is a large prime number, $g \in \mathbb{F}_p^*$ is a multiplicative generator element (since p is prime, we will from now on use \mathbb{Z}_p^* in place of \mathbb{F}_p^*), and $y \equiv g^x \pmod{p}$ for a secret integer $x < p - 1$. Alice's private key is x . *)

Signature Generation

To create a signature on message $m \in \mathbb{Z}_p^*$, Alice picks a random number $k \in_U \mathbb{Z}_{p-1}^*$ (i.e., $k < p - 1$ and $\gcd(k, p - 1) = 1$) and creates a signature pair (r, s) where

$$\begin{aligned} r &\stackrel{\text{def}}{=} g^k \pmod{p}, \\ s &\stackrel{\text{def}}{=} k^{-1}(m - xr) \pmod{p - 1}. \end{aligned} \tag{9.3.2}$$

(* Here, the calculation of k^{-1} can use the extended Euclid's algorithm (Algorithm 4.2). *)

Signature Verification

(* Let Bob be a verifier who knows that the public-key material (g, y, p) belongs to Alice *)

Given a message-signature pair $(m, (r, s))$, Bob's verification procedure is

$$\text{Verify}_{(g, y, p)}(m, (r, s)) = \text{True} \text{ iff } y^r r^s \equiv g^m \pmod{p}.$$

Figure 9.4.

message formatting mechanism can defeat this forgery. We shall introduce such a mechanism in a later chapter.

9.3.3.2 Caution for the ephemeral key

Similar to the case of the ElGamal encryption: the ElGamal signature generation is also a randomized algorithm. The randomization is due to the random value k which is called the **ephemeral key**.

Alice should never reuse an ephemeral key in different instances of signature issuance. If an ephemeral key k is reused to issue two signatures for two messages

$m_1 \neq m_2 \pmod{p-1}$, then from the second equation in (9.3.2), we have

$$k(s_1 - s_2) \equiv m_1 - m_2 \pmod{p-1}.$$

Since $k^{-1} \pmod{p-1}$ exists, $m_1 \neq m_2 \pmod{p-1}$ implies

$$k^{-1} \equiv (s_1 - s_2)/(m_1 - m_2) \pmod{p-1},$$

i.e., k^{-1} is disclosed. In turn, Alice's private key x can be computed from the second equation in (9.3.2) as

$$x \equiv (m_1 - ks_1)/r \pmod{p-1}.$$

Notice also that the ephemeral key need to be uniformly random in \mathbb{Z}_{p-1}^* . A particular caution should be taken when a signature is generated by a small device such as a smart-card: one must make sure that such devices should be equipped with adequately reliable randomness source.

As long as k is used once only per signature and is generated at uniformly random, the second equation for signature generation (9.3.2) shows that it essentially provides a one-time multiplication cipher to encrypt the signer's private key x . Therefore, these two secrets protect one another in an information-theoretical secure sense.

9.3.4 ElGamal-like Signature Schemes

After ElGamal's original work, several variations to the ElGamal signature scheme emerged. Two influential ones are the Schnorr signature scheme [Sch90], [Sch91] and the Digital Signature Standard (DSS) [NIS91], [NIS94].

9.3.4.1 The Schnorr signature

The Schnorr signature scheme is a variation of the ElGamal signature scheme but possesses a feature which forms an important contribution to public-key cryptography: a considerably shortened representation of prime field elements without having degenerated the underlying intractable problem (which is the DLP, see §8.2.2). This idea is later further developed to finite fields of a more general form in a new cryptosystem: the XTR public-key system [LV00], which we shall introduce in a later chapter.

The shortened representation is realized by constructing a field \mathbb{F}_p such that it contains a much smaller subgroup of prime order q . We notice that the current standard parameter setting for p in ElGamal-like cryptosystems is $p \approx 2^{1024}$. We should further notice that the size for p is likely to grow to suit the advances in solving the DLP. However, since Schnorr's work, the standard parameter setting for q is $q \approx 2^{160}$ and it is quite possible that this setting be more or less a constant

regardless of the growth of the size of p . This is because that the subgroup information does not play a role in general methods for solving the DLP in \mathbb{F}_p , even the target element is known to be in the given subgroup. The constant-ish 2^{160} setting for q is merely imposed by the lower-bound requirement due to the square-root attack (see §3.5).

The Schnorr signature scheme is specified in Algorithm 9.4

Notice that in the setting-up of public parameters, a generator g can be found quickly. This is because

$$\text{Prob}[\gcd(\text{ord}(f), q) = 1 \mid f \in_U \mathbb{Z}_p^*] \leq 1/q,$$

i.e., the probability for encountering $g = 1$ is negligibly small. By Fermat's little theorem (Theorem 6.10), we have

$$g^q \equiv 1 \pmod{p}.$$

There $\langle g \rangle$ is indeed a subgroup of q elements.

The signature verification works correctly because if $(m, (s, e))$ is a valid message-signature pair created by Alice, then

$$r' \equiv g^s y^e \equiv g^{xe+k} y^e \equiv y^{-e} g^k y^e \equiv g^k \equiv r \pmod{p}.$$

As we have discussed earlier, working in the order- q subgroup of \mathbb{F}_p , a signature in the Schnorr signature scheme is much shorter than that of a signature in the ElGamal signature scheme: $2|q|$ bits are required for transmitting a Schnorr signature, in comparison with $2|p|$ bits for transmitting an ElGamal signature. The shortened signature also means fewer operations in signature generation and verification: $O_B(\log_2 q \log^2 p)$ in Schnorr vs $O_B(\log^3 p)$ in ElGamal. Further notice that in signature generation, the modulo p part of the computation can be conducted in an off-line manner. This consideration makes signature generation to only cost one multiplication modulo q , which is suitable for a small device to perform.

9.3.4.2 Caution for the ephemeral key

Similar to the case of the ElGamal signature, the ephemeral key k should never be reused, and should be uniformly random. Under these conditions, the ephemeral key and the signer's private key protect one another in an information-theoretical secure sense.

9.3.4.3 The Digital Signature Standard (DSS)

In August 1991, the US standards body, National Institute of Standards and Technology (NIST), announced a new proposed digital signature scheme called the Digital Signature Standard (DSS) [NIS91], [NIS94]. The DSS is essentially the ElGamal

Algorithm 9.4: The Schnorr Signature Scheme**Setup of Public Parameters**

1. Set up two prime numbers p and q such that $q|p-1$.
(* Typical sizes for them are: $|p| = 1024$ and $|q| = 160$. *)
2. Set up an element $g \in \mathbb{Z}_p^*$ of order q . This can be achieved by picking $f \in_U \mathbb{Z}_p^*$ and setting $g \stackrel{\text{def}}{=} f^{(p-1)/q} \pmod{p}$. If $g = 1$, then change an f and repeat the process until $g \neq 1$.
3. Set up a cryptographic hash function $h : \{0, 1\}^* \mapsto \mathbb{Z}_q$

(* (for example, SHA-1 is a good candidate.) The public parameters (p, q, g, h) can be shared by system-wise users. *)

Setup of a Principal's Public/Private Key

User Alice picks a random number $x \in_U \mathbb{Z}_q$ as her private key, and computes her public key by

$$y \stackrel{\text{def}}{=} \alpha^{-x} \pmod{p}.$$

(* Alice's public-key material is (p, q, g, y, h) ; her private key is $x \in \mathbb{Z}_q$. *)

Signature Generation

To create a signature on message $m \in \{0, 1\}^*$, Alice picks a random number $k \in_U \mathbb{Z}_q$ and computes a signature pair (e, s) where

$$\begin{aligned} r &\stackrel{\text{def}}{=} g^k \pmod{p}, \\ e &\stackrel{\text{def}}{=} h(m \parallel r), \\ s &\stackrel{\text{def}}{=} xe + k \pmod{q}. \end{aligned}$$

Signature Verification

(* Let Bob be a verifier who knows that the public-key material (p, q, g, y, h) belongs to Alice. *)

Given a message-signature pair $(m, (e, s))$, Bob's verification procedure is

$$\begin{aligned} r' &\stackrel{\text{def}}{=} g^s y^e \pmod{p}, \\ e' &\stackrel{\text{def}}{=} h(m \parallel r'), \\ \text{Verify}_{(p,q,g,y,h)}(m, (s, e)) &= \text{True} \text{ iff } e' = e. \end{aligned}$$

Figure 9.5.

signature scheme, but like the Schnorr signature scheme, it works in a much smaller prime-order subgroup of a larger group in which the DLP is believed to be hard. Therefore, the DSS has a much reduced signature size than that for the ElGamal signature scheme.

The DSS is specified in Algorithm 9.5

The signature verification works correctly because if $(m, (r, s))$ is a valid message-signature pair created by Alice, then

$$g^{u_1}y^{u_2} \equiv g^{h(m)s^{-1}}y^{rs^{-1}} \equiv g^{(h(m)+xr)s^{-1}} \equiv g^k \pmod{p};$$

comparing the right-hand side with the first equation for signature generation, this congruence should return r if is further operated modulo q .

The communication bandwidth and the computational requirements for the DSS are the same as those for the Schnorr signature scheme if the public parameters of these two schemes have the same size.

9.3.4.4 Message recognizability and caution for the ephemeral key

The DSS has been standardized together with a compatible standardization process for its hash function, namely SHA-1 [NIS95]. The use of the a standard hash function provides the needed property for message recognizability. However, the use the hash function in the straightforward message hashing manner, even the hash function being a properly standard one, cannot enable us to reach a rigorous argument for relating the difficulty for signature forgery to the standard difficulty assumption of the underlying cryptographic problem (i.e., the DLP, see Definition 8.2). Such an augment is desirable, however is still not available. We believe that a different way of using the hash function (like a padding method which we will study in a later chapter for RSA and Rabin, also see discussion in §9.3.5) will enable us to reach such an argument.

Finally, the caution for the ephemeral key is also necessary as in all ElGamal-like signature scheme.

9.3.5 Security of Digital Signature Schemes

Analogous to the discussion on security proof for the encryption algorithms that we have given in §??, we should also provide a brief discussion on the issue of provable security for digital signature schemes.

Review Remark 9.1, it is reasonable to think that forgery of a signature “from scratch” should be harder than doing the job by using some other message-signature pairs which an attacker may have in possession, or even by manipulating communications which may enable the attacker to use the targeted signer as an oracle service provider (see §8.3.2). The forgery method using some available message-signature

Algorithm 9.5: The Digital Signature Standard**Setup of Public Parameters**

(* The setting-up for the system public parameters is identical to those in the Schnorr signature scheme. Thus, parameters (p, q, g, h) , which have the same meaning as those in Algorithm ??, are shared by the system-wise users. *)

Setup of a Principal's Public/Private Key

User Alice picks a random number $x \in_U \mathbb{Z}_q$ as her private key, and computes her public key by

$$y \stackrel{\text{def}}{=} \alpha^x \pmod{p}.$$

(* So Alice's public-key material is (p, q, g, y, h) , and her private key is x . *)

Signature Generation

To create a signature on message $m \in \{0, 1\}^*$, Alice picks a random number $k \in_U \mathbb{Z}_q$ and computes a signature pair (r, s) where

$$\begin{aligned} r &\stackrel{\text{def}}{=} (g^k \pmod{p}) \pmod{q}, \\ s &\stackrel{\text{def}}{=} k^{-1}(h(m) + xr) \pmod{q}. \end{aligned}$$

Signature Verification

(* Let Bob be a verifier who knows that the public-key material (p, q, g, y, h) belongs to Alice *)

Given a message-signature pair $(m, (r, s))$, Bob's verification procedure is

$$\begin{aligned} w &\stackrel{\text{def}}{=} s^{-1} \pmod{q}, \\ u_1 &\stackrel{\text{def}}{=} h(m)w \pmod{q}, \\ u_2 &\stackrel{\text{def}}{=} rw \pmod{q}, \\ \text{Verify}_{(p,q,g,y,h)}(m, (r, s)) &= \text{True} \text{ iff } r = (g^{u_1} y^{u_2} \pmod{p}) \pmod{q}. \end{aligned}$$

Figure 9.6.

pairs is called **existential forgery**, and the forgery method using manipulation and oracle services is called **adaptive forgery**. Since in reality, message-signature pairs regarding a given public key are abundantly available; also since the vulnerability of the open computing, communications network means ease of adaptive attacks, we should anticipate the easier ways of attacks such as existential forgery

or adaptively chosen message forgery; namely, the stronger notion of security for digital signatures is indispensable.

We have also seen that it is generally easy to forge a “message”-signature pair, even to forge it “from scratch” if the “message” is not recognizable (see §9.3.1.1 and §9.3.3.1 for general need of message recognizability in digital signature schemes). To prevent such easy ways of forgery, any digital signature scheme must be equipped with a mechanism to transform the message to be signed into one which is recognizable. The most frequently realization of the message-recognizability mechanism is cryptographic hash functions. It is thus reasonable to expect that the security argument for a digital signature scheme is reasoned about together with the behavior of cryptographic hash functions.

Indeed, in absence of a clearly stated behavior of hash functions, we have not been able to provide any valid argument on the security for the various signature schemes introduced so far as they all use hash functions. Such a valid argument should relate the difficulty for signature forgery to a relevant difficult problem which underlies the public key cryptosystem. In Chapter ?? we shall consider a reasonable model for the behavior of cryptographic hash functions used in many practical digital signature schemes. That model, named **random oracle model** (see §14.2.1 and §14.2.2), idealizes a hash function into one having so-called “random oracle behavior” with which we will be able to relate the difficulty for signature forgery to some well-known computational assumptions which base the underlying public-key cryptosystem.

9.4 Asymmetric Techniques II: Data Integrity Without Source Identification

In a data integrity mechanism realized by a digital signature scheme, the usual setting for key parameters stipulates that K is a private key and KV is the matching public key. Under this setting, a correct integrity verification result of a message provides the message verifier the identity of the message transmitter who is the signer of the message, i.e., the owner of the public key KV .

We should notice however that this “usual setting for key parameters”, while being a necessary element for achieving a digital signature scheme, is unnecessary for a data-integrity system. In fact, in Definition 9.1 we have never put any constraint on the two keys for constructing and for verifying MDC.

Thus, for example, we can actually set the two keys, K and KV , opposite to that for a digital signature scheme, that is, let K be a public key and KV be a private key. Under such a key setting, anybody is able to use the public key K to create a consistent (i.e., cryptographically integral) pair (Data, MDC) or a “message-signature pair” (m, s) , while only the holder of the private key KV is able to verify the consistency of the pair (Data, MDC) or the validity of the “signature”

(m, s) . Of course, under such an unusual key setting, the system can no longer be regarded to be a digital signature scheme. However, we must notice that, according to Definition 9.1, the system under such an unusual key setting remains to be a data-integrity system!

Since anybody can have used the public key K to have created the consistent pair (Data, MDC), we shall name this kind of data-integrity system **data-integrity without source identification**. From our familiarity with the behavior of Malice (the bad guy), there is no danger for us to conveniently rename this data-integrity service “*data integrity from Malice*”.

Let us now look at an example of a public-key encryption scheme which provides a “data-integrity-from-Malice” service. This is scheme with the following property: Malice can send to Bob a confidential message such that the message is “non-malleable” (e.g., by other friends of Malice) that is, it’s computationally difficult for any other member in the clique of Malice to modify the message without being detected by Bob, the receiver. This algorithm, with its RSA instantiation being specified in Algorithm 9.6, is named **Optimal Asymmetric Encryption Padding (OAEP)** and is invented by Bellare and Rogaway [BR95a].

If the ciphertext has not been modified after its departure from the sender, then from the encryption algorithm we know that Bob will retrieve the random number r correctly, and therefore

$$v = s \oplus G(r) = (m \parallel 0^{k_1}) \oplus G(r) \oplus G(r) = m \parallel 0^{k_1}.$$

Therefore, Bob will see k_1 zeros trailing the retrieved plaintext message.

On the other hand, any modification on the ciphertext will cause an alteration to the message sealed under the RSA function. This alteration will further cause “uncontrollable” alteration to the plaintext message, including the random input and the redundancy of k_1 zeros trailing the plaintext message, which have been input to the OAEP function. Intuitively, the “uncontrollable” alteration is due to a so-called “random oracle” property of the two hash functions used in the scheme (see our discussions for random oracle in §9.2.1.1). The uncontrollable alteration will show itself up by damaging the redundancy (the string of k_1 zeros) added into the plaintext with a probability at least $1 - 2^{-k_1}$. Given 2^{-k_1} being negligible, $1 - 2^{-k_1}$ is significant. Thus, indeed, the scheme provides a data-integrity protection on the encrypted message.

Notice that this data-integrity protection is a strange one: although upon seeing the string of k_1 zeros Bob is assured that the ciphertext has not been modified, he can have no idea who the sender is. That is why in Algorithm 9.6 we have deliberately specified Malice to be the sender. The notion of “integrity from Malice” is very important. This notion became apparent as a result of advances in public-key encryption schemes secure with respect to adaptively chosen ciphertext attack (CCA2, see Definition 8.3). In a public-key cryptosystems secure with respect to CCA2, the decryption procedure includes a data-integrity verification step. Such a

Algorithm 9.6: Optimal Asymmetric Encryption Padding for RSA (RSA-OAEP) [BR95a]

Key Parameters

Let (N, e) be Bob's RSA public key encryption parameters with

$$k = |N| = n + k_0 + k_1$$

where k_0 and k_1 are another two parameters with $k_0 + k_1 < k$ satisfying that 2^{-k_0} and 2^{-k_1} are negligible quantities; n is the length for the plaintext message.

Let G, H be two cryptographic hash functions satisfying

$$G : \{0, 1\}^{k_0} \mapsto \{0, 1\}^{k-k_0}, \quad H : \{0, 1\}^{k-k_0} \mapsto \{0, 1\}^{k_0}.$$

Encryption

To send a message $m \in \{0, 1\}^n$ to Bob, Malice picks $r \in_U \{0, 1\}^{k_0}$, and operates the following steps:

1. $s \stackrel{\text{def}}{=} (m \parallel 0^{k_1}) \oplus G(r);$
2. $t \stackrel{\text{def}}{=} r \oplus H(s);$
3. $c \stackrel{\text{def}}{=} (s \parallel t)^e \pmod{N}.$

The ciphertext is c .

(* here, " \parallel " denotes the bit-string concatenation, " \oplus ", the bit-wise XOR operation, and " 0^{k_1} ", the string of k_1 zeros functioning as redundancy for data-integrity checking in decryption time *)

Decryption

Upon receipt of the ciphertext c , Bob performs the following steps:

1. $s \parallel t \stackrel{\text{def}}{=}} c^d \pmod{N}$ satisfying $|s| = n + k_1 = k - k_0, |t| = k_0;$
2. $u = t \oplus H(s);$
3. $v = s \oplus G(u).$

4. output $\begin{cases} m & \text{if } v = m \parallel 0^{k_1} \\ \text{REJECT} & \text{otherwise} \end{cases}$

(* when REJECT is output, the ciphertext is deemed invalid *)

Figure 9.7. The RSA-OAEP scheme

cryptosystem is considered to be invulnerable even in the following extreme form of abuse by an attacker:

- The attacker and a public-key owner play a challenge-response game. The attacker is in the position of a challenger and is given freedom to send, as many as he wishes (of course the attacker is polynomially bounded), “*adaptively chosen ciphertext*” messages to the owner of the public key for decryption in an oracle-service manner (review “oracle service” in 8.3.2).
- The owner of the public key is in the position of a responder. If the data-integrity verification in the decryption procedure passes, the key owner should simply send the decryption result back regardless of the fact that the decryption request may even be from an attacker who may have created the ciphertext in some clever and unpublicized way with an intention to break the target cryptosystem (either to obtain a plaintext message which the attacker is not entitled to see, or to discover the private key of the key owner).

If a ciphertext has the correct data integrity, then it is considered that the sender should have already known the plaintext encrypted in. This is a notion known as “**plaintext awareness**”. If the attacker has already known the encrypted plaintext, then an oracle decryption service should provide him no new information, not even in terms of providing him with a cryptanalysis training for how to break the target cryptosystem.

In Chapter 13 we will introduce a formal model for capturing the security notion under adaptively chosen ciphertext attack (CCA2). We will also study some public-key cryptosystems which are formally provably secure with respect to such attacks in Chapter 14. The RSA-OAEP is one of them. In §14.2 we shall provide a detailed analysis on the security of the RSA-OAEP encryption scheme. The analysis will be a formal proof that the RSA-OAEP is secure under a very strong attacking scenario: indistinguishability against an adaptively chosen ciphertext attacker. Due to this stronger security quality, the RSA-OAEP is no longer a textbook encryption algorithm; it is a fit-for-application cryptosystem.

As having been shown in the RSA-OAEP algorithm, the usual method to achieve a CCA2-secure cryptosystem is to have the cryptosystem include a data-integrity checking mechanism without having the least concern of **message source identification**.

Message source identification is part of authentication service called data-origin authentication. Authentication is the topic for the next Chapter.

9.5 Chapter Summary

In this chapter we have introduced the basic cryptographic techniques for providing data-integrity services. These techniques include symmetric techniques based on us-

ing MACs constructed from hash functions and from block ciphers, and asymmetric techniques based on digital signatures. Data-integrity served by these techniques comes together with a sub-service: message source identification.

We also identify a peculiar data-integrity service which does not come together with identification of the message source, and exemplify with a public-key cryptosystem which makes use of this service. In a later chapter we will see that this data-integrity service forms a general method for achieving “fit-for-application” cryptosystems.

Part IV

Authentication

If business transactions, electronic commerce, or confidential communications are conducted on an open communications network, the bona-fide-ness of the intended communication partners and that of messages must be established. The security service needed here is authentication which can be obtained by applying cryptographic techniques. This part has three chapters on various protocol techniques of authentication. In Chapter 10 we study authentication protocols on their basic working principles, examine typical errors in authentication protocols and investigate causes. In Chapter 11 we case-study several important authentication protocol techniques applied in the real world. In Chapter 12 we introduce the authentication framework for public-key infrastructure.

AUTHENTICATION PROTOCOLS — PRINCIPLES

10.1 Introduction

In Chapter 2 we have exposed ourselves to a number of authentication protocols. Most protocols there are fictional ones (with two exceptions): we have deliberately designed them flawed in several ways in order for them to serve the purpose of an introduction to a culture of caution and vigilance in the areas of cryptography and information security.

In this chapter we return to the topic of authentication. The purpose of returning to the topic is for us to have a more comprehensive study of the area. Our study in this chapter will be divided into two categories:

An introduction to various authentication techniques

In this category we shall study various basic techniques for authentication. These include the very basic mechanisms and protocol constructions for message and entity authentication, password-based authentication techniques and some important authenticated key establishment techniques. We believe that a number of basic authentication mechanisms and protocol constructions in several international standards are the ones which have been selected from the literature and subsequently gone through a careful (and long) process of expert review and improvement revision. Therefore, in our introduction to the basic authentication techniques, we shall pay a particular attention to the mechanisms which have been standardized by international standard bodies. In addition, we shall introduce a few other reputable authentication and authenticated key establishment protocols. We believe that authentication mechanisms and protocols introduced in this category have a value for serving building blocks and guidelines for designing good protocols. We therefore consider that this category provides the model authentication techniques for protocol designers.

An exemplified study of a wide range of protocol flaws

This is an inevitable part in the subject of authentication. We shall list various known and typical attacking techniques which can be mounted on authentication protocols. We shall analyze and discuss each attacking technique using some flawed protocols with the applicable attacks demonstrated. Through this study, we shall become familiar with a common phenomenon that authentication protocols are ready to contain security flaws even they have been designed by experts. The comprehensive list of typical protocol flaws and the related attacking techniques provide an essential knowledge for a protocol designer: “Did you know this sort of attack?”.

Unlike in the cases of Chapter 2 where we have deliberately designed fictional protocols with artificial flaws, the security flaws in the protocols to be demonstrated in this chapter are not artificial ones; indeed, none of them is! These flaws are all discovered after the flawed protocols have been published by reputable authors in information security and/or cryptography. A fact we shall see through the study in this chapter is that, even conforming standard documents, following well-thought design principles, and even having been well familiar with many typical protocol flaws, design of authentication protocol remains to be extremely error-prone, even for experts in the areas.

Due to the notorious error-prone nature of authentication protocols, this chapter plus the next, as follow-up of Chapter 2, are still not an end for the topic of authentication in this book. Systematic approaches (i.e., formal methods) to the development of correct authentication protocols are currently serious research topics. We shall study the topics of formal approaches to correct authentication protocols in chapter 15.

10.1.1 Chapter Outline

In §10.2 we discuss the notion of authentication by introducing several refined notions. In §10.3 we agree on conventions for expressing components in authentication protocol and for the default behavior of protocol participants. The next three sections form the first category our study in this chapter: in §10.4 we study the very basic and standard constructions for authentication protocols; in §10.5 we study some password based authentication techniques, and in §10.6 we study an important protocol which achieves authentication and authenticated key exchange using cryptographic techniques which is alternative to those used in the previous two sections. The second category of our study is contained in §10.7 where we list and demonstrate typical attacking techniques applicable on authentication protocols. Finally, we end this chapter in §10.8 by recommending a brief but important literature references in the area.

10.2 Authentication and Refined Notions

For a very short description for authentication, we may say that it is a procedure by which an entity establishes a claimed property to another entity. For example, the former is a subject claiming a legitimate entry to or use of the latter which is a system or a service, and by authentication, the latter establishes the claimed legitimacy. From this short description we can already see that authentication involves at least two separate entities in communication. Conventionally, a communication procedure run between or among co-operative principals is called a protocol. An authentication procedure is hence an authentication protocol.

The notion of authentication can be broken down to three sub-notions: **data-origin authentication**, **entity authentication** and **authenticated key establishment**. The first concerns validating a claimed property of a message; the second pays more attention on validating a claimed identity of a message transmitter; and the third further concerns to output a secure channel for a subsequent, application-level secure communication session.

10.2.1 Data-Origin Authentication

Data-origin authentication (also called **message authentication** relates closely to data integrity. Early textbooks in cryptography and information security viewed these two notions with no essential difference (e.g., Chapter 5 of [DP89] and §1.2-§1.3 of [Den82]). Such a view was based on a consideration that using information which has been modified in a malicious way is at the same risk as using information which has no reputable source.

However, data-origin authentication and data integrity are two *very* different notions. They can be clearly differentiated from a number of aspects.

First, data-origin authentication necessarily involves communications. It is a security service for a message receiver to verify whether a message is from a purported source. Data integrity needn't have a communication feature: the security service can be provided on stored data.

Secondly, data-origin authentication necessarily involves identifying the source of a message, while data integrity needn't do so. In §9.4, we have shown and argued with a convincing example that data integrity as a security service can be provided without message source identification. We have even coined a phrase “data integrity from Malice” to label a data-integrity service with such a property. We should remind the reader that according to our stipulation made in Chapter 2 Malice is a faceless principal whose identity has the least to do with a reputable source of a message. In Chapter 14 we shall realize that “data integrity from Malice” is a general mechanism for achieving a provably secure public-key cryptosystems.

Thirdly and the most significantly, data-origin authentication necessarily involves to establish **liveness** of a message transmitter, while, again, data integrity

needn't do so: a piece of stale can have a perfect data integrity. To obtaining data-origin authentication service, a message receiver should verify whether or not a claimed message transmitter has sent the message sufficiently *recently* (that is, the time interval between the message issuance and its receipt is sufficiently small). A message which is deemed by the receiver to have been issued sufficiently recently is often referred to as a **fresh** message. Requiring a message to be fresh follows a common sense that a fresh message implies a good *correspondence* between the communication principals, and this may further imply a less likelihood that, e.g., the communication principals, apparatus, systems, or the message itself may have been sabotaged. In §2.5.4 we have seen an attack on the Needham-Schroeder Symmetric-key Authentication Protocol (the attack of Denning and Sacco, Example 2.2) in which a replayed old message has absolutely valid data integrity but has invalid authenticity. Authentication failure of this kind can be referred to as a *valid data integrity without liveness of the message source*.

Notice that whether or not a message is fresh should be determined by applications. Some applications require a rather short time interval for a message to be fresh which can be a matter of seconds (as in many challenge-response based real-time secure communication applications). Some applications allows a longer freshness period; for example, in World War II, the German military communications encrypted by the famous Enigma machine stipulated a rule that each day all Enigma machines must be set to a new “day-key” [Sin99]. This rule has become a widely used key-management principle for many security systems today, though “day-key” may have been changed to “hour-key” or even “minute-key”. Some other applications permit a much longer time interval for message freshness; for example, let a bank cheque have passed examinations in terms of its integrity and source identification; then its validity (authenticity) for authorizing the payment should be determined by the age of the cheque, that is, the time interval between the date of the cheque's issuance and that of the cheque's deposit; most banks permit three months as the valid age for a cheque.

Finally, we point out that some anonymous credential enabled by some cryptographic schemes (e.g., blind signature) also provide a good differentiation between data-origin authentication and data integrity. A user can be issued an anonymous credential which enables the holder to gain a service by proving a membership to a system anonymously. Notice that here, the data integrity evidence can even be demonstrated in a lively correspondent fashion, however, the system is prevented from performing source identification. We will study such cryptographic techniques in a later chapter.

From our discussions so far, we can characterize the notion of data-origin authentication as follows:

- i) it consists of transmitting a message from a purported source (the transmitter) to a receiver who will validate the message upon reception;

- ii) the message validation conducted by the receiver concerns to establish the identity of the message transmitter;
- iii) the validation also concerns to establish the data integrity of the message subsequent to its departure from the transmitter;
- iv) the validation further concerns to establish liveness of the message transmitter.

10.2.2 Entity Authentication

Entity authentication is a communication process (i.e., protocol) by which a principal establishes a *lively correspondence* with a second principal whose claimed identity should meet what is sought by the first. Often, the word “entity” is omitted, as in the following statement: “an important goal of an authentication protocol is to establish lively correspondence of a principal”.

Often, a claimed identity in a protocol is a protocol message in its own right. In such a situation, confidence about a claimed identity and about the liveness of the claimant can be established by applying data-origin authentication mechanisms. Indeed, as we shall see in many cases in this chapter, for a claimed identity being in the position of a protocol message, treating it as a subject of data-origin authentication does form a desirable approach to entity authentication.

There are several types of entity authentication scenarios in distributed systems depending on various ways of classifying principals. We list several usual scenarios which are by no means to be exhaustive.

Host-host type Communication partners are computers called “nodes” or platforms in a distributed system. Host-level activities often require cooperation among them. For example, in remote “reboot”^a of a platform, upon reboot, the platform must identify a trusted server to supply necessary information, such as a trusted copy of an operating system, trusted clock setting, or the current trusted environment settings. The establishment of the trusted information is usually achieved via running an authentication protocol. A customary case in this the host-host type of communication is a **client-server** setting where one host (client) requests for certain services from the other (server).

User-host type A user gains access to a computer system by logging in a host in the system. The simplest examples are to login to a computer via telnet, or to conduct file transfer via ftp (file transfer protocol); both can be achieved via running a password authentication protocol. In a more serious application where a compromised host will cause a serious loss (e.g., when a user to

^a “Reboot” is a technical term in computer science for re-initialization of a computer system from some simple preliminary instructions or a set of information which may be hardwired in the system.

make an electronic payment via a smart card), **mutual authentication** is necessary.

Process-host type Nowadays distributed computing has been so highly advanced that a great deal of functionalities and services are possible. A host may grant a foreign process various kinds of access rights. For example, a piece of “mobile code” or a “JavaTM applet”^b can travel to a remote host and run on it as a remote process. In sensitive applications, it is necessary and possible to design authentication mechanisms so that an “applet” can be deemed to be a friendly one by a host and be granted an appropriate access right on it.

Member-club type A proof of holding a credential by a member to a club can be viewed as a generalization of the “user-host type”. Here a club may need only to concern the validation of the member’s credential without necessarily knowing further information such as the true identity of the member. Zero-knowledge identification protocols, undeniable signature schemes can enable this type of entity authentication scenario. We shall study these authentication techniques in a few later chapters.

10.2.3 Authenticated Key Establishment

Often, communication partners run an entity authentication protocol as a means to bootstrap further secure communications at a higher or application level. In modern cryptography, cryptographic keys are the basis for secure communication channels. Therefore, entity authentication protocols for bootstrapping higher or application-level secure communications generally feature a sub-task of (authenticated) key establishment, or **key exchange**, or **key agreement**.

Like in the case that entity authentication can be based on data-origin authentication regarding the identity of a claimant, in protocols for authenticated key establishment, key establishment material also forms important protocol messages which should be the subject for data-origin authentication.

In the literature, (entity) authentication protocols, authenticated key establishment (key exchange, key agreement) protocols, security protocols, or sometimes even cryptographic protocols, often refer to the same set of communication protocols.

10.2.4 Attacks on Authentication Protocols

Since the goal of an authentication protocol (data-origin, entity, key establishment) is to establish a claimed property, cryptographic techniques are inevitably used. Very inevitably also, the goal of an authentication protocol will be matched with

^bJavaTM applet is an executable code to run by a “web browser” on a remote host in order to effect a function on the issuing host’s behalf.

a counter-goal: attack. An attack on an authentication protocol consists of an attacker or a coalition of them (who we name collectively, Malice, see §2.2) achieving an un-entitled gain. Such a gain can be a serious one such as Malice obtains a secret message or key, or a less serious one such as Malice successfully deceives a principal to establish a wrong belief about a claimed property. In general, an authentication protocol is considered flawed if a principal concludes a normal run of the protocol with its intended communication partner while the intended partner would have a different conclusion.

We must emphasize that attacks on authentication protocols are mainly those which do *not* involve breaking of the underlying cryptographic algorithms. Usually, authentication protocols are insecure not because the underlying cryptographic algorithm they use are weak, it is because of protocol design flaws which permit Malice to break the goal of authentication without necessarily breaking any cryptographic algorithm. We shall see many such attacks in this chapter. Due to this reason, in the analysis of authentication protocols, we usually assume that the underlying cryptographic algorithms are “*perfect*” without considering their possible weakness. Those weakness are usually considered in other subjects of cryptography.

10.3 Convention

In authentication protocols to appear in the rest of this chapter, we will use the following convention for the semantical meanings of some protocol messages according to their syntactic structures:

- $Alice, Bob, Trent, Malice, \dots$: principal names appeared as protocol messages; sometimes they may be abbreviated to A, B, T, M, \dots ;
- $Alice \rightarrow Bob: M$; Alice sends to Bob message M ; a protocol specification is a sequence of several such message communications;
- $\{M\}_K$: a ciphertext which encrypts the message M under the key K ;
- $K, K_{AB}, K_{AT}, K_A, \dots$: cryptography keys, where K_{XY} denotes a key shared between principals X and Y , and K_X denotes a public key of principal X ;
- N, N_A, \dots : nonces, which stands for “numbers use for once” [BAN89]; these are random numbers sampled from a sufficiently large space; N_X is generated by principal X ;
- T_X : a timestamp created by principal X ;
- $sig_A(M)$: a digital signature on message M created by principle A .

Remark 10.1: *However, we should notice that the semantical meanings of protocol messages which are associated to their syntactic structures (types) as above are not*

necessarily comprehensible by a protocol participant (say Alice). In general, for any message or part of a message in a protocol, if the protocol specification does not require Alice to perform cryptographic operation on that message or message part, then Alice (in fact, her protocol compiler) will only understand that message part at the syntactic level. At the syntactic level, Alice may misinterpret the semantical meanings of a protocol message. We exemplify various possibilities of misinterpretations in Example 10.1. \square

Example 10.1: At the syntactic level, Alice may make wrong interpretations on protocol messages. The following are a few examples:

- She may consider a message chunk as a ciphertext and may try to decrypt it if she thinks she has the right key, or forward the message chunk to Bob if she thinks that the chunk is for him. However, the message chunk may in fact be a principal's identity (e.g., *Alice* or *Bob*) or a nonce;
- She may try to “verify” a nonce or a timestamp as if the message part is a digital signature;
- She may view a piece of public key material as a nonce; etc.

It seems that Alice is very “stupid” in understanding of protocol messages. No, we should rather consider that she is too innocent and cannot always anticipate the existence of “clever” Malice who may have already “re-compiled” a protocol by misplacing various message parts in order to cause the misinterpretation. \square

In general, we have the following further convention for the behavior of a protocol participant, whether a legitimate one or an uninvited one:

- An honest principal in a protocol does not understand the semantical meanings of any protocol message before a run of the protocol terminates successfully.
- An honest principal in a protocol cannot recognize $\{M\}_K$ or create it or decompose it unless the principal has in its possession the correct key; here the meaning of “recognize a message” follows our agreement made in §9.3.1.1.
- An honest principal in a protocol cannot recognize a random-looking number such as a nonce, a sequence number or a cryptographic key, unless the random-looking number either has been created by the principal in the current run of the protocol, or is an output to the principal as a result of a run of the protocol.
- An honest principal in a protocol does not record any protocol messages unless the protocol specification instructs so. In general, an authentication protocol

is *stateless*, that is, it does not require a principal to maintain any state information after a protocol run terminates, except for the information which is deemed to be the output of the protocol to the principal.

- Malice, in addition to his power specified in §2.2, knows the “stupidities” (to be more fair, the weaknesses) of honest principals which we have exemplified in Example 10.1, and will always try to exploit them in mounting his attack.

Authentication protocols are meant to transmit messages in a public communication network, which is assumed to be under Malice’s control, and to thwart his attacks in such an environment albeit Malice is “clever” and honest principals are “stupid”.

Now let us see how this is to be achieved.

10.4 Basic Authentication Techniques

There are numerous protocol based techniques for realizing (data-origin, entity) authentication and authenticated key establishment. However, the basic protocol constructions, in particular those which should be regarded as good ones, and the simple technical ideas behind the good constructions, are not so diverse.

In this section let us study basic authentication techniques through introducing some basic but important protocol constructions. In our study, we shall pay particular attentions on constructions which have been documented in a series of international standards. We consider that these constructions should serve the model ones for the design of authentication protocols. We shall also argue why some constructions are more desirable than others, exemplify a few bad ones and reason why they are bad.

The following basic authentication techniques will be studied in this section:

- Standard mechanisms for establishing message freshness and principal liveness (§10.4.1)
- Mutual authentication vs unilateral authentication (§10.4.2)
- Authentication involving trusted third party (10.4.3)

10.4.1 Message Freshness and Principal Liveness

To deem whether a message is fresh is a necessary part in data-origin authentication (please notice the difference between message source identification and data-origin *authentication* which we have discussed in §10.2.1), likewise in the case of entity authentication where a principal concerns lively correspondence of an intended communication partner. Therefore, mechanisms which establish message freshness or principal liveness are the most basic components in authentication protocols.

Let us now describe the basic and standard mechanisms to achieve these functions. In our descriptions, we shall let Alice be in the position of a claimant regarding a property (e.g., her liveness, or freshness of a message), and Bob be in the position of a verifier regarding the claimed property. We assume that Alice and Bob share a secret key K_{AB} if a mechanism uses symmetric cryptographic techniques, or that Bob knows Alice's public key via a public-key certification framework^c if a mechanism uses asymmetric cryptographic techniques.

10.4.1.1 Challenge-response mechanisms

In a **challenge-response mechanism**, Bob (the verifier) has his input to a composition of a protocol message and the composition involves a cryptographic operation performed by Alice (the claimant) so that Bob can verify the lively correspondence of Alice via the freshness of his own input. The usual form of Bob's input can be a random number (called a nonce) which is generated by Bob and passed to Alice beforehand. Let N_B denote a nonce generated by Bob. This message freshness mechanism has the following interactive format:

1. Bob \rightarrow Alice: N_B
2. Alice \rightarrow Bob: $\mathcal{E}_{K_{AB}}(M, N_B)$
3. Bob decrypts the cipher chunk and $\begin{cases} \text{accepts} & \text{if he sees } N_B \\ \text{rejects} & \text{otherwise} \end{cases}$ (10.4.1)

Here, the first message transmission is often called Bob's **challenge** to Alice, and the second message transmission is thereby called Alice's **response** to Bob. Bob is in a position of an **initiator** while Alice is in a position of a **responder**.

The specified mechanism uses symmetric cryptographic technique: symmetric encryption. Therefore, upon receipt of Alice's response, Bob has to decrypt the ciphertext chunk using the shared key K_{AB} . If the decryption extracts his nonce *correctly* (be careful for the meaning of "correctly", it actually means correct data integrity, to discuss in a moment) then Bob can conclude that Alice has indeed performed the required cryptographic operation *after* his action of sending the challenge; if the time interval between the challenge and the response is acceptably small (according to an application requirement as we have discussed in §10.2.1), then the message M is deemed to be fresh. The intuition behind this message freshness mechanism is a confidence that Alice's cryptographic operation must have taken place after her receipt of Bob's nonce. This is because Bob's nonce has been sampled at random from a sufficiently large space and so no one can have predicted its value before his sampling.

Now let us explain what we meant by Bob's decryption and extraction of his nonce "correctly" (as we warned in the previous paragraph). The use of symmetric encryption in this mechanism may deceptively imply that the cryptographic service

^cPublic-key certification framework will be introduced in Chapter 12.

provided here is confidentiality. In fact, the necessary security service for achieving message freshness should be data integrity. The reader might want to argue that the two principals may want to keep the message M confidential, e.g., M may be a cryptographic key to be used for securing a higher-level communication session later (and thus this basic construction includes a sub-task of session key establishment). This does constitute a legitimate reason for using encryption. We could actually further consider that the two parties may also like to keep Bob's nonce secret and so in that case Bob should also encrypt the first message transmission. Therefore, we are not saying that the use of encryption for providing the confidentiality service is wrong here provided such a service is needed. What we should emphasize here is that if the encryption algorithm does not provide a proper data-integrity service (an encryption algorithm usually doesn't), then the specified mechanism is a dangerous one because the necessary service needed here, data integrity, is missing! In §15.2.1 we shall see with convincing evidence the reason behind the following statement:

Remark 10.2: *If the encryption algorithm in authentication mechanism (10.4.1) does not offer a proper data-integrity service then Bob cannot establish freshness of the message M .* \square

The really correct and a standard approach to achieving data-integrity service using symmetric cryptographic techniques is to use a manipulation detection code (MDC, see Definition 9.1). Therefore, in mechanism (10.4.1), the encryption should be accompanied with an MDC which is keyed with a shared key and inputs the ciphertext chunk which needs integrity protection. If the message M does not need confidentiality protection, then the following mechanism is a proper one for achieving message freshness:

1. Bob \rightarrow Alice: N_B
2. Alice \rightarrow Bob: $M, \text{MDC}(K_{AB}, M, N_B)$
3. Bob reconstructs $\text{MDC}(K_{AB}, M, N_B)$ and (10.4.2)

$$\begin{cases} \text{accepts} & \text{if two MDCs are identical} \\ \text{rejects} & \text{otherwise} \end{cases}$$

Notice that in order for Bob to be able to re-construct the MDC in step 3, the message M now must be sent in cleartext in step 2. Of course, M can be a ciphertext encrypting a confidential message.

In §15.2.1 we shall argue with convincing evidence that, in terms of achieving authentication using symmetric cryptographic techniques, mechanism (10.4.2) is a correct approach while mechanism (10.4.1) is an incorrect one. There we shall also see that, without a proper data-integrity, confidentiality of M in (10.4.1) needn't be in place even if the mechanism uses a strong encryption algorithm.

The challenge-response mechanism can also be achieved by applying asymmetric

cryptographic technique, as in the following mechanism:

1. Bob \rightarrow Alice: N_B
2. Alice \rightarrow Bob: $\text{sig}_A(M, N_B)$
3. Bob verifies the signature using his nonce and (10.4.3)

$$\begin{cases} \text{accepts} & \text{if signature verification passes} \\ \text{rejects} & \text{otherwise} \end{cases}$$

Notice that in this mechanism, Alice's free choice of the message M is very important. Alice's free choice of M should be part of the measure to prevent this mechanism from being exploited by Bob to trick Alice to sign inadvertently a message of Bob's preparation. For example, Bob may have prepared his "nonce" as

$$N_B = h(\text{Transfer 1000 pounds to Bob's Acc.No. 123 from Alice's Acc.No. 456.})$$

where $h()$ is a hash function.

In some applications, a signer in the position of Alice in mechanism (10.4.3) may not have freedom to choose M . In such situations, specialized keys can be defined to confine the usages of keys. For example, the public key for verifying Alice's signature in mechanism (10.4.3) can be specified for the specific use in this mechanism. Specialization of cryptographic keys is a subject in **key management** practice.

10.4.1.2 Standardization of the challenge-response mechanisms

The ISO (the International Organization for Standardization) and the IEC (the International Electrotechnical Commission) have standardized the three challenge-response mechanisms introduced so far as the basic constructions for **unilateral entity authentication** mechanisms. The standardization for mechanism (10.4.1) is called "ISO Two-Pass Unilateral Authentication Protocol" and is as follows [ISO98a]:

1. $B \rightarrow A : R_B \parallel \text{Text1}$

2. $A \rightarrow B : \text{TokenAB}$

where $\text{TokenAB} = \text{Text3} \parallel \mathcal{E}_{K_{AB}}(R_B \parallel B \parallel \text{Text2})$.

Upon receipt of TokenAB , Bob should decrypt it; he should accept the run if the decryption reveals his nonce R_B correctly, or reject the run otherwise.

Here and below in the ISO/IEC standards, we shall use precisely the notation of the ISO/IEC for protocol specification. In the ISO/IEC specification, Text1 , Text2 , etc. are optional fields, \parallel denotes bit-string concatenation, R_B is a nonce generated by Bob.

We should remind the reader the importance for the encryption algorithm to provide data integrity service which is a necessary condition to allow testing whether or not a decryption result is correct (review Remark 10.2).

Notice also that while we regard (10.4.1) as a basic message freshness mechanism, its ISO/IEC standard version is an entity authentication mechanism. Therefore the inclusion of the message “ B ”, i.e., Bob’s identity, in place of M in (10.4.1) becomes vitally important: the inclusion makes it explicit that the ISO/IEC mechanism is for the purpose of establishing Bob’s lively correspondence, is an entity authentication protocol in which Bob is the subject of authentication. Abadi and Needham proposed a list of prudent engineering principles for cryptographic protocols design [AN95]; making explicit the identity of the intended authentication subject is an important principle in their list. In §10.7.7 we shall see the danger of omission of principal’s identity in authentication protocols.

The ISO/IEC standardization for mechanism (10.4.2) is called “ISO Two-Pass Unilateral Authentication Protocol Using a Cryptographic Check Function (CCF)”, and is as follows [ISO99]:

1. $B \rightarrow A : R_B \parallel \text{Text1}$

2. $A \rightarrow B : \text{TokenAB}$

where^d $\text{TokenAB} = \text{Text2} \parallel f_{K_{AB}}(R_B \parallel B \parallel \text{Text2})$; f is a CCF, and is essentially a cryptographic hash function. The use of the CCF here is keyed.

Upon receipt of TokenAB , B should re-construct the keyed CCF using the shared key, his nonce, his identity and Text2 ; he should accept the run if the re-constructed CCF block is identical to the received block, or reject the run otherwise.

The ISO/IEC standardization for mechanism (10.4.3) is called “ISO Public Key Two-Pass Unilateral Authentication Protocol”, and is as follows [ISO98b]:

1. $B \rightarrow A : R_B \parallel \text{Text1}$

2. $A \rightarrow B : \text{CertA} \parallel \text{TokenAB}$

where $\text{TokenAB} = R_A \parallel R_B \parallel B \parallel \text{Text3} \parallel \text{sig}_A(R_A \parallel R_B \parallel B \parallel \text{Text2})$; CertA is Alice’s public key certificate (we shall study public-key certification in the next chapter).

Upon receipt of TokenAB , B should verify the signature; he should accept the run if the verification passes, or reject the run otherwise.

As we have discussed following mechanism (10.4.3), in this ISO/IEC protocol, A ’s free choice of R_A forms part of the measure preventing A from inadvertently signing a message of B ’s preparation.

^d In [ISO99], Text2 in the cleartext part is mistaken to Text3 . Without Text2 in cleartext, B cannot verify the CCF by re-constructing it.

10.4.1.3 Timestamp mechanisms

In a **timestamp mechanism**, Alice adds the current time to her message composition which involves a cryptographic operation so that the current time is cryptographically integrated in her message.

Let T_A denote a timestamp created by Alice when she composes her message. This message freshness mechanism has the following non-interactive format:

1. Alice \rightarrow Bob: $\mathcal{E}_{K_{AB}}(M, T_A)$
2. Bob decrypts the cipher chunk and

$$\begin{cases} \text{accepts} & \text{if } T_A \text{ is deemed valid} \\ \text{rejects} & \text{otherwise} \end{cases} \quad (10.4.4)$$

Analogous to mechanism (10.4.1), the decryption performed by Bob must be tested for data-integrity correctness (review §10.4.1.1 and Remark 10.2). After decryption, Bob can compare the revealed T_A with his own time (we assume that the protocol participants use a global standard time, such as the Greenwich Mean Time). If the time difference is sufficiently small allowed by the application in Bob's mind, then the message M is deemed to be fresh.

Analogous to our criticism in §10.4.1.1 on encryption without data-integrity as misuse of security service, a more desirable version of the timestamp mechanism using symmetric cryptographic techniques should be as follows:

1. Alice \rightarrow Bob: $M, T_A, \text{MDC}(K_{AB}, M, T_A)$
2. Bob reconstructs $\text{MDC}(K_{AB}, M, T_A)$ and

$$\begin{cases} \text{accepts} & \text{if two MDCs are identical and } T_A \text{ is deemed valid} \\ \text{rejects} & \text{otherwise} \end{cases} \quad (10.4.5)$$

In this version, Bob performs data-integrity validation by checking a one-way transformation style of cryptographic integration between the timestamp and message. Of course, if M also needs confidentiality protection, then it is necessary to use encryption; however, the use of encryption does not rule out the necessity of data-integrity protection.

Obviously, a timestamp mechanism can also be obtained by applying asymmetric cryptographic techniques:

1. Alice \rightarrow Bob: $\text{sig}_A(M, T_A)$
2. Bob verifies the signature and

$$\begin{cases} \text{accepts} & \text{if signature verification passes \& } T_A \text{ is deemed valid} \\ \text{rejects} & \text{otherwise} \end{cases} \quad (10.4.6)$$

A timestamp mechanism avoids the need for interaction, and is therefore suitable for applications which involves no interaction, e.g., in an electronic mail application. However, the disadvantage of a timestamp mechanism is that synchronized

time clocks are required and must be maintained securely. This can be difficult. Difficulties, precautions and objections to timestamps have been well-documented in the literature [BM90b], [BGH⁺92], [GS91], [DvOW92].

In the basic protocol constructions introduced so far, a nonce or a timestamp are special message components. They play the role of identifying the freshness of other messages which are cryptographically integrated with them. We shall use **freshness identifier** to refer to a nonce or a timestamp.

10.4.1.4 Standardization of timestamp mechanisms

The ISO/IEC have also standardized timestamp mechanisms for authentication protocols.

The ISO/IEC standardization for mechanism (10.4.4) is called “ISO Symmetric Key One-Pass Unilateral Authentication Protocol” [ISO98a] and is as follows:

1. $A \rightarrow B : \text{Token}AB$

where $\text{Token}AB = \text{Text}2 \parallel \mathcal{E}_{K_{AB}} \left(\begin{array}{c} T_A \\ N_A \end{array} \parallel B \parallel \text{Text}1 \right)$.

Again, because this simple mechanism uses an encryption-decryption approach, we should remind Remark 10.2 for the importance for the encryption algorithm to serve data-integrity protection.

Here $\begin{array}{c} T_A \\ N_A \end{array}$ denotes the choice between the use of T_A , which is a timestamp, and N_A , which is a **sequence number**. In the case of using a sequence number, Alice and Bob maintain a synchronized sequence-number keeper (e.g., a counter) so that the sequence number N_A will increase in a manner known to Bob. After a successful receipt and validation of a sequence number, each of the two principals should update its sequence-number keeper to the new state.

There are two disadvantages in a sequence-number mechanism. First, a set of state information must be maintained for each potential communication partner; this can be difficult for applications in an open environment where each principal may communicate with many other principals. Therefore a sequence-number mechanism does not scale well. Secondly, management of a sequence-number keeper can be very troublesome in the presence of communication errors, either genuine ones or deliberate ones (such as a result of a **denial-of-service attack**). Recall our convention made in §10.3 that an authentication protocol should be stateless; a stateful protocol cannot function properly in a hostile environment. We therefore do not recommend a sequence-number mechanism even though such mechanisms have been documented in ISO/IEC standards.

The ISO/IEC standardization for mechanism (10.4.5) is called “ISO One-Pass Unilateral Authentication with Cryptographic Check Functions” [ISO99], and is as

follows:

1. $A \rightarrow B : \text{Token}AB$

where^e $\text{Token}AB = \frac{T_A}{N_A} \parallel B \parallel \text{Text1} \parallel f_{K_{AB}}(\frac{T_A}{N_A} \parallel B \parallel \text{Text1})$; f is a keyed CCF, e.g., a keyed hash function.

The reader may have already predicted the following named protocol as the public-key counterpart for encryption and cryptographic-check-function versions: “ISO Public Key One-Pass Unilateral Authentication Protocol” [ISO98b]:

1. $A \rightarrow B : \text{Cert}A \parallel \text{Token}AB$

where $\text{Token}AB = \frac{T_A}{N_A} \parallel B \parallel \text{Text2} \parallel \text{sig}_A(\frac{T_A}{N_A} \parallel B \parallel \text{Text1})$.

10.4.1.5 Non-standard mechanisms

We have introduced so far several basic constructions for building authentication protocols. It is not difficult at all to imagine numerous other variations which can achieve the same purpose as that have been achieved by the introduced basic constructions. For example, a variation for mechanism (10.4.1) using symmetric cryptographic techniques can be

1. Bob \rightarrow Alice: $Bob, \mathcal{E}_{K_{AB}}(M, N_B)$
 2. Alice \rightarrow Bob: N_B
 3. Bob $\begin{cases} \text{accepts} & \text{if the returned nonce is correct} \\ \text{rejects} & \text{otherwise} \end{cases}$
- (10.4.7)

For another example, a variation for mechanism (10.4.3) using asymmetric cryptographic techniques can be:

1. Bob \rightarrow Alice: $\mathcal{E}_{K_A}(M, Bob, N_B)$
 2. Alice \rightarrow Bob: N_B
 3. Bob $\begin{cases} \text{accepts} & \text{if the returned nonce is correct} \\ \text{rejects} & \text{otherwise} \end{cases}$
- (10.4.8)

Here \mathcal{E}_{K_A} denotes a public-key encryption algorithm under Alice’s public key. In these two variations, Bob validates Alice’s lively correspondence by encrypting a freshness identifier and testing if she can perform timely decryption. We shall use *encryption-then-decryption* (of freshness identifier) to refer to these mechanisms.

^eAs in Footnote d, [ISO99] mistakenly specifies Text2 in the cleartext part of Text1, and so B may not be able to check the CCF.

While performing encryption-then-decryption of freshness identifier does provide a means for validating the lively correspondence of a an intended communication partner, such a mechanism is not desirable for constructing authentication protocols. In such a mechanism Alice can be used as a decryption oracle and discloses inadvertently confidential information. For example, Malice may record a ciphertext chunk from a confidential conversation between Alice and Bob, and inserts it in a protocols which uses encryption-then-decryption mechanism; then Alice may be tricked to disclose the confidential conversation. Recall our convention for honest principals (in §10.3): Alice may misinterpret a message into a nonce therefore return the “nonce” by following the “protocol instruction”.

The undesirability of encryption-then-decryption mechanisms has also been manifested by the fact that the ISO/IEC standardization process has not considered to standardize such a mechanism. That is part of the reason why we name mechanisms in (10.4.7) and (10.4.8) non-standard ones.

However, many authentication protocols have been designed to use a encryption-then-decryption mechanism. We will analyze several such protocols in §15.2; there we shall identify that the use of the non-standard mechanisms is the main cause of the security flaws in those protocols.

10.4.2 Mutual Authentication

The basic mechanisms for message freshness or principal-liveness introduced so far achieve so-called “unilateral authentication” which means that only one of the two protocol participants is authenticated. In **mutual authentication**, both communicating entities are authenticated to each other.

ISO and IEC have standardized a number of mechanisms for mutual authentication. A signature based mechanism named “ISO Public Key Three-Pass Mutual Authentication Protocol” [ISO98b] is specified in Protocol 10.1. We choose to specify this mechanism in order to expose a common misunderstanding on mutual authentication.

One might want to consider that that mutual authentication is simply twice unilateral authentication; that is, mutual authentication could be achieved by applying one of the basic unilateral authentication protocols in §10.4.1 twice in the opposite directions. However, this is not generally true!

A subtle relationship between mutual authentication and unilateral authentication was not clearly understood in an early stage of the ISO/IEC standardization process for Protocol 10.1. In several early standardization drafts for Protocol 10.1 [ISO91b], [Gol01], *TokenBA* was slightly different from that in the current version:

$$\text{TokenBA} = R'_B \parallel R_A \parallel A \parallel \text{sig}_B(R'_B \parallel R_A \parallel A).$$

The early draft intentionally disallowed *B* to re-use his challenge nonce R_B in order

Protocol 10.1: ISO Public Key Three-Pass Mutual Authentication Protocol

PREMISE: A has public key certificate Cert_A ;
 B has public key certificate Cert_B ;

GOAL: They achieve mutual authentication.

1. $B \rightarrow A : R_B$
2. $A \rightarrow B : \text{Cert}_A, \text{Token}AB$
3. $B \rightarrow A : \text{Cert}_B, \text{Token}BA$

where

$\text{Token}AB = R_A \parallel R_B \parallel B \parallel \text{sig}_A(R_A \parallel R_B \parallel B)$;

$\text{Token}BA = R_B \parallel R_A \parallel A \parallel \text{sig}_B(R_B \parallel R_A \parallel A)$.

(* Optional text fields are omitted. *)

Figure 10.1.

to avoid him signing a string which is partly defined, and fully known in advance, by A. Apart from this reasonable consideration, $\text{Token}BA$ in the early drafts was a syntactic and symmetric mirror image of $\text{Token}AB$. This version survived through a few revisions of ISO/IEC 9798-3, until an attack was discovered by the Canadian member body of ISO [ISO91b]. The attack is hence widely known as “Canadian Attack”. The attack is due to Wiener (see §12.9 of [MvOV97]). In addition to the ISO documentation, Diffie, van Oorschot and Wiener discuss the attack in [DvOW92]. We shall therefore also call the attack the Wiener Attack.

10.4.2.1 The Wiener attack (the Canadian attack)

The Wiener Attack on an early draft for “ISO Public Key Three-Pass Mutual Authentication Protocol” is given in Example 10.2 (recall our notation agreed in Chapter 2 for describing Malice sending and intercepting messages in a masquerading manner).

After the discovery of the Wiener Attack, the subsequent ISO/IEC 9798 series for standardization of authentication protocols start to take a cautious approach to mutual authentication. If $\text{Token}AB$ appears in a unilateral authentication protocol, then in a mutual authentication protocol which is augmented from the unilateral

Example 10.2: The Wiener Attack on ISO Public Key Three-Pass Mutual Authentication Protocol

PREMISE: In addition that of Protocol 10.1,
Malice has public key certificate Cert_M ;

1. Malice("B") $\rightarrow A : R_B$
2. $A \rightarrow \text{Malice}("B") : \text{Cert}_A, R_A \parallel R_B \parallel B \parallel \text{sig}_A(R_A \parallel R_B \parallel B)$
- 1'. Malice("A") $\rightarrow B : R_A$
- 2'. $B \rightarrow \text{Malice}("A") : \text{Cert}_B, R'_B \parallel R_A \parallel A \parallel \text{sig}_B(R'_B \parallel R_A \parallel A)$
3. Malice("B") $\rightarrow A : \text{Cert}_B, R'_B \parallel R_A \parallel A \parallel \text{sig}_B(R'_B \parallel R_A \parallel A)$

CONSEQUENCE:

A thinks that it is B who has initiated the run and accepts B 's identity; but B did not initiate the run, and is still awaiting for terminating a run started by Malice("A").

Figure 10.2.

version, the matching counterpart Token_{BA} for mutual authentication will have a context-sensitive link to Token_{AB} ; this link is usually made via re-using a freshness identifier used in the same (i.e., current) run.

In the current version of "ISO Public Key Three-Pass Mutual Authentication Protocol" (i.e., Protocol 10.1 which has been fixed from the early version vulnerable to the Wiener Attack), A is explicitly instructed to maintain the state regarding B 's nonce R_B until the current run terminates.

Gollmann investigated in-depth the subtle relation between unilateral authentication and mutual authentication [Gol01].

10.4.3 Authentication Involving Trusted Third Party

In the basic constructions for authentication protocols introduced in this chapter so far, we have assumed that the two protocol participants either already share a secure channel in the cases of the constructions using symmetric cryptographic techniques, or one knows the public key of the other in the cases of the constructions

using asymmetric cryptographic techniques. So we may say that these protocol constructions are to be used by principals who already know each other. Then why they still want to run an authentication protocol? One simple answer is that they want to refresh the secure channel between them by re-confirming a lively correspondence between them.

Another answer, a better one, is that, these basic protocol constructions actually form building blocks for authentication protocols which are for a more general and standard mode of communications in an open system environment.

The standard mode of communications in an open system is that principals “interact then forget”. An open system is too large for a principal to maintain the state information about its communications with other principals in the system. If two principals, who may unknown to each other, want to conduct secure communications, they will first establish a secure channel. In modern cryptography, a secure communication channel is under-pinned by a cryptographic key. Therefore, the two principals who wish to establish a secure channel between them should run an authentication protocol which has a sub-task of establishing an authenticated key. Such a protocol is called an authenticated key establishment protocol. Upon completion of a session of secure communication which is under-pinned by the key established, the two principals will promptly throw the channel away. Here, “throw channel away” means that a principal forgets the key underpinning that channel and will never re-use it anymore. That is why a secure channel established as an output of a run of an authenticated key establishment protocol is often called a session channel and the output key underpinning the channel is called a **session key**.

The standard architecture for principals to run authentication and key establishment protocols in an open system is to use a centralized authentication service from a **trusted third party** or a TTP. Such a TTP service may be an on-line one, or an off-line one. In the next chapter we shall introduce the authentication frameworks for authentication services provided by an off-line TTP.

In authentication services provided by an on-line TTP, the TTP has a long-term relationship with a large number of subjects in the system or in a subsystem. Authentication and/or authenticated key establishment protocols under the on-line TTP architecture are so designed that they are built upon the basic protocol constructions in §10.4.1 and §10.4.2 where one of the two “already known to each other” principals is the TTP, and the other is a subject. Cryptographic operation performed by the TTP can imply or introduce a proper cryptographic operation performed by a subject. With the help from the TTP, a secure channel between any two subjects can be established even if the two principals may not know each other at all. In Chapter 2 we have already seen a number of such protocols, where we name the TTP Trent.

The ISO/IEC standards for authentication protocols (the 9798 series) have two standard constructions involving an on-line trusted third party [ISO98a]. One of

them is named “ISO Four-Pass Authentication Protocol” and the other, “ISO Five-Pass Authentication Protocol”. These two protocols achieve mutual entity authentication and authenticated session key establishment. We shall, however, not specify these two protocols here for the following two reasons.

First, these protocols are built upon applying the basic protocol constructions we have introduced in §10.4.1 and §10.4.2, and therefore, in terms of providing design principles, they will not offer us anything new or positive in terms of conducting our further study of the topic. On the contrary, they contain a prominent feature of standardization which we do not wish to introduce in a textbook: many optional fields which obscure the simple ideas behind the protocols.

Secondly, they already have a “normal size” of authentication protocols, and should no longer be considered as building blocks for constructing authentication protocols for higher-level applications. Moreover, they actually contain some undesirable features such as sequence number maintained by the protocol participants (including TTP, i.e., stateful TTP!). Therefore, these two protocols must not be considered as model protocol constructions for any future protocol designers! On the contrary again, great care should be taken if any of these two protocols is to be applied in real applications.

We shall look at an entity authentication protocol involving TTP. However, this protocol is an insecure one: it is vulnerable to several kinds of attacks which we should expose in a later section.

10.4.3.1 The Woo-Lam protocol

The protocol is due to Woo and Lam [WL92] and hence we name it the Woo-Lam Protocol. The protocol is specified in Protocol 10.2.

We should notice that by choosing to introduce this protocol, we do *not* recommend it as a model protocol. On the contrary, not only is this protocol fatally flawed in several ways, has several different repaired versions which are all still flawed, but also it contains an undesirable design features we should expose, criticize and identify to be one fundamental reason for the discovered flaws in it. So we think that the Woo-Lam Protocol serves a useful role in our study of the difficult matter of designing correct authentication protocols.

The goal of this protocol is for Alice to authenticate herself to Bob even though the two principals do not know each other initially.

Initially, since Alice and Bob do not know each other, Alice’s cryptographic capability can only be shown to Trent: she encrypts Bob’s nonce N_B using her long term key shared with Trent (step 3). Trent, as TTP, will honestly follow the protocol and decrypt the ciphertext formed by Alice (after receiving the message in step 4). Finally, when Bob sees his fresh nonce retrieved from the cipher chunk from Trent, he can conclude: Trent’s honest cryptographic operation is only possible

Protocol 10.2: The Woo-Lam Protocol

PREMISE: Alice and Trent share a symmetric key K_{AT} ,
 Bob and Trent share a symmetric key K_{BT} ;

GOAL: Alice authenticates herself to Bob
 even though Bob does not know her.

1. Alice \rightarrow Bob: $Alice$;
2. Bob \rightarrow Alice: N_B ;
3. Alice \rightarrow Bob: $\{N_B\}_{K_{AT}}$;
4. Bob \rightarrow Trent: $\{Alice, \{N_B\}_{K_{AT}}\}_{K_{BT}}$;
5. Trent \rightarrow Bob: $\{N_B\}_{K_{BT}}$;
6. Bob decrypts the cipher chunk using K_{BT} , and accepts if the decryption returns his nonce correctly; he rejects otherwise.

Figure 10.3.

after Alice's cryptographic operation, and both of these operations are on his nonce which he has deemed to be fresh; thus, Alice's identity and her liveness have been demonstrated and confirmed.

On the one hand, the Woo-Lam protocol can be viewed as being built upon applying a standard protocol construction which we have introduced and recommended in §10.4.1.1. For example, message lines 2 and 3 follow mechanism (10.4.1); the same mechanism is also applied in message lines 3 and 4.

We shall defer the revelation of several security flaws in the Woo-Lam Protocol to §10.7. In addition, this protocol has a deeper undesirable design feature which we believe to be responsible for its security flaws. However, we shall further defer our analysis and critics on that undesirable feature to §15.2.1 where we take formal approaches to developing correct authentication protocols.

10.5 Password-based Authentication Techniques

Due to being easily memorable by human brain, **password based authentication** is widely applied in the “user-host” mode of remotely accessed computer systems. In this type of authentication, a user and a host share a password which is essentially a long-term but rather small size symmetric key.

So a user U wishing to use services of a host H must first be initialized with H a password. H keeps an archive of all users’ passwords. Each entry of the archive is a pair (ID_U, P_U) where ID_U is the identity of U , and P_U is the password of U . A straightforward password-based protocol for U to access H can be as follows:

1. $U \rightarrow H : ID_U$;
 2. $H \rightarrow U : \text{“Password”}$;
 3. $U \rightarrow H : P_U$;
 4. H finds entry (ID_U, P_U) from its archive;
- Access is granted if P_U received matches the archived.

We should note that this protocol does not actually achieve any sense of entity authentication, not even a unilateral authentication from U to H . This is because in no part of the protocol there involves a freshness identifier for identifying lively correspondence of U . However, the term “password authentication” began its use in early 1970s when a user accessed a mainframe host from a dumb terminal and the communication link between the host and the terminal was a dedicated line and was not attackable. Under such a setting of devices and communications, the above protocol does provide unilateral entity authentication from U to H .

However, under a remote and open network communication setting, because no principal in the password protocol performs any cryptographic operation, this protocol has two serious problems.

The first problem is the vulnerability of the password file kept in H . The stored password file in H may be read by Malice (now Malice is an insider who can even be a system administrator). With the password file, Malice obtains all rights of all users; he can gain access to H by impersonating a user and causes undetectable damage to the impersonated user or even to the whole system. Obviously, to cause damage under a user’s name has a much lowered risk for Malice to be detected.

10.5.1 Needham’s Password Protocol

Needham initiated an astonishingly simple and effective method to overcome the secure storage of passwords in a host (see “Acknowledgements” in [EJKW74], see also [GM84b]). The host H should use a one-way function to encode the passwords,

Protocol 10.3: Needham's Password Authentication Protocol

PREMISE: User U and Host H have setup U 's password entry $(ID_U, f(P_U))$ where $f()$ is a one-way function;
 U memorizes password P_U ;

GOAL: U logs in H using her/his password.

1. $U \rightarrow H : ID_U$;
2. $H \rightarrow U : \text{"Password"}$;
3. $U \rightarrow H : P_U$;
4. H applies $f()$ on P_U , finds entry $(ID_U, f(P_U))$ from its archive;
 Access is granted if the computed $f(P_U)$ matches the archived.

Figure 10.4.

that is, the entry (ID_U, P_U) should be replaced with $(ID_U, f(P_U))$ where f is a one-way function which is extremely difficult to invert. The simple "password protocol" given above should also be modified to one in Protocol 10.3.

Now in Needham's Password Protocol, stealing $f(P_U)$ from H will no longer provide Malice with an easy way to attack the system. First, $f(P_U)$ cannot be used in Protocol 10.3 because using it will cause H to compute $f(f(P_U))$ and fail the test. Secondly, because it is computationally infeasible to invert the one-way $f()$, if the users choose their passwords properly so that a password is not easy to be guessed, then it will be very difficult for Malice to find P_U from $f(P_U)$. (We shall discuss password guessing problem in §10.5.3.)

In order for $f(P_U)$ to have a more random distribution over the range space of $f()$, a random number can be added to the input of $f()$. Such an added random number, which is usually called a **salt**, is generated in the user initialization time, and is stored together with the user's password data entry. Thus, an entry stored in the password file in H should be $(ID_U, f(P_U, salt), salt)$.

Although the secrecy of the password file is no longer a concern, the data-integrity of the file must be maintained.

The second problem with the simple password-based remote access protocol is that a password travels from U to H in cleartext and therefore it can be eavesdropped by Malice. This attack is called **on-line password eavesdropping**.

10.5.2 A One-time Password Scheme (and a Flawed Modification)

Lamport proposed a simple idea to thwart on-line password eavesdropping [Lam81]. The technique can be considered as a one-time password scheme. Here “one-time” means that the passwords transmitted from a given U to H do not repeat, however they are computationally related one another.

In the user initialization time, a password entry of U is set to $(ID_U, f^n(P_U))$ where

$$f^n(P_U) \stackrel{\text{def}}{=} \underbrace{f(\cdots(f(P_U))\cdots)}_n$$

for a large integer n . The user U still memorizes P_U as in the case of the Password Authentication Protocol.

When U and H engages in the first run of password authentication, upon prompted by “Password” (message line 2 in the Password Authentication Protocol), a computing device of U , such as a client platform or a calculator, will ask U to key in P_U , and will then compute $f^{n-1}(P_U)$ by repeatedly applying $f()$ $n - 1$ times. This can be efficiently done even for a large n (e.g., $n = 1000$). The result will be sent to H as in message line 3 in the Password Authentication Protocol.

Upon receipt of $f^{n-1}(P_U)$, H will apply $f()$ once on the received password to obtain $f^n(P_U)$ and then performs the correctness test as in step 4 in the Password Authentication Protocol. If the test passes, it assumes that the received value is $f^{n-1}(P_U)$ and must have been computed from P_U which was setup in the user initialization time, and hence it must be U in the other end of the communication. So U is allowed to enter the system. In addition, H will update U ’s password entry: replace $f^n(P_U)$ with $f^{n-1}(P_U)$.

In the next run of the protocol, U (whose computing device) and H will be in the state of using $f^{n-2}(P_U)$ with respect to $f^{n-1}(P_U)$, as in the previous case of using $f^{n-1}(P_U)$ with respect to $f^n(P_U)$. The protocol is hence a stateful one on a counter number descending from n to 1. Upon the counter number reaches 1, U and H have to reset a new password.

It is easy to see that in this method, a password eavesdropped from a protocol run is no good for further use, and hence the password eavesdropping problem is successfully prevented.

However, the method requires U and H to be synchronous for the password state: when H is in state of using $f^i(P_U)$ then U must be in state of sending $f^{i-1}(P_U)$. This synchronization can be lost if the communication link is “unreliable” or when system “crashes”. Notice that “unreliable” or “crash” can be the working of Malice!

Lamport considered a simple method to re-establish synchronization if it is lost [Lam81]. The method is essentially to have the system to “jump forward”: if H ’s state is $f^j(P_U)$ while U ’s state is $f^k(P_U)$ with $j \neq k + 1$, then synchronization is

Protocol 10.4: The S/KEY Protocol

PREMISE: User U and Host H have setup U 's initial password entry $(\text{ID}_U, f^n(P_U), n)$ where $f()$ is a cryptographic hash function;
 U memorizes password P_U ;
 The current password entry of U in H is $(\text{ID}_U, f^c(P_U), c)$ for $1 \leq c \leq n$.

GOAL: U authenticates to H without transmitting P_U in cleartext.

1. $U \rightarrow H : \text{ID}_U$;
2. $H \rightarrow U : c$, “Password”;
3. $U \rightarrow H : Q = f^{c-1}(P_U)$;
4. H finds entry $(\text{ID}_U, f^c(P_U), c)$ from its archive;
 Access is granted if $f(Q) = f^c(P_U)$, and U 's password entry is updated to $(\text{ID}_U, Q, c - 1)$.

Figure 10.5.

lost. The system should “jump forward” to a state $f^i(P_U)$ for H and $f^{i-1}(P_U)$ for U where $i \leq \min(j, k)$. It is clear that this way of re-synchronization requires mutually authenticated communications between H and U , however, Lamport did not provide details for this need in his short technical note.

Lamport's password-based remote access scheme has been modified and implemented into a “one-time password” system named **S/KEY**^f [Hal94]. The S/KEY modification attempts to overcome the “unreliable communication” problem by H maintaining a counter number c for U . In the user initialization time H stores U 's password entry $(\text{ID}_U, f^c(P_U), c)$ where c is initialized to n . Protocol 10.4 specifies the scheme.

Clearly, in Protocol 10.4, U and H will no longer lose synchronization and thereby unreliable communication link will no longer be a problem.

Unfortunately, the S/KEY modification to Lamport's original technique is a dangerous one. We notice that a password-based remote access protocol achieves, at best, an identification of U to H . Thus, the counter number sent from H can actually be one from Malice, or one modified by him. The reader may consider how

^fS/KEY is a trademark of Bellcore.

Malice should, e.g., modify the counter number and how to follow up an attack. The reader is encouraged to attack the S/KEY Protocol before reading §10.7.2.

One may want to argue: “the S/KEY Protocol cannot be more dangerous than Needham’s Password Authentication Protocol (Protocol 10.3) which transmits password in cleartext!” We should however notice that Needham’s Password Authentication Protocol has never claimed a security for preventing an on-line password eavesdropping attack. The S/KEY Protocol is designed to have this claim, which unfortunately does not stand.

10.5.3 Add Your Own Salt: Encrypted Key Exchange (EKE)

Most password-based systems advise users to choose their passwords such that a password should have eight keyboard (ASCII) characters. A password of this length is memorable by most users without writing down. Since an ASCII character is represented by a byte (8 bits), an eight-character password can be translated to a 64-bit string. A space of 64-bit strings has 2^{64} elements and is therefore comfortably large. So it seems that an 8-key-board-character password should resist guessing and even automated searching attacks mounted by non-dedicated attacker.

However, the “64-bit” password is not a true story. Although the information rate of the full set of ASCII characters is not too substantially below 8 bits/character (review §3.7 for information rate of a language), people usually do not choose their passwords using random characters in the ASCII table. In contrast, they choose bad passwords in order to be easily memorable. A typically bad password is a dictionary word, or a person’s name, all in lower case, maybe trailed by a digit or two. Shannon estimated that the rate of English is in the range of 1.0 to 1.5 bits/character ([Sha51], this estimate is based on English words of all lower case letters, see §3.7). Thus, in fact, the space of 8-character passwords should be much much smaller than 2^{64} , and be significantly much smaller if many passwords in the space are bad ones (lower case alphabetic, person’s names, etc). The smaller password space permit **off-line dictionary attacks**. In such an attack, Malice uses $f(P_U)$ to search through a dictionary of bad passwords for the matching P_U . Because the attack is mounted off-line, it can be automated and can be fast. We should notice that Lamport’s one-time password scheme does not provide a protection against off-line dictionary attacks either: Malice can eavesdrop the current state value i and $f^i(P_U)$ and hence can conduct the dictionary search.

Bellovin and Merritt proposed an attractive protocol for achieving secure password based authentication. Their protocol is named **encrypted key exchange (EKE)** [BM92]. The EKE Protocol protects password against not only on-line eavesdropping, but also off-line dictionary attacks. The technique used in the EKE scheme is essentially **probabilistic encryption**. In Chapter 13 we shall study general techniques for probabilistic encryption. Here, the reader may consider the trick to be “add your own salt” to a password.

Unlike in the password based protocols (Protocol 10.3 or Protocol 10.4) where H only possesses a one-way image of U 's password, in the EKE Protocol U and H share a password P_U . The shared password will be used as a symmetric cryptographic key, though, as we have mentioned, this symmetric key is chosen from a rather small space.

The EKE Protocol is specified in Protocol 10.5.

The ingeniousness of the EKE Protocol is in the first two steps. In step 1, the cipher chunk $P_U(\mathcal{E}_U)$ is a result of encrypting a piece of one-time and random information \mathcal{E}_U under the shared password P_U . In step 2, the content which is doubly encrypted in the cipher chunk $P_U(\mathcal{E}_U(K))$ is another one-time and random number: a session key K . Since P_U is human-brain memorable and hence is small, the random strings \mathcal{E}_U and K must have larger sizes than that of P_U . So the two cipher chunks in message lines 1 and 2 can hide P_U in such a way that P_U is statistically independent from these two cipher chunks.

We must emphasize that it is the one-time randomness of \mathcal{E}_U that plays the “adding you own salt” trick. Should the “public key” be not one-time, the unique functionality of the EKE Protocol would have failed completely: it would even be possible to facilitate Malice to search the password P_U using the weakness of a textbook public-key encryption algorithm (see, e.g., a “meet-in-the-middle” attack in §8.3.2).

If the nonces N_U , N_H encrypted in message lines 3, 4 and 5 are generated at random and have adequately large sizes (i.e, larger than that of the session key K), then they further hide the session key K in the same fashion as the password P_U is hidden in the first two messages. Thus, P_U remains to be statistically independent from any messages passed in the EKE Protocol.

The statistical independence of the password P_U from the messages passed in a protocol run means that the password is hidden from an eavesdropper in an information-theoretically secure sense (see §7.4). So a passive eavesdropper can no longer mount an off-line dictionary attack on P_U using the protocol messages. The only possible other ways to attack the protocol are either to try to guess P_U directly, or to mount an active attack by modifying protocol messages. The guessing attack is an uninteresting one, it can never be prevented however fortunately it can never be effective. An active attack, on the other hand, will be detected with a high probability by a honest protocol participant, and will cause a run being promptly abandoned.

The encryption of a random public key in step 1 by U , and that of a random session key in step 2 by H , are what we have referred to as “add your own salt” to the password P_U . The ever changing “salt” keeps an attacker out of pace. Therefore, the first two message lines in the EKE Protocol provide an ingenious technical novelty. The message lines 3, 4 and 5 form a conventional challenge-response-based mutual authentication protocol. Indeed they can be replaced by a suitable mutual

Protocol 10.5: Encrypted Key Exchange (EKE)

PREMISE: User U and Host H share a password P_U ;
 The system has agreed on a symmetric encryption algorithm,
 $K()$ denotes symmetric encryption keyed by K ;
 U and H have also agreed on an asymmetric encryption scheme,
 \mathcal{E}_U denotes asymmetric encryption under U 's key.

GOAL: U and H achieve mutual entity authentication,
 they also agree on a shared secret key.

1. U generates a random “public” key \mathcal{E}_U , and sends to H :

$$U, P_U(\mathcal{E}_U);$$

(* the “public” key is in fact not public, it is the encryption key of an asymmetric encryption algorithm *)

2. H decrypts the cipher chunk using P_U and retrieves \mathcal{E}_U ;
 H generates random symmetric key K , and sends to U :

$$P_U(\mathcal{E}_U(K));$$

3. U decrypts the doubly encrypted cipher chunk and obtains K ;
 U generates a nonce N_U , and sends to H :

$$K(N_U);$$

4. H decrypts the cipher chunk using K , generates a nonce N_H , and sends to U :

$$K(N_U, N_H);$$

5. U decrypts the cipher chunk using K , and return to H :

$$K(N_H);$$

6. If the challenge-response in steps 3, 4, 5 is successful, login is granted and the parties proceed further secure communication using the shared key K .

Figure 10.6.

protocol construction based on a shared symmetric key.

The EKE Protocol is very suitable to be realized using the Diffie-Hellman key exchange mechanism. Let α generate a group of order larger than $2^{64} > 2^{|P_U|}$. Then in step 1, U 's computing device picks at random $x \in (0, 2^{64})$ and computes $\mathcal{E}_U = \alpha^x$, and in step 2, H picks at random $y \in (0, 2^{64})$ and computes $\mathcal{E}_U(K) = \alpha^y$. The agreed session key between U and H will be $K = \alpha^{xy}$. Now, each party has its own contribution to the agreed session key. Also, the protocol has the perfect forward secrecy property. In this realization, the group generator α can be agreed between U and H in public: U sends to H α and the group description in a pre-negotiation step.

Notice that we have only required that α generates a group of order larger than 2^{64} . This is a very small number as a lower bound for a group order in an asymmetric cryptographic system. So the protocol can be very efficient. The small group order renders easy ways to compute discrete logarithm, and hence to solve the computational Diffie-Hellman problem. However, without α^x , α^y , α^{xy} , the ease of solving Diffie-Hellman problem is of no help for finding the password: P_U remains to be statistically independent in a space of size 2^{64} . Likewise, for sufficiently large and random nonces encrypted in message lines 3, 4, and 5, the session key K shall remain independent in the group of an order larger than 2^{64} . Thus, the difficulties for off-line dictionary attacks or on-line key guess remain without being eased.

In essence, the random “salt” added to a password “amplifies” the size of the password space from that of a dictionary to that of the random asymmetric key. This is the trick behind the EKE Protocol.

10.6 Authenticated Key Exchange Based on Asymmetric Cryptography

We say that a protocol establishes a shared session key via a **key transport** mechanism if the protocol outputs a shared key which comes from one of the protocol participants. We say that a protocol establishes a shared session key via a **key exchange** (or **key agreement**) mechanism if a run of the protocol outputs a shared key which is a function of all protocol participants' random input. The advantage of key exchange over key transport is that each of the key-sharing parties can have its own control, hence a high confidence, on the quality of the key output.

The basic authentication techniques introduced so far, if involving key establishment, are all key transport mechanisms. Now let us introduce a key exchange mechanism.

Key exchange can be achieved by generating a key as the output of a pseudo random function or a one-way hash function where the key-sharing parties have their own input parts to the function. The most commonly used method is the great discovery of Diffie and Hellman: Diffie-Hellman Key Exchange which we have

considered as a one-way function (see Remark 8.1.3). We have specified Diffie-Hellman Key Exchange in Protocol 8.1. This mechanism achieves to agree a key between two remote principals without using encryption.

Protocol 8.1 is the basic version of Diffie-Hellman Key Exchange which achieves unauthenticated key agreement. We have seen a **man-in-the-middle attack** in Example 8.2 with which Malice shares one key with Alice and another with Bob and hence can relay the “confidential” communications in between Alice and Bob. A proper use of Diffie-Hellman Key Exchange must be a variation of Protocol 8.1. The simplest variation is a two-party protocol in which Alice knows for sure that g^b is Bob’s public key:

$$1. \text{ Alice} \rightarrow \text{Bob: } \textit{Alice}, g^a \quad (10.6.1)$$

where number a is picked at random by Alice from a suitably large integer interval.

After sending the message in (10.6.1), Alice knows that g^{ab} is a key exclusively shared with Bob since for anybody other than herself and Bob, to find g^{ab} is to solve the computational Diffie-Hellman problem (CDHP, see Definition 8.1) which is assumed to be computationally infeasible. Since Alice has picked her exponent at random which is new, the agreed key is fresh and this means that the key is authenticated to Alice. However, upon receipt of g^a , Bob cannot know with whom he shares the key g^{ab} or whether the key is fresh. Therefore, this simple variation achieves unilateral authenticated key agreement.

Applying various mechanisms introduced so far, it is not difficult to augment mechanism (10.6.1) to one which allows the agreed key to be mutually authenticated. For example, Alice may digitally sign g^a with her identity and a timestamp.

Let us introduce here a well-known authenticated key exchange protocol which is a variation of Diffie-Hellman Key Exchange.

10.6.1 The Station-to-station Protocol

The **Station-to-station (STS) Protocol** is proposed by Diffie et al [DvOW92].

In the STS Protocol, Alice and Bob have agreed on using a large finite abelian group which is generated by a common element α . System wise users can use a common generator α . The reader may review §?? for cautions to setting up the shared group to be used in the STS Protocol.

Alice and Bob also have their respective public key certificates:

$$\text{Cert}_A = \text{sig}_{CA}(\textit{Alice}, P_A, \text{desc } \langle \alpha \rangle)$$

$$\text{Cert}_B = \text{sig}_{CA}(\textit{Bob}, P_B, \text{desc } \langle \alpha \rangle)$$

where CA is a certification authority (see Chapter 12), P_A and P_B are the public keys of Alice and Bob, respectively, and $\text{desc } \langle \alpha \rangle$ is the description of the shared

Protocol 10.6: The Station-To-Station (STS) Protocol

PREMISE: Alice has her public-key certificates Cert_A ,
 Bob has his public-key certificates Cert_B ,
 the system-wise users share a large finite abelian group $\text{desc } \langle \alpha \rangle$,
 and they agree on a symmetric encryption algorithm $\mathcal{E}()$;

GOAL: Alice and Bob achieves mutual authentication
 and mutually authenticated key agreement.

1. Alice picks a random large integer x , and sends to Bob:

$$\alpha^x;$$

2. Bob picks a random large integer y , and sends to Alice:

$$\alpha^y, \text{Cert}_B, \mathcal{E}_K(\text{sig}_B(\alpha^y, \alpha^x));$$

3. Alice sends to Bob:

$$\text{Cert}_A, \mathcal{E}_K(\text{sig}_A(\alpha^x, \alpha^y)).$$

Where $K = \alpha^{xy} = \alpha^{yx}$.

WARNING:

This protocol is flawed in a minor way; to be analysed in §10.6.3.

Figure 10.7.

group generated by α . In addition, the two parties have also agreed on using a symmetric-key encryption algorithm, which we shall use the notation given in Definition 7.1. The encryption algorithm can also be agreed upon for system-wise users.

The STS Protocol is specified in Protocol 10.6.

It is intended that the STS Protocol should have the following four security properties:

Mutual entity authentication However this property actually does not hold according to the rigorous definition for authentication given by the authors

of the STS Protocol. In [DvOW92], Diffie et al made two mistakes in this respect. We shall discuss them in §10.6.2 and §10.6.3, respectively.

Mutually authenticated key agreement Key agreement is obvious from the Diffie-Hellman key exchange protocol; the freshness of the agreed key is guaranteed if each party has picked her/his random exponent properly; the exclusive sharing of the agreed key is implied by both parties digital signature on their key agreement material. However, pulling together all these features does not actually result in mutually authenticated key agreement: the property will only hold if the minor flaw in the protocol is fixed.

Mutual key confirmation Upon termination of a run, both parties have seen that the other party has used the agreed key to encrypt the key agreement material.

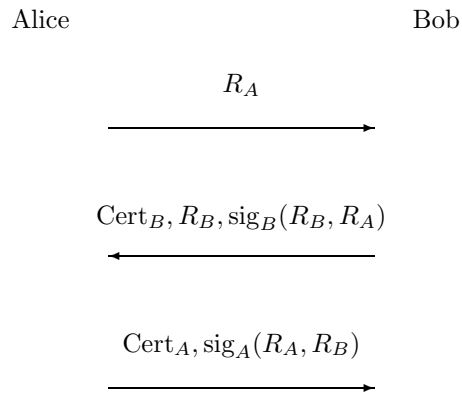
Perfect forward secrecy (PFS) This is an attractive property of a key establishment protocol which means that if a long-term private key used in a key establishment protocol is compromised at a point in time, the security of any session established earlier than that point will not be effected [Gün90], [DvOW92]. The PFS property holds for a key establishment protocol where a session key is properly agreed using the Diffie-Hellman key exchange mechanism. Here in the case of the STS protocol, the long-term keys are the private keys of Alice and Bob. Since each session key agreed in a run of the protocol is a one-way function of two ephemeral secrets which will be securely disposed upon termination of the run, compromise of either of the signing long-term keys cannot have any effect on the secrecy of the previously agreed session keys.

Anonymity If the public-key certificates are encrypted inside the respective cipher chunks, then the messages communicated in a run of the protocol will not reveal to any third party who are involved in the message exchanges. However we should notice that addressing information transmitted in a lower-layer communication protocol may disclose the identities of the protocol participants. Therefore, precisely speaking, “anonymity” should be rephrased to a kind of “deniability” which means that a network monitor cannot prove that a given protocol transcript takes place between two specific principals. Because the STS Protocol is one of the bases for the Internet Key Exchange (IKE) protocol suite for Internet Security [HC98], [Kau02b], [Orm96], this property is a feature in the IKE. We shall study IKE (and this feature) in the next chapter (§11.2).

The STS Protocol, although the version specified in Protocol 10.6 is flawed in a minor way, is an important and influential work in the area of authentication and authenticated key-exchange protocols. It is one of the bases for an authentication protocol suite for Internet security “Internet Key Exchange (IKE) Protocol” [HC98],

Protocol 10.7: Flawed “Authentication-only STS Protocol”

PREMISE: Alice has her public-key certificates Cert_A ,
 Bob has his public-key certificates Cert_B .

**Figure 10.8.**

[Orm96]. We shall see in §11.2 the influence of the STS Protocol on the IKE Protocol.

The paper [DvOW92] contains two flaws: a serious one in a simplified version of the STS Protocol for “authentication only” usage; a minor one in the STS Protocol proper. If a widely recognized protocol design principle is followed (that principle was documented and became popularly aware after the publication of [DvOW92]), then both flaws disappear. Let us now look at these flaws. Our study of these flaws shall lead to that widely recognized protocol design principle.

10.6.2 A Flaw in a Simplified STS Protocol

In order to argue the mutual authentication property, Diffie et al simplified the STS Protocol into one they name “Authentication-only STS Protocol” (§5.3 of [DvOW92]). They then claimed that the “simplified protocol is essentially the same as the three-way authentication protocol” proposed by ISO. The “ISO protocol” they referred to is in fact what we named (following ISO/IEC’s name) the “ISO Public Key Three-Pass Mutual Authentication Protocol” (Protocol 10.1 with the “Wiener Attack” being fixed, see §10.4.2).

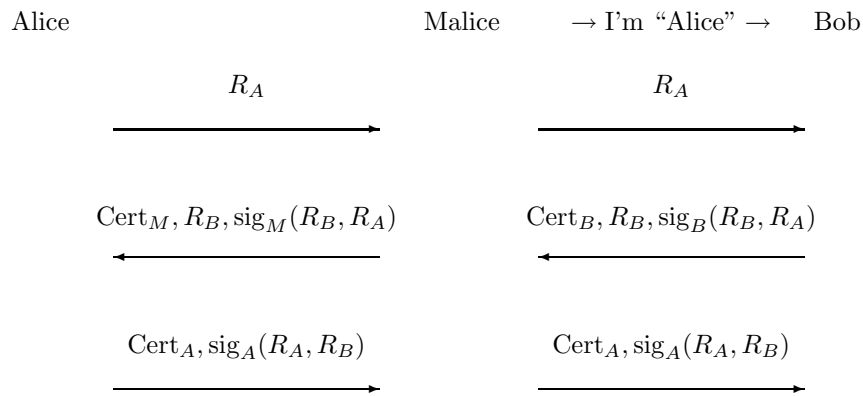
The simplified “Authentication-only STS Protocol” is specified in Protocol 10.7.

However, Protocol 10.7 is subtly different from the ISO Protocol. In this simplified STS Protocol, the signed messages do not contain the identities of the protocol participant, while in the ISO Protocol the signed messages contain the identities. The simplified STS Protocol suffers a “certificate-signature-replacement attack” which is demonstrated in Example 10.3.

Example 10.3: An Attack on the “Authentication-only STS Protocol”

PREMISE: In addition to that in Protocol 10.7,
 Malice has his public key certificate Cert_M .
 (* So Malice is also a normal user in the system *)

(* Malice faces Alice using his true identity, but he faces Bob by masquerading as Alice: *)



CONSEQUENCE:

Bob thinks to have been talking with Alice while she thinks to have been talking with Malice.

Figure 10.9.

In this attack, Malice, who is a legitimate user of the system and hence has a public-key certificate, waits for Alice to initiate a run. Upon occurrence of such an opportunity, he starts to talk to Bob by impersonating Alice and using her nonce. Upon receipt of Bob’s reply, Malice replaces Bob’s certificate and signature with his own copies, respectively. Doing so can successfully persuade Alice to sign Bob’s nonce, which in turn allows Malice to cheat Bob successfully. This is a perfect

attack because neither Alice nor Bob can discover anything wrong.

One may think that because in the communications between Malice and Bob, Malice is passive, i.e., is acting just like a wire, therefore this should not be regarded as a real attack. However, Malice is not passive in the whole attacking run orchestrated by him: he signs Bob's nonce and hence successfully persuades Alice to sign Bob's nonce so that he can fool Bob completely. Should Malice really behave like a wire, then Bob would have never receive Alice's signature on his nonce, and thereby would not have been cheated.

This “certificate-signature-replacement attack” does not apply to the STS Protocol because the encryption used in the full version of the protocol prevents Malice from replacing Bob's signature. The attack does not apply to the ISO Protocol (Protocol 10.1) either because there, Malice's identity will appear in the message signed by Alice, and hence that message cannot be passed to Bob to fool him.

Including the identity of the intended verifier inside a signature does indeed constitute a method to fix the flaw. Of course, we do not suggest that adding identity is the only way to fix this flaw. In some applications (e.g., “Internet Key Exchange (IKE) Protocol”, see §11.2.3), identities of the protocol participants are desirably omitted in order to obtain a privacy property (see §??). A novel way for fixing such flaws while keeping the desired privacy property can be devised by using a novel cryptographic primitive, which we shall introduce in a later chapter.

10.6.3 A Minor Flaw of the STS Protocol

Lowe discovered a minor attack on the STS Protocol [Low94]. Before presenting Lowe's attack, let us review a rigorous definition for authentication given by the authors of the STS Protocol.

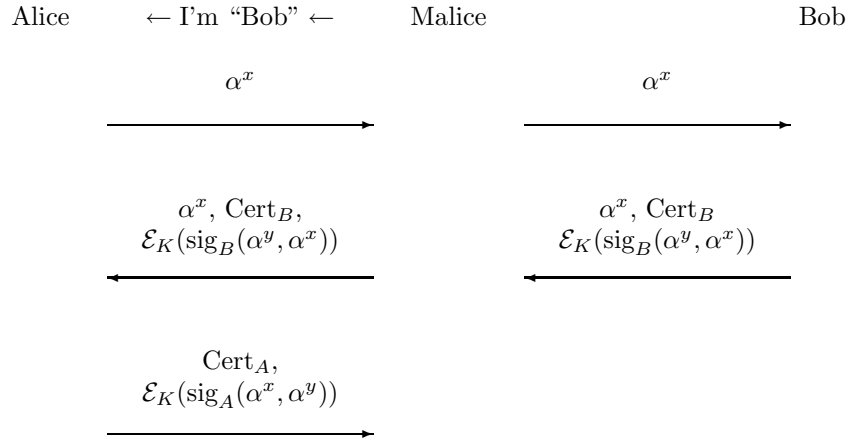
In [DvOW92], Diffie et al define a secure run of an authentication protocol using the notion of “matching records of runs”. Let each protocol participant record messages received during a run. A “matching records of a run” means that the messages originated by one participant must appear in the record of the other participant in the same order of sequence as the messages are sent, and vice versa. Then an insecure run of an authentication protocol (Definition 1 of [DvOW92]) is one:

if any party involved in the run, say Alice, executes the protocol faithfully, accepts the identity of another party, and the following condition holds: at the time that Alice accepts the other party's identity, the other party's record of the partial or full run does not match Alice's record.

Under this definition for an insecure run of an authentication protocol, What is demonstrated in Example 10.4 qualifies a legitimate attack on the STS Protocol, even though the damage it can cause is very limited.

Example 10.4: Lowe's Attack on the STS Protocol (a Minor Flaw)

(* Malice faces Bob using his true identity, but he faces Alice by masquerading as Bob: *)

**CONSEQUENCE:**

Alice is fooled perfectly and thinks to have been talking and sharing a session key with Bob, while Bob thinks to have been talking with Malice in an incomplete run. Alice will never be notified of any abnormality, and her subsequent requests or preparation for secure communications with Bob will be denied without any explanation.

Figure 10.10.

Lowe's attack is a minor one in the following two senses:

- i) in the run part between Alice and Malice, although Malice is successful in fooling Alice, he does not know the shared session key and hence cannot fool Alice any further after the run;
- ii) in the run part between Malice and Bob, Malice cannot complete the run, and so this part is not a successful attack.

We however think Lowe's attack qualifies a legitimate attack also for two reasons:

- I) Alice accepts the identity of Bob as a result of Malice simply copying bit-by-bit all Bob's message to Alice. However, in Bob's end, since he sees the

communication partner to be Malice while he signs Alice's random challenge, his recorded messages do not match those of Alice's. Therefore, the attack meets the "insecure run" criterion defined by the STS authors, i.e., mutual authentication actually fails. Of course, as the notion of entity authentication is quite hard to capture precisely, and as the area of study has been developing through mistakes, a verdict of an attack given on the basis of a quite old definition (i.e., that of and "insecure run" given by Diffie et al [DvOW92]) may not be sufficiently convincing. Today, one may well question whether that early definition is correct at all. However, a better question should be: "Will this 'attack' be a concern in practice today?" This is answered in (II).

- II) Malice successfully fools Alice into believing a normal run with Bob. Her subsequent requests or preparation for secure communications with Bob will be denied without any explanation since Bob thinks he has never been in communication with Alice. Also, nobody will notify Alice any abnormality. We should compare this consequence with one resulted from a much less interesting "attack" in which Malice is passive except for cutting the final message from Alice to Bob without letting Bob receive it. In this less interesting "attack", due to the expected matching records, Bob may notify Alice the missed final message. Regarding whether Lowe's attack is a concern in practice today, we may consider that if Alice is in a centralized server's situation, and is under a distributed attack (i.e., is under a mass attack launched by Malice's team distributed over the network), lack of notification from end-users (i.e., from many Bob) is indeed a concern: the server will reserve resource for many end-users and its capacity to serve the end-users can be drastically powered down. We should particularly notice that in Lowe's attack, Malice and his friends do not use any cryptographic credentials (certificates). So this attack costs them very little. This is again very different from a conventional denial-of-service attack in which Malice and his friends have to talk to Alice in their true names (i.e., with certificates).

For the reason explained in (II), we shall name Lowe's attack a **perfect denial of service attack** against Alice: the attackers succeed using other parties' cryptographic credentials.

This attack can be avoided if the protocol is modified into one which follows a correct and well recognized protocol design principle proposed by Abadi and Needham [AN95]:

If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

Indeed, the signed messages in the STS Protocol should include the identities of both protocol participants! This way, the message from and signed by Bob will contain "Malice" so that Malice cannot forward it to Alice in Bob's name, i.e.,

Malice can no longer fool Alice. Moreover, if the simplified “Authentication-only STS Protocol” is simplified from this identities-in-signature version, then it will not suffer the “certificate-signature-replacement attack” either since now the simplified version is essentially the ISO Protocol (Protocol 10.1).

As we have mentioned earlier, the STS Protocol is one of the bases for the “Internet Key Exchange (IKE) Protocol” [HC98], [Kau02b], [Orm96]. As a result, we shall see in §11.2 that the “perfect denial of service attack” will also apply to a couple of modes in IKE.

Finally, we should recap a point we have made in §10.6.2: adding identity of a signature verifier is not the only way to prevent this attack. For example, using a **designated verifier signature** can achieve a better fix while without adding the identity. Such a fix will be a topic in a later chapter.

10.7 Typical Attacks on Authentication Protocols

In §2.2 we have agreed that Malice (and perhaps by co-working with his friends distributed over an open communication network) is able to eavesdrop, intercept, alter and inject messages in the open communication network, and is good at doing these by impersonating other principals. Viewing from a high layer (the application layer) of communications, Malice’s capabilities for mounting these attacks seem to be like magics: how can Malice be so powerful?

However, viewing from a lower-layer (the network layer) communication protocol, it actually does not require very sophisticated techniques for Malice to mount these attacks. We shall see the technical knowhow for mounting such attacks on a lower-layer communication protocol in §11.2 where we shall also see how communications take place in the network layer. For the time being, let us just accept that Malice has magic-like capabilities. Then, a flawed authentication protocol may permit Malice to mount various types of attacks.

While it is impossible for us to know all the protocol attacking techniques Malice may use (since he will constantly devise new techniques), knowing several typical ones will provide us with insight for how to develop stronger protocols avoiding these attacks. In this section, let us look at several well-known protocol attacking techniques in Malice’s portfolio. We should notice that, although we classify these attacking techniques into separate types, Malice may actually apply them in a combined way: a bit of this and a bit of that, until he can end up with a workable attack.

Before we go ahead, we should emphasize the following important point.

Remark 10.3: *A successful attack on an authentication or authenticated key establishment protocol usually does not refer to breaking a cryptographic algorithm, e.g., via a complexity theoretic based cryptanalysis technique. Instead, it usually refers*

to Malice's unauthorized and undetected acquisition of a cryptographic credential or nullification of a cryptographic service without breaking a cryptographic algorithm. Of course, this is due to an error in protocol design, not one in the cryptographic algorithm. \square

10.7.1 Message Replay Attack

In a message replay attack, Malice had previously recorded an old message from a previous run of a protocol and now replays the recorded message in a new run of the protocol. Since the goal of an authentication protocol is to establish lively correspondence of communication parties and the goal is generally achieved via exchanging fresh messages between/among communication partners, replay of old message in an authentication protocol violates the goal of authentication.

In §2.5.4.2 we have seen an example of message replay attack on the Needham-Schroeder Symmetric-key Authentication protocol (Example 2.2). Notice that there (review the last paragraph of §2.5.4.2) we have only considered one danger of that message replay attack: the replayed message encrypts an old session key which is assumed to be vulnerable (Malice may have discovered its value, maybe because it has been discarded by a careless principal, or maybe due to other vulnerabilities of a session key that we have discussed in §2.4).

Another consequence, probably a more serious one, of that attack should be referred to as authentication failure, i.e., absence of a lively correspondence between the two communication partners. Indeed, for that attack to work (review Example 2.2) Malice does not have to wait for an opportunity that Alice starts a run of the protocol with Bob, and does not have to know the value of the old session key; he can just start his attack by jumping to message line 3 and replays the recorded messages (now, not only the message in line 3, but also that in line 5, are the recorded old messages):

3. Malice("Alice") \rightarrow Bob: $\{K', Alice\}_{K_{BT}}$;
4. Bob \rightarrow Malice("Alice"): $\{I'm Bob! N_B\}_{K'}$;
5. Malice("Alice") \rightarrow Bob: $\{I'm Alice! N_B - 1\}_{K'}$.

Now Bob thinks Alice is communicating with him, while in fact Alice is not even on-line at all.

Message replay is a classic attack on authentication and authenticated key establishment protocols. It seems that we have already established a good awareness of message-replay attacks. This can be evidently seen from our ubiquitous use of freshness identifiers (nonces, timestamps) in the basic and standard protocol constructions introduced in §10.4. However, a good awareness does not necessarily mean that we must also be good at preventing such attacks. One subtlety of

authentication protocols is that mistakes can be made and repeatedly made even the designers know the errors very well in a different context. Let us look at the following case which shows another form of message replay attack.

In [VAB91], Varadharajan et al present a number of “proxy protocols” by which a principal passes on its trust in another principal to others who trust the former. In one protocol, Bob, a client, shares the key K_{BT} with Trent, an authentication server. Bob has generated a timestamp T_B and wants a key K_{BS} to communicate with another server S . Then S constructs $\{T_B + 1\}_{K_{BS}}$, and sends:

5. $S \rightarrow \text{Bob: } S, B, \{T_B + 1\}_{K_{BS}}, \{K_{BS}\}_{K_{BT}}$

The authors reason:

Having obtained K_{BS} , Bob is able to verify using T_B that S has replied to a fresh message, so that the session key is indeed fresh.

However, although a freshness identifier is cryptographically integrated with K_{BS} , Bob can obtain no assurance that K_{BS} is fresh. All that he can deduce is that K_{BS} has been used recently, but it may be an old, or even compromised key.

So we remark:

Remark 10.4: *Sometimes, a cryptographic integration between a freshness identifier and a message may only indicate the fresh action of the integration, not the freshness of the message being integrated.* \square

10.7.2 Man-in-the-Middle Attack

Man-in-the-middle attack in the spirit of the well-known “chess grand-master problem”[§] is generally applicable in a communication protocol where mutual authentication is absent. In such an attack, Malice is able to pass a difficult question asked by one protocol participant to another participant for an answer, and then passes the answer (maybe after a simple processing) back to the asking party, and/or vice versa.

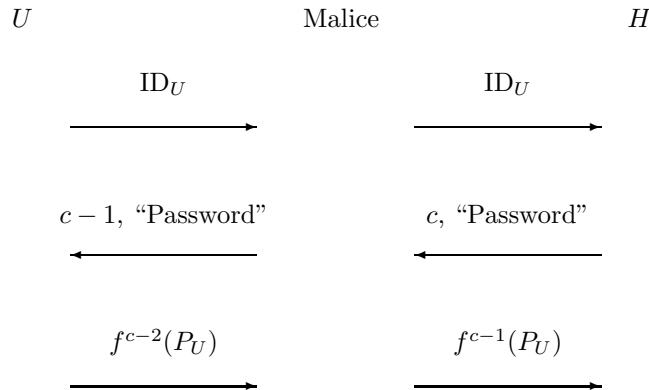
In §2.5.6.3 and §??, we have seen two cases of man-in-the-middle attack, one on the Needham-Schroeder Public-key Authentication Protocol, one on the unauthenticated Diffie-Hellman key exchange protocol.

The following man-in-the-middle attack on the S/KEY Protocol (Protocol 10.4) shows another good example on how Malice can gain a cryptographic credential without breaking the cryptographic algorithm used in the scheme.

[§]A novice who engages in two simultaneous chess games with two distinct grand-masters, playing Black in one game and White in the other, can take his opponents’ moves in each game and use them in the other to guarantee himself either two draws or a win and a loss, and thereby unfairly have his chess rating improved.

Example 10.5: An Attack on the S/KEY Protocol

(* Notations in the attack are the same as those in Protocol 10.4 *)

**CONSEQUENCE:**

Malice has in his possession $f^{c-2}(P_U)$ which he can use for logging-in in the name of U in the next session.

Figure 10.11.

The cryptographic hash function $f()$ used in the S/KEY scheme can be very strong so that it is computationally infeasible to invert; also the user U can have chosen the password P_U properly so that an off-line dictionary attack aiming at finding P_U from $f^c(P_U)$ does not apply (review §10.5.3 for off-line dictionary attack). However, the protocol fails miserably on an active attack which is demonstrated in Example 10.5.

The attack 10.5 works because in the S/KEY Protocol, messages from H are not authenticated to U .

The countermeasure for man-in-the-middle attack is to provide data-origin authentication service in both directions of message exchanges.

10.7.3 Parallel Session Attack

In a parallel session attack, two or more runs of a protocol are executed concurrently under Malice's orchestration. The concurrent runs make the answer to a difficult question in one run available to Malice so that he can use the answer in another

run.

An early attack on the Woo-Lam protocol (Protocol 10.2) discovered by Abadi and Needham [AN95] illustrates a parallel session attack. The attack is shown in Example 10.6.

This attack should work if Bob is willing to talk to Alice and Malice at roughly the same time. Then Malice can block messages flowing to Alice. In messages 1 and 1', Bob is asked to respond two runs, one with Malice and one with "Alice". In messages 2 and 2', Bob responds with two different nonce challenges, of course both will be received by Malice (one of them, N_B , will be received via interception). Malice throws away N'_B which is meant for him to use, but uses the intercepted N_B which is intended for Alice to use. So in messages 3 and 3' Bob receives $\{N_B\}_{K_{MT}}$. Notice that the two ciphertext chunks received in messages 3 and 3' may or may not be identical, this depends on the encryption algorithm details (see Chapters 13 and 14). At any rate, Bob should simply follow the protocol specification in messages 4 and 4': just encrypts them and sends to Trent. Notice that even if the two cipher chunks received by Bob in messages 3 and 3' are identical (when the encryption algorithm is deterministic, a less likely case in nowadays applications of encryption algorithms), Bob should not be able to notice it since the cipher chunk is not recognizable by Bob as it is not a message for Bob to process. Not to process a "foreign ciphertext" is consistent with our convention for the behavior of honest principals (see §10.3): Bob is not anticipating an attack and cannot recognize ciphertext chunks which are not meant for him to decrypt. We know such a behavior is stupid, but we have agreed as convention that Bob should be so "stupid". Now in messages 5 and 5', one of the cipher chunks Bob receiving from Trent will have the nonce N_B correctly returned, which will deceive Bob to accept "the run with Alice", but Alice is not on-line at all; the other cipher chunk will be decrypted to "garbage" because it is a result of Trent decrypting $\{N_B\}_{K_{MT}}$ using K_{AT} . The result, Bob rejects the run with Malice, but accepts the run "with Alice".

In a parallel session attack, the sequence of the two parallelled sessions are not important. For example, if Bob receives message 3' before receiving message 3, the attack works the same way. Bob can know who has sent which of these two messages from the addressing information in the network layer and we will see this clearly in §11.2.

Abadi and Needham suggested a fix for the Woo-Lam Protocol which we shall see in a moment. They also informed Woo and Lam the attack in Example 10.6 [AN95]. The latter authors proposed a series of fixes [WL94] which includes the fix suggested by Abadi and Needham (called Π^3 in [WL94]) and several more aggressive fixes; the most aggressive fix is called Π^f : adding the identities of the both subjects, i.e., *Alice* and *Bob*, inside all cipher chunks. They claimed that their fixes are secure. Unfortunately, none of them is (and hence the fix suggested by Abadi and Needham is also flawed). Each of their fixes can be attacked in a attack type which we shall

Example 10.6: A Parallel-Session Attack on the Woo-Lam Protocol

PREMISE: In addition to that of Protocol 10.2,
 Malice and Trent share long term key K_{MT} .
 (* So Malice is also a normal user in the system *)

1. Malice("Alice") \rightarrow Bob: *Alice*;
- 1'. Malice \rightarrow Bob: *Malice*;
2. Bob \rightarrow Malice("Alice"): N_B ;
- 2'. Bob \rightarrow Malice: N'_B ;
3. Malice("Alice") \rightarrow Bob: $\{N_B\}_{K_{MT}}$;
- 3'. Malice \rightarrow Bob: $\{N_B\}_{K_{MT}}$;
4. Bob \rightarrow Trent: $\{Alice, \{N_B\}_{K_{MT}}\}_{K_{BT}}$;
- 4'. Bob \rightarrow Trent: $\{Malice, \{N_B\}_{K_{MT}}\}_{K_{BT}}$;
5. Trent \rightarrow Bob: $\{\text{"garbage"}\}_{K_{BT}}$;
 (* "garbage" is the result of Trent decrypting $\{N_B\}_{K_{MT}}$ using K_{AT} *)
- 5'. Trent \rightarrow Bob: $\{N_B\}_{K_{BT}}$;
6. Bob rejects the run with Malice;
 (* since decryption returns "garbage" rather than nonce N'_B *)
- 6'. Bob accepts "the run with Alice", but it is a run with Malice;
 (* since decryption returns N_B correctly *)

CONSEQUENCE:

Bob believes that Alice is corresponding to him in a run while in fact Alice has not participated in the run at all.

Figure 10.12.

describe now.

10.7.4 Reflection Attack

In a reflection attack, when an honest principal sends to an intended communication partner a message for the latter to perform a cryptographic process, Malice intercepts the message and simply sends it back to the message originator. Notice that so reflected message is not a case of “message bounced back”: Malice has manipulated the identity and address information which is processed by a lower-layer communication protocol so that the originator will not notice that the reflected message is actually one “invented here”. We shall see the technical knowhow in §11.2.

In such an attack, Malice tries to deceive the message originator into believing that the reflected message is expected by the originator from an intended communication partner, either as a response to, or as a challenge for, the originator. If Malice is successful, the message originator either accepts an “answer” to a question which was in fact asked and answered by the originator itself, or provides Malice with an **oracle service** (see 8.3.2 for the meaning of oracle service) which Malice needs but cannot provide to himself.

After having discovered the parallel-session attack (Example 10.6) on the original Woo-Lam Protocol (Protocol 10.2), Abadi and Needham suggested a fix [AN95]: the last message sent from Trent to Bob in Protocol 10.2 should contain the identity of Alice as follows:

$$5. \text{ Trent} \rightarrow \text{Bob: } \{Alice, N_B\}_{K_{BT}} \quad (10.7.1)$$

This fix indeed removes the parallel-session attack in Example 10.6 since now if Malice still attacks that way then the following will occur:

$$5. \text{ Trent} \rightarrow \text{Bob: } \{Malice, N_B\}_{K_{BT}}$$

while Bob is expecting (10.7.1), and hence detects the attack.

However, while having the identities of the protocol participants to be explicitly specified in a protocol is definitely an important and prudent principle for developing secure authentication protocols (an issue to be addressed in a different attacking type in §10.7.7), it is only one of many things which need to be considered. Often, one countermeasure prevents one attack but introduces another. The fixed version of Abadi and Needham [AN95] for the Woo-Lam Protocol is still insecure. Their fixed version version of the Woo-Lam Protocol suffers a reflection attack which is given in Example 10.7.

Here, Malice mounts reflection attack twice: message 3 is a reflection of message 2, and message 5 is that of message 4. This attack works under an assumption that, in messages 3 and 5, Bob receives messages and cannot detect anything wrong. This assumption holds perfectly for both cases according to our agreed convention for

Example 10.7: A Reflection Attack on a “Fixed” Version of the Woo-Lam Protocol

PREMISE: Same as that of Protocol 10.2;

1. Malice(“Alice”) \rightarrow Bob: $Alice$;
2. Bob \rightarrow Malice(“Alice”): N_B ;
3. Malice(“Alice”) \rightarrow Bob: N_B ;
4. Bob \rightarrow Malice(“Trent”): $\{Alice, Bob, N_B\}_{K_{BT}}$;
5. Malice(“Trent”) \rightarrow Bob: $\{Alice, Bob, N_B\}_{K_{BT}}$;
6. Bob accepts.

CONSEQUENCE:

Bob believes that Alice is alive in a protocol run, however in fact Alice has not participated in the run at all.

Figure 10.13.

the behavior of honest principals (§10.3). First, the random chunk Bob receives in message 3 is actually Bob’s nonce sent out in message 2; however, Bob can only treat it as an unrecognizable foreign cipher chunk; to follow the protocol specification is all he can and should do. Again, the cipher chunk Bob receives in message 5 is actually one created by himself and sent out in message 4; however, Bob is stateless with respect to the message pair 4 and 5. This also follows our convention on stateless principal agreed in §10.3). Therefore Bob cannot detect the attack.

A series of fixes for the Woo-Lam Protocol proposed by Woo and Lam in [WL94] are also flawed in a similar way: they all suffer various ways of reflection attack. For the most aggressive fix: Π^f in which the identities of both user principals will be included in each ciphertext, reflection attack will still work if Bob is not sensitive about the size of a foreign cipher chunk. This is of course a reasonable assumption due to our agreement on the “stupidity” of honest principals.

A more fundamental reason for the Woo-Lam Protocol and its various fixed versions to be flawed will be investigated in §15.2.1 where we take formal approaches to developing correct authentication protocols. A correct approach to the specification of authentication protocols will be proposed in §15.2.2. The correct approach will lead to a general fix for the Woo-Lam Protocol and for many other protocols, too. We shall see in §15.2.3.2 that the Woo-Lam Protocol under the general fix (Protocol 15.2) will no longer suffer any of the attacks we have demonstrated so far.

10.7.5 Interleaving Attack

In an interleaving attack, two or more runs of a protocol are executed in an overlapping fashion under Malice's orchestration. In such an attack, Malice may compose a message and sends it out to a principal in one run, from which he expects to receive an answer; the answer may be useful for another principal in another run, and in the latter run, the answer obtained from the former run may further stimulate the latter principal to answer a question which in turn be further used in the first run, and so on.

Some authors, e.g., [BGH⁺92], consider that interleaving attack is a collective name for the previous two attacking types, i.e., parallel-session attack and reflection attack. We view these attacking types different by thinking that an interleaving attack is more sophisticated than reflection and parallel-session attacks. In order for an interleaving attack to be successful, Malice must exploit a sequentially dependent relation among messages in different runs.

The “Wiener Attack” (Example 10.2) on an early draft of the “ISO Public Key Three-Pass Mutual Authentication Protocol”, which we have seen in §10.4.2, is a good example of interleaving attack. In that attack, Malice initiates a protocol run with *A* by masquerading as *B* (message line 1); upon receipt *A*'s response (message line 2), Malice initiates a new run with *B* by masquerading as *A* (message line 1'); *B*'s response (message line 2') provides Malice with the answer that *A* is waiting for, and thus, Malice can return to and complete the run with *A*. In comparison with the parallel-session attack (e.g., Example 10.6), an interleaving attack is very sensitive to the sequence of the message exchanges.

Related to the “Wiener Attack”, the “certificate-signature-replacement attack” on the “Authentication-only STS Protocol” (Example 10.3) is another perfect interleaving attack. Also, Lowe's attack on the Needham-Schroeder Public-key Authentication Protocol (Example 2.3) is an interleaving attack.

Usually, a failure in mutual authentication can make an interleaving attack possible.

10.7.6 Attack due to Type Flaw

In a type flaw attack, Malice exploits a fact we agreed in §10.3 on an honest principal's inability to associate a message or a message component with its semantic meaning (review Remark 10.1 and Example 10.1).

Typical type flaws include a principal being tricked to mis-interpret a nonce, a timestamp or an identity into a key, etc. Mis-interpretations are likely to occur when a protocol is poorly design in that the type information of message components are not explicit. We should use a protocol proposed by Neuman and Stubblebine [NS93] to exemplify a type flaw attack [Sv93], [Car94]. First, the protocol is as follows:

1. Alice \rightarrow Bob: A, N_A ;
2. Bob \rightarrow Trent: $B, \{A, N_A, T_B\}_{K_{BT}}, N_B$;
3. Trent \rightarrow Alice: $\{B, N_A, K_{AB}, T_B\}_{K_{AT}}, \{A, K_{AB}, T_B\}_{K_{BT}}, N_B$;
4. Alice \rightarrow Bob: $\{A, K_{AB}, T_B\}_{K_{BT}}, \{N_B\}_{K_{AB}}$.

This protocol intends to let Alice and Bob achieve mutual authentication and authenticated key establishment by using a trusted service from Trent. If a nonce and a key are random numbers of the same size, then this protocol permits Malice to mount a type-flaw attack:

1. Malice("Alice") \rightarrow Bob: A, N_A ;
2. Bob \rightarrow Malice("Trent"): $B, \{A, N_A, T_B\}_{K_{BT}}, N_B$;
3. none;
4. Malice("Alice") \rightarrow Bob: $\{A, N_A, T_B\}_{K_{BT}}, \{N_B\}_{N_A}$.

In this attack, Malice uses the nonce N_A in place of the session key K_{AB} to be established, and Bob can be tricked to accept it if he cannot tell the type difference. Indeed, there is no good mechanism to prevent Bob from being fooled.

A type flaw is usually implementation dependent. If a protocol specification does not provide sufficiently explicit type information for the variables appearing in the protocol, then type flaw can be very common in implementation. Boyd [Boy90] exemplified the problem using the Otway-Rees Authentication Protocol [OR87] where he discussed the importance for avoiding hidden assumptions in cryptographic protocols.

10.7.7 Attack due to Name Omission

In authentication protocols often the names relevant for a message can be deduced from other data parts in the context, and from what encryption keys have been applied. However, when this information cannot be deduced, name omission is a blunder with serious consequences.

It seems that experts in the fields (reputable authors in cryptography, computer security and protocol design) are ready to make a name-omission blunder. Perhaps this is because of their desire to obtain an elegant protocol which should contain little redundancy. The two attacks on two versions of the STS Protocol, which we have studied in §10.6.2 and §10.6.3, respectively, are such vivid examples. The following is another.

Denning and Sacco proposed a protocol as a public-key alternative to their fix of the Needham-Schroeder Symmetric-key Authentication Protocol [DS81]. The protocol of Denning and Sacco is as follows:

1. Alice \rightarrow Trent: A, B ;
2. Trent \rightarrow Alice: $\text{Cert}_A, \text{Cert}_B$;
3. Alice \rightarrow Bob: $\text{Cert}_A, \text{Cert}_B, \{\text{sig}_A(K_{AB}, T_A)\}_{K_B}$.

In this protocol, the third message is encrypted for both secrecy and authenticity. When Bob receives the message from Alice, he sees that the session key K_{AB} should be exclusively shared between Alice and him because he sees Alice's signature and the use of his public key.

Unfortunately, nothing in this protocol guarantees such an exclusive-key sharing property. Abadi and Needham discovered a simple but rather shocking attack [AN95] in which Bob, after receiving the message from Alice, can fool another principal to believe this "property":

- 3'. Bob("Alice") \rightarrow Charlie: $\text{Cert}_A, \text{Cert}_C, \{\text{sig}_A(K_{AB}, T_A)\}_{K_C}$.

Charlie will believe that the message is from Alice, and may subsequently send a confidential message to Alice encrypted under the session key K_{AB} . Alas, Bob can read it!

The intended meaning of message line 3 is: "At time T_A , Alice says that K_{AB} is a good key for communication between Alice and Bob". The obvious way to specify this in this protocol should be:

3. Alice \rightarrow Bob: $\text{Cert}_A, \text{Cert}_B, \{\text{sig}_A(A, B, K_{AB}, T_A)\}_{K_B}$.

Making explicit the identities of participants in authentication protocols, especially making them explicit inside the scope of a cryptographic operation, must have been a “common sense” for protocol designers. However, we have witnessed that it is not rare for experienced protocol designers to neglect this “common sense”. Abadi and Needham have documented this “common sense” as one of the prudent principles for authentication protocol design [AN95]. We should quote here again this prudent principle for protocol design:

If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal’s name explicitly in the message.

Our re-emphasis of this prudent principle is not redundant: in §11.2 we shall further see name-omission blunders in the *current* version of the IKE Protocol for Internet security [HC98], even after many years of the protocol’s development by a committee of highly experienced computer security experts.

10.7.8 Attack due to Misuse of Cryptographic Services

We should mention finally a very common protocol design flaw: misuse of cryptographic services.

Misuse of cryptographic services means that a cryptographic algorithm used in a protocol provides an incorrect protection so that the needed protection is absent. This type of flaw can lead to various attacks. The following are two most common ones:

- i) Attacks due to absence of data-integrity protection. We will demonstrate an attack on a flawed protocol to illustrate the importance of data-integrity protection. Many more attacking examples of this type on public-key cryptographic schemes will be shown in Chapter 13 where we study the notion of security against adaptively active attackers. We shall further study this type of protocol failure in depth in §15.2 where we study a topic of formal approaches to authentication protocols analysis.
- ii) Confidentiality failure due to absence of “semantic security” protection. In this type of protocol (and cryptosystem) failure, Malice can extract some partial information about a secret message encrypted in a ciphertext and hence achieves his attacking agenda without fully breaking an encryption algorithm in terms of a “all-or-nothing” quality of confidentiality (see Property 8.2). We shall study the notion of semantic security in Chapter 13 and show many such attacks there. There and in Chapter 14 we shall also study cryptographic techniques which offer semantic security.

Now let us demonstrate a flaw due to missing of integrity service. The flawed protocol is a variation of the Otway-Rees Protocol [OR87]. We have derived the

variation by following a suggestion in [BAN89]. The variation is specified in Protocol 10.8.

Protocol 10.8: A Minor Variation of Otway-Rees Protocol

PREMISE: Alice and Trent share key K_{AT} ;
Bob and Trent share key K_{BT} ;

GOAL: Alice and Bob authenticate to each other;
they also establish a new and shared session key K_{AB} .

1. Alice \rightarrow Bob : $M, Alice, Bob, \{N_A, M, Alice, Bob\}_{K_{AT}}$;
2. Bob \rightarrow Trent : $M, Alice, Bob, \{N_A, M, Alice, Bob\}_{K_{AT}},$
 $\{N_B\}_{K_{BT}}, \{M, Alice, Bob\}_{K_{BT}}$;
3. Trent \rightarrow Bob : $M, \{N_A, K_{AB}\}_{K_{AT}}, \{N_B, K_{AB}\}_{K_{BT}}$;
4. Bob \rightarrow Alice : $M, \{N_A, K_{AB}\}_{K_{AT}}$.

(* M is called a run identifier for Alice and Bob to keep tracking the run between them *)

Figure 10.14.

Protocol 10.8 applies rather standard technique of on-line authentication server (Trent) to achieve mutual authentication and authenticated session establishment between two user principals. Let us consider Bob's view on a protocol run (Alice's view can be considered likewise). Bob can conclude that the session received in in step 3 is fresh from the cryptographic integration between the key and his nonce. He should also be able to conclude that the session key is shared with Alice. This is implied by the cryptographic integration between the run identifier M and the two principals' identities; the integration has been created by Bob himself and has been verified by Trent.

The variation differs from the original Otway-Rees Protocol only very slightly: in step 2 of the variation, Bob's encrypted messages (encrypted under the key K_{AT}) are in two separate cipher chunks, one encrypts his nonce N_B , the other encrypts other message components. In the original Otway-Rees Protocol, the nonce and the rest of the message components are encrypted (more precisely, specified to be encrypted) inside one cipher chunk: $\{N_B, M, Alice, Bob\}_{K_{BT}}$.

It is interesting to point out that for some implementors, this variation may not qualify a variation at all: encryption of a long message is always implemented in a plural number of blocks whether or not the specification uses one chunk or two chunks. This is an important point and we shall return to clarify it at the end of our discussion of this type of protocol failure.

This minor variation is actually a much less aggressive version of modification to the original protocol than that suggested in [BAN89]. There it is considered that Bob's nonce needn't be a secret, and hence Bob can send it in cleartext. Indeed, if the freshness identifier has been sent in cleartext in step 2, Bob can of course still use N_B returned in step 3 to identify the freshness of the session key K_{AB} . We, however, insist on using encryption in step 2 in order to expose our point more clearly.

Protocol 10.8 is fatally flawed. Example 10.8 shows an attack. This attack was discovered by Boyd and Mao [BM93].

In this attack, Malice begins with masquerading Alice to initiate a run with Bob. He then intercepts the message from Bob to Trent (step 2); he changes Alice's identity into his own, does not touch Bob's first cipher chunk (no need for him to know the encrypted nonce) and replaces Bob's second cipher chunk $\{M, Alice, Bob\}_{K_{BT}}$ with an old chunk $\{M, Malice, Bob\}_{K_{BT}}$ which he has recorded from a previous normal run of the protocol between himself and Bob. After sending the modified messages to Trent by masquerading as Bob (step 2'), everything will go fine with Trent and Bob: Trent thinks that the two client-users requesting for authentication service are Malice and Bob, while Bob thinks that the run is between Alice and himself. Alas, Bob will use the established session key which he thinks to share with Alice but in fact with Malice, and will disclose to Malice the confidential messages which should be sent to Alice!

This attack reveals an important point: to protect the freshness identifier N_B in terms of confidentiality is to provide a wrong cryptographic service! The correct service is data integrity which must be provided to integrate the nonce and the principals identities. N_B can indeed be sent in clear if a proper integrity protection is in place. Without integrity protection, encryption of N_B is missing the point!

We have mentioned that for some implementors, Protocol 10.8 will not be viewed as a variation from the original Otway-Rees Protocol. This is true indeed because, encryption of a long message is always implemented in a plural number of blocks. If in an implementation, a plural number of ciphertext blocks are not integrated one another cryptographically, then both protocols will be implemented into the same code, and hence one cannot be a "variation" of the other.

In the usual and standard implementation of block ciphers, a sequence of separate ciphertext blocks are cryptographically chained one another. The cipher-block-chaining (CBC, see §7.7.2) mode of operation is the most likely case. We should notice that in the CBC mode, the cryptographically chained cipher blocks are ac-

Example 10.8: An Attack on a Variant Otway-Rees Protocol

PREMISE: In addition to that in Protocol 10.8,
 Malice and Trent share key K_{MT} .
 (* So Malice is also a normal user in the system *)

1. Malice(“Alice”) \rightarrow Bob: $M, Alice, Bob, \{N_M, M, Malice, Bob\}_{K_{MT}}$;
2. Bob \rightarrow Malice(“Trent”): $M, Alice, Bob, \{N_M, M, Malice, Bob\}_{K_{MT}}$,

$$\{N_B\}_{K_{BT}}, \{M, Alice, Bob\}_{K_{BT}};$$

- 2'. Malice(“Bob”) \rightarrow Trent: $M, Malice, Bob, \{N_M, M, Malice, Bob\}_{K_{MT}}$,

$$\{N_B\}_{K_{BT}}, \{M, Malice, Bob\}_{K_{BT}};$$

(* where $\{M, Malice, Bob\}_{K_{BT}}$ is an old cipher chunk which Malice preserves from a previous normal run between himself and Bob. *)

3. Trent \rightarrow Bob: $M, \{N_M, K_{MB}\}_{K_{MT}}, \{N_B, K_{MB}\}_{K_{BT}}$;
- 4 Bob \rightarrow Malice(“Alice”): $M, \{N_M, K_{MB}\}_{K_{MT}}$.

CONSEQUENCE:

Bob believes that he has been talking to Alice and shares a session key with her. However, in fact he has been talking to Malice and shares the session key with the latter.

Figure 10.15.

tually not protected in terms of data integrity service, as in contrast to a common mis-belief. Without integrity protection, some of the chained blocks can be modified without having the modification to be detected during decryption time. We shall show how CBC misses the point of providing data-integrity protection in §15.2.1.2.

Not End of List

It is still possible to further name several other ways to attack authentication protocols, such as “implementation dependent”, “binding”, “encapsulation” attacks (see §4 of [CJ97]) or “misplaced trust in server” attack (see §12.9.1 of [MvOV97]), etc. Because some of these types of attacks have certain overlapping parts with some of

the types we have listed, also because, even including them, we still cannot exhaust all possible types of attacks, we should therefore stop our listing here.

The readiness for authentication protocols to contain security flaws, even under the great care of experts in the fields, have urged researchers to consider to take systematic approaches to design and analysis of authentication protocols. In Chapter 15 we shall study several topics on formal methods for design and analysis of authentication protocols.

10.8 A Brief Literature Note

Authentication is a big subject in cryptographic protocols. We recommend a few important literature references in this subject.

- A logic of authentication by Burrows, Abadi and Needham [BAN89]. This seminal paper is essential reading. Most security protocol papers reference it. It is a good source of many early authentication protocols and an early exposure of many security flaws with them.
- A survey on various ways cryptographic protocols fails by Moore [Moo88], [Moo92]. This is an important paper. It is a good introduction to various cryptographic failures which are not a result of any inherent weakness in the cryptographic algorithms themselves, rather it is because the way in which they are used requires that they provide certain cryptographic services which they do not in fact provide.
- Prudent engineering practice for cryptographic protocols summarized by Abadi and Needham [AN95]. This paper sets out eleven heuristic principles which intend to guide protocol designers to develop good protocols. The principles form an engineering account of protocol development, serving a menu for protocol designers: “have I checked this sort of attack?” An excellent piece of work and will prove of considerable use for protocol designers.
- A survey of authentication protocol literature written by Clark and Jacob [CJ97]. This document includes a library of numerous authentication and key establishment protocols, which not only include protocols appeared in research papers, but also real-world ones such as **Kerberos** [MNSS87], [JC93] (we will introduce the Kerberos Protocol in §11.4) and many from the ISO/IEC standardization documents. Many of the protocols in the library are accompanied with attacks. The document also has a comprehensive and well annotated literature survey. This is an essential reading for protocol developers. A Web site “Security Protocols Open Repository” (SPORE) has been setup as the further development of the document of Clark and Jacob. The Web address for SPORE is <http://www.lsv.ens-cachan.fr/spore/>
- A new book of Boyd...

10.9 Chapter Summary

Our study of authentication in this chapter covers wide and in-depth topics. These include basic concepts (data-origin, entity, authenticated-key-establishment, unilateral, mutual, liveness), standard and good constructions of authentication protocols (recommended by the international standards), several interesting and useful protocols (one-time password, EKE, STS) and a taxonomy of attacks.

As an active academic research topic, authentication is an important but rather a pre-mature subject in the area of cryptographic protocols. Our coverage of the subject in this chapter is by no means comprehensive. We have listed a brief literature note for some readers who will be interested in a further study of the subject in academic research direction. We will also conduct a further study on formal analysis of authentication in a later chapter.

Authentication protocols have importance in real world applications. This chapter has touched little aspects in applications. Let us turn to the real world applications of authentication protocols in the next chapter.

AUTHENTICATION PROTOCOLS FOR THE REAL WORLD

11.1 Introduction

Our study of authentication protocols in the preceding chapter has an academic focus: we have studied good (and standard) constructions for authentication protocols, introduced a few important authentication protocols and techniques selected from the literature, and conducted a systematic examination of various “academic attacks” on authentication protocols. However, we have touched little application aspect. Undoubtedly, real world applications of authentication protocols must have real world problems to solve, some of which are very challenging.

In this chapter, let us face some authentication problems in the real world. We shall introduce and discuss a number of authentication protocols which have been proposed for, and some already in use in, various important applications in the real world.

The first real world protocol we shall study is the Internet Key Exchange Protocol (IKE) [HC98], [Kau02b] which is the authentication mechanism for an industrial standard for Internet Security (IPSec). This protocol (suite) contains authentication and authenticated key exchange protocols which operate at a low layer of communications called the network layer. Our study shall let us see how communications take place at the network layer, understand how various attacks demonstrated in the previous chapter can be based on various ways for Malice to manipulate addressing information handled by the network-layer protocol, and realize that security offered at the network layer can be very effective in thwarting those attacks. We shall also see that a challenging problem in IKE is for the protocol suite to serve an optional privacy feature which is desirable at the network layer of communications in order not to cause privacy damage to applications in higher layers of communications.

Next, we shall introduce the Secure Shell (SSH) Protocol [Ylo95], [Ylo02c], [Ylo02d], [Ylo02a], [Ylo02b]. This is a public-key based authentication protocol for achieving secure access of remote computer resource (secure remote login) from an untrusted machine (i.e., from an untrusted client machine to a remote server host). SSH mainly runs on computer platforms which use UNIX^a, or its popular variant, Linux, operating systems. A challenging problem for this protocol is to enable a security service in a harmonic manner: insecure systems are already in wide use, secure solutions should be added on with the least interruption to the insecure systems which are already in operation (backward compatibility).

Next, we shall introduce another important and already-in-wide-use authentication protocol: the Kerberos Authentication Protocol [MNSS87], [JC93]. This is the network authentication basis for another popular operating system: Windows 2000. This operating system is in wide use in an enterprise environment where a user is entitled to using enterprise-wide distributed services while is unable to keep many different cryptographic credentials for using the different servers (it is unrealistic for a user to memorize many different passwords and is uneconomic for a user to manage many smartcards). We shall see that Kerberos' "single-sign-on" authentication architecture finds a good application in such an environment.

Finally, we shall overview the Secure Socket Layer (SSL) Protocol [Hic95], or the Transport Layer Security (TLS) Protocol named by the Internet-wise industrial standard community Internet Engineering Task Force (IETF). At the time of writing, this protocol qualifies the most-wide-in-use public-key based authentication technique: it is nowadays an integral part in every World Wide Web browser and Web server, though most cases of its use is limited to unilateral authentication only. This is an authentication protocol for a typical client-server environment setting. Although the idea behind the protocol is extremely simple (this is the simplest protocol of the four real-world authentication protocols to be introduced in this chapter), we shall see from this case that a real-world realization of any simple authentication protocol is never a simple job.

11.1.1 Chapter Outline

IPSec and the IKE Protocol will be introduced in §11.2. The SSH Protocol will be introduced in §11.3. An enterprise single-sign-on scenario suitable for using the Windows 2000 operating system will be discussed in §11.4, and then the Kerberos Protocol, the network authentication basis for this operating system, will be described. Finally we overview the SSL (TLS) Protocol in §11.5.

^a UNIX is a trademark of Bell Laboratories.

11.2 Authentication Protocols for Internet Security

We have been introducing various cryptographic techniques for protecting messages transmitted through open networks. The techniques introduced so far in this book all provide protections at a high level or the application level. A high-level protection means that only the content part of a message is protected whereas the addressing part, regarded as low-level information, is not.

However, for securing communications over the Internet, protection provided at a low level of communication which covers the addressing information as well as the content can be very effective. This is because, as we have witnessed in §10.7, manipulation of a message's addressing information is the main source of tricks available to Malice for mounting various attacks.

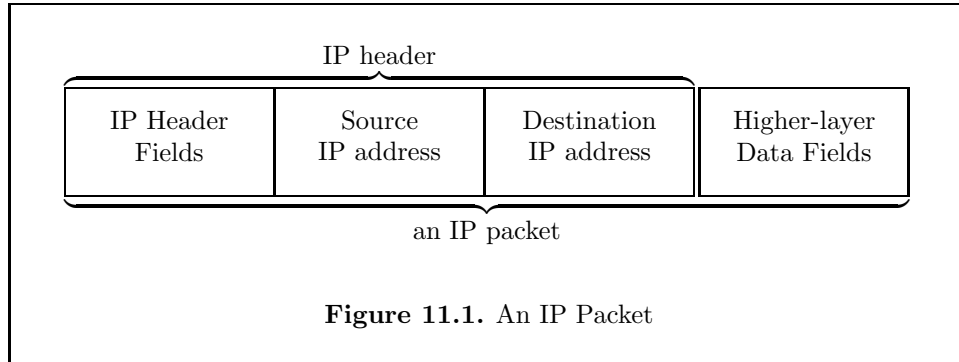
In this section we shall first look at how messages are processed by a low-layer communication protocol. There, we shall realize how Malice could materialize his tricks due to absence of security in that protocol. We shall then study a suite of authentication protocols proposed by a standard body for Internet security. That suite of protocols is collectively named the **Internet Key Exchange (IKE)**; they are intended to protect messages in the low-layer communication protocol using some authentication techniques we have introduced in this chapter. We shall analyze a couple of important “modes” in IKE and reveal some vulnerabilities they have. We shall also report some critical comments and concerns on IKE from the research community.

11.2.1 Communications at the Internet Protocol Layer

The Internet is an enormous open network of computers and devices called “nodes”. Each node is assigned a unique network address so that messages sent to and from the node are attached with this address. A protocol which processes the transmission of messages using the network address is called the **Internet Protocol (IP)** for short) and hence, the unique network address of a node is called the IP address of the node. According to the ISO's “Open Systems Interconnection (ISO-OSI) Seven-layer Reference Model” (e.g., pages 416–417 of [Oxf91] or §1.5.1 of [KPS02]), the IP works at “layer 3” (also called the network layer or the IP layer). The authentication protocols introduced prior to this section are all working at “layer 7” (also called the application layer). This is another reason why we have called the IP a “low-level” communication protocol and the other protocols “high-level” ones.

Communications at the IP layer take the form of “IP packets”. Figure 11.1 illustrates an IP packet.

Let us use email communication to exemplify so organized Internet communications. Let `James_Bond@007.654.321` and `Miss_Moneypenny@123.456.700` be two email addresses. Here, `James_Bond` and `Miss_Moneypenny` are users' identities, each



is called an “endpoint identity”, 007.654.321 and 123.456.700 are two IP addresses^b; for example, the former can be the IP address of a palm-top multi-purpose device, while the latter can be the IP address of an office computer. An email sent from Miss_Moneypenny@123.456.700 to James_Bond@007.654.321 viewed at the IP layer can be

IP Header Fields	123.456.700	007.654.321	Dear James,
---------------------	-------------	-------------	---------------------------

Notice that the two endpoint identities will appear in some “IP Header Fields”, and hence when James_Bond receives the email, he may know who has sent it and may be able to reply.

These two parties may wish to conduct confidential communications by applying end-to-end encryption, using either a shared key or public keys. Since an end-to-end encryption operates in the application-layer protocol, only the message content in the fourth box in the IP packet will be encrypted. If the IP they use offers no security, then the data fields in “IP header” are not protected. Modification of data in these fields forms the main source of tricks behind the attacks which we have listed in §10.7. Let us now see how.

11.2.2 Internet Protocol Security (IPSec)

The Internet Engineering Task Force (IETF) has been in a standardization process for a suite of authenticated key exchange protocols for securing real-time communications in the IP layer [HC98]. This standardization process is widely known as **IPSec**. Briefly speaking, IPSec is to add security to “IP header” which consists of

^bOften an IP address is mapped to a “distinguished name” for ease of memory; for instance, 007.654.321 may be mapped to the following distinguished name: spy1.mi.five.gb.

the first three boxes in an IP packet (see Figure 11.1). IPSec stipulates a mandatory authentication protection for “IP header” and an optional confidentiality protection for the endpoint-identity information which is in some “IP Header Fields”.

We should notice that, in absence of security at the IP layer, it is the unprotected transmission of “IP header” that may permit Malice to mount various attacks on Internet communications such as spoofing (masquerading), sniffing (eavesdropping) and session hijacking (combination of spoofing and sniffing while taking over a legitimate party’s communication session). For example, if Malice intercepts an IP packet originated from `James_Bond@007.654.321`, copies “Source IP address” to “Destination IP address”, and sends it out, the packet will go back to `James_Bond@007.654.321`. If this modification is undetected due to lack of security means at the IP layer, then the modification can essentially cause a “reflection attack” which we have seen in §10.7.4. Moreover, if Malice also forges a “Source IP address” and an endpoint identity (say “`Miss_Moneypenny`”), then `James_Bond@007.654.321` may be fooled to believe as if the message came from the forged sender. This is exactly the attack scenario which we have denoted at the application layer by

$$\text{Malice}(\text{“Miss_Moneypenny”}) \rightarrow \text{James_Bond: ...}$$

Virtually all attacks which we have seen in §10.7 require Malice to perform some manipulations on the IP-address and endpoint-identity information in an “IP header”. Security protection offered at the IP layer can therefore effectively prevent such attacks since now any message manipulation in “IP header” can be detected. In general, security at the IP layer can provide a wide protection on any applications at higher layers.

Moreover, for traffics between two **firewalls**^c, because each firewall is a node which shields many nodes “inside” or “behind” it, an IP-layer protection can cause encryption on the IP address of any node “inside” the firewall. This means that unauthorized penetration through a firewall can be prevented via cryptographic means which is a very strong form of protection. Without security offered at the IP layer, the firewall technique uses much weaker form of “secrets” such as IP addresses, passwords, machine names and user names, etc., and so penetration is much easier.

It has been widely agreed that offering security at the IP layer is a wise thing to do.

^cA firewall is a special-purpose computer which connects a cluster of protected computers and devices to the Internet so that accessing the protected computers and devices from the Internet requires knowing some identity and IP address information.

11.2.3 The Internet Key Exchange (IKE) Protocol

The Internet Key Exchange (IKE) Protocol [HC98] is the current authenticated key exchange protocol standard for IPSec. It is a suite of authentication and authenticated key exchange protocols. Each protocol in the suite is a hybrid one which uses part of “Oakley” (the Oakley Key Determination Protocol [Orm96]), part of “SKEME” (A Versatile Secure Key Exchange Mechanism for Internet [Kra96]) and part of “ISAKMP” (the Internet Security Association and Key Management Protocol [MSST98]).

Oakley describes a series of key exchanges – called “modes” – and gives details of the services provided by each (e.g., perfect forward secrecy for session keys, endpoint identity hiding, and mutual authentication). SKEME describes an authenticated key exchange technique which supports deniability of connections between communication partners (due to using shared key, a feature adopted in IKE and IKE v2, to be discussed in a moment) and quick key refreshment. ISAKMP provides a common framework for two communication parties to achieve authentication and authenticated key exchange, to negotiate and agree on various security attributes, cryptographic algorithms, security parameters, authentication mechanisms, etc., which are collectively called “Security Associations (SAs)”. However ISAKMP does not define any specific key exchange technique so that it can support many different key exchange techniques.

As a hybrid of these works, IKE can be thought of as a set of two-party protocols, featuring authenticated session key exchange, most of which using the Diffie-Hellman key exchange mechanism, having many options for the two participants to negotiate and agree upon in an on-line fashion.

The IKE Protocol consists of two phases, called “Phase 1” and “Phase 2”, respectively.

Phase 1 assumes that each of the two parties involved in a key exchange has an identity by which the other party knows. Associated with that identity is some sort of cryptographic capability which can be shown to the other party. This capability might be enabled by a pre-shared secret key for a symmetric cryptosystem, or by a private key matching a reliable copy of a public key for a public-key cryptosystem. Phase 1 attempts to achieve mutual authentication^d based on showing that cryptographic capability, and establishes a shared session key which is used in the current run of Phase 1, can be used to protect Phase 2 exchanges if they are needed, or can be further used to secure higher-level communications as an output from the IKE phases of exchanges.

A multiple number of Phase 2 exchanges may take place after a Phase 1 exchange between the same pair of entities involved in Phase 1. Phase 2 is often referred to as

^dWe shall see in a moment that some modes in IKE Phase 1 fail to achieve mutual authentication in that an entity may be fooled perfectly to believe sharing a session key with an intended party, whereas actually sharing it with another party.

“Quick Mode”. It relies on the shared session key agreed in Phase 1. The reason for having a multiple number of Phase 2 exchanges is that they allow the users to set up multiple connections with different security properties, such as “integrity-only”, “confidentiality-only”, “encryption with a short” or “encryption with a strong key”.

To see a flavor of IKE, let us focus our attention only on a couple of IKE Phase 1 modes.

11.2.3.1 IKE phase 1

There are eight variants for the IKE Phase 1. This is because there are three types of keys (pre-shared symmetric key, public key for encryption, and public key for signature verification), and in addition there are two versions of protocols based on public encryption keys, one of which is intended to replace the other, but the first must still be documented for backward compatibility. Thus there are actually four types of keys (pre-shared symmetric key, old-style public encryption key, new-style public encryption key, and public signature-verification key). For each key type there are two types of Phase 1 exchanges: a “main mode” and an “aggressive mode”.

Each main mode has six messages exchanges; 3 messages sent from an initiator (I for short) to a responder (R for short), 3 sent from R to I . A main mode is mandatory in IKE, that is, two users cannot run an aggressive mode without running a main mode first.

Each aggressive mode has only three messages; I initiates a message, R responds one, then I sends a final message to terminate a run. An aggressive mode is optional, that is, it can be omitted.

For IKE Phase 1, we shall only describe and analyze “signature based modes”. Other modes generally use an encryption-then-decryption of freshness identifier mechanism for achieving authentication; we have labelled such a mechanism non-standard (see §10.4.1.5) which we will further criticize in §15.2.

11.2.3.2 Signature based IKE phase 1 main mode

Signature Based IKE Phase 1 Main Mode (also named “Authenticated with Signatures”, §5.1 of [HC98]) is specified in Protocol 11.1. This mode is born under the influence of several protocols, however, its real root can be traced back to two protocols: the STS Protocol (Protocol 10.6), and a protocol proposed by Krawczyk [Kra] named SIGMA Protocol (we shall discuss SIGMA design in §11.2.4).

In the first pair of messages exchange I sends to R HDR_I and SA_I , and R responds with HDR_R and SA_R . The header messages HDR_I and HDR_R include “cookies” C_I and C_R ; the former is for R to keep the run (session) state information for I , and vice versa for the latter. Of the two Security Associations, SA_I specifies a list of security attributes that I would like to use; SA_R specifies ones chosen by

Protocol 11.1: Signature Based IKE Phase 1 Main Mode

1. $I \rightarrow R$: HDR_I, SA_I
2. $R \rightarrow I$: HDR_R, SA_R
3. $I \rightarrow R$: HDR_I, g^x , N_I
4. $R \rightarrow I$: HDR_R, g^y , N_R
5. $I \rightarrow R$: HDR_I, {ID_I, Cert_I, Sig_I} _{g^{xy}}
6. $R \rightarrow I$: HDR_R, {ID_R, Cert_R, Sig_R} _{g^{xy}}

Notation (*) For ease of exposition, we omitted some minute details. Our omission will not effect the functionality of the protocol, in particular, will not effect an attack we shall describe in a moment. *)

I, R : An initiator and a responder, respectively.

HDR_I, HDR_R: Message headers of I and R , respectively. These data contain C_I, C_R which are “cookies”^a of I and R , respectively, which are for keeping the session state information for these two entities.

SA_I, SA_R: Security Associations of I and R , respectively. The two entities use SA_I, SA_R to negotiate parameters to be used in the current run of the protocol; negotiable things include: encryption algorithms, signature algorithms, pseudo-random functions for hashing messages to be signed, etc. I may propose multiple options, whereas R must reply with only one choice.

g^x, g^y : Diffie-Hellman key agreement material of I and R , respectively.

ID_I, ID_R: Endpoint identities of I and R , respectively.

N_I, N_R : Nonces of I and R , respectively.

Sig_I, Sig_R: Signature created by I and R , respectively. The signed messages are M_I and M_R , respectively, where

$$M_I = \text{prf}_1(\text{prf}_2(N_I|N_R|g^{xy})|g^x|g^y|C_I|C_R|SA_I|ID_I)$$

$$M_R = \text{prf}_1(\text{prf}_2(N_I|N_R|g^{xy})|g^y|g^x|C_R|C_I|SA_R|ID_R)$$

where prf_1 and prf_2 are pseudo random functions agreed in SAs.

^aA “cookie” is a text-only string that gets entered into a remote host system’s memory or saved to file there for the purpose of keeping the state information for a client-server communication session.

Figure 11.2.

R.

The second pair of messages consists of the Diffie-Hellman key exchange material.

In message 5 and 6, the algorithms for encryption, signature and pseudo-random functions for hashing messages to be signed are the ones agreed in the SAs.

Signature Based IKE Phase 1 Main Mode has some similarity to the STS Protocol (Protocol 10.6). However, two significant differences can be spotted:

- i) The STS Protocol leaves the certificates outside of the encryptions, whereas here the certificates are inside the encryptions. Encryption of the certificates allows an anonymity feature which we have discussed when we introduced the STS Protocol (see page 305). This is possible and a useful feature for *I* and/or *R* being endpoints inside firewalls.
- ii) Signatures in the STS Protocol does not involve the agreed session key, whereas here a signed message is input to a pseudo random function prf which is also seeded by the agreed session key g^{xy} . Hence in this mode, the signatures are exclusively verifiable by the parties who have agreed the shared session key.

11.2.3.3 Authentication failure in signature based IKE phase 1 main mode

Similar to the situation in the STS Protocol, a signed message in this mode of IKE only links to the endpoint identity of the signer, and not also to that of the intended communication partner. The missing of this specific explicitness also makes this mode suffering from an authentication-failure flaw similar to that we have demonstrated on the STS Protocol (Example 10.4). The flaw is illustrated in Example 11.1.

With this flaw, Malice can successfully fool *R* into believing that *I* has initiated and completed a run with it. However in fact *I* did not do so. Notice that *R* is fooled perfectly in that not only it accepts a wrong communication partner and believes sharing a key with the wrong partner, but also that nobody will report anything abnormal to *R*. So Example 11.1 indeed demonstrates an authentication failure.

The authentication-failure attack can also be called a “denial of service attack” for the following reason. In IKE, after a successful Phase 1 exchange, a server in the position of *R* will keep the current state with *I* so that they may use the agreed session key for further engagement in a multiple number of Phase 2 exchanges. However, after an attack run shown in Example 11.1, *I* will never come to *R* and hence, *R* may keep the state, allocate resource with *I* and wait for *I* to come back for further exchanges. If Malice mounts this attack in a distributed manner, using a large team of his fiends over the Internet to target a single server at the same time, then the server’s capacity to serve other honest nodes can be drastically reduced or

Example 11.1: Authentication Failure in Signature Based IKE Phase 1 Main Mode

(* Malice faces I using his true identity, but he faces R by masquerading as I : *)

1. $I \rightarrow \text{Malice}$: $\text{HDR}_I, \text{SA}_I$
 - 1' $\text{Malice}("I") \rightarrow R$: $\text{HDR}_I, \text{SA}_I$
 - 2' $R \rightarrow \text{Malice}("I")$: $\text{HDR}_R, \text{SA}_R$
2. $\text{Malice} \rightarrow I$: $\text{HDR}_R, \text{SA}_R$
3. $I \rightarrow \text{Malice}$: HDR_I, g^x, N_I
 - 3' $\text{Malice}("I") \rightarrow R$: HDR_I, g^x, N_I
 - 4' $R \rightarrow \text{Malice}("I")$: HDR_R, g^y, N_R
4. $\text{Malice} \rightarrow I$: HDR_R, g^y, N_R
5. $I \rightarrow \text{Malice}$: $\text{HDR}_I, \{\text{ID}_I, \text{Cert}_I, \text{Sig}_I\}_{g^{xy}}$
 - 5' $\text{Malice}("I") \rightarrow R$: $\text{HDR}_I, \{\text{ID}_I, \text{Cert}_I, \text{Sig}_I\}_{g^{xy}}$
 - 6' $R \rightarrow \text{Malice}("I")$: $\text{HDR}_R, \{\text{ID}_R, \text{Cert}_R, \text{Sig}_R\}_{g^{xy}}$
6. Dropped.

CONSEQUENCE:

R is fooled perfectly and thinks to have been talking and sharing a session key with I , while I thinks to have been talking with Malice in an incomplete run. R will never be notified of any abnormality and may either be denied a service from I , or be awaiting permanently for a service request from I .

Figure 11.3.

even nullified. Notice that this attack does not demand sophisticated manipulation nor complex computation from Malice and his distributed friends, and hence the distributed denial of service attack can be very effective.

This attack works because a signed message in the protocol only contains the identity of the signer, and so it can be used to fool a principal who is not the intended communication partner of the signer. If both two endpoint identities of

the intended principals are included in a signed message, then the message becomes specific to these two principals, and hence cannot be used for any other purpose.

We have witnessed again the generality of attacks due to name omission.

11.2.3.4 Signature based IKE phase 1 aggressive mode

Signature Based IKE Phase 1 Aggressive Mode is a cut-down simplification from Main Mode: it does not use encryption and has three message exchanges instead of six. This aggressive Mode is specified as follows (the notation is the same as that in its Main Mode (see Protocol 11.1):

1. $I \rightarrow R$: $\text{HDR}_I, \text{SA}_I, g^x, N_I, \text{ID}_I$
2. $R \rightarrow I$: $\text{HDR}_R, \text{SA}_R, g^y, N_R, \text{ID}_R, \text{Cert}_R, \text{Sig}_R$
3. $I \rightarrow R$: $\text{HDR}_R, \text{Cert}_I, \text{Sig}_I$

At first glance, this mode is very similar to “Authentication-only STS Protocol” (Protocol 10.7) due to omission of encryption. A more closer look exposes a difference: in “Authentication-only STS Protocol”, signed messages do not involve the session key, whereas here, a signed message is input to pseudo random function prf which is also seeded by the agreed session key g^{xy} . So in this mode, the signatures are exclusively verifiable by the principals who holds the agreed session key. This difference prevents the “certificate-signature-replacement attack” (Example 10.3) from being applied to this mode.

However, this mode fails to achieve mutual authentication in a different way. A similar “denial of service attack” applies to this mode. It is essentially Lowe’s attack on the STS Protocol (Example 10.4). Now it is I who can be fooled perfectly to believe having been talking and sharing a session key with R , whereas R does not agree so. We shall leave the concrete construction of the attack as an exercise for the reader.

We should further notice that if the signature scheme used in this mode features message recovery, then Malice can gain more. For example, from a signed message Malice can obtain $\text{prf}_2(N_I | N_R | g^{xy})$ and so he can use this material to create his own signature using his own certificate and identity. Thus he can mount a “certificate-signature-replacement attack” which we have seen in Example 10.3 against the “Authentication-only STS Protocol”. Such an attack is a perfect one because the both interleaved runs which Malice orchestrates in between I and R will terminate successfully and so none of the two honest entities can find anything wrong. Notice that some signature schemes do feature message recovery (e.g., [NR93] which is even standardized [ISO00]). Therefore, it is not impossible for the two communication partners to have negotiated to use a signature scheme with message recovery feature. In §11.2.5, we shall discuss the IKE’s feature of supporting flexible options.

Without using encryption or MAC, the IKE's Aggressive Mode cannot have a "plausible deniability feature" which we shall discuss in §11.2.4. When this feature is not needed, a fix for the authentication-failure flaw is standard: both two endpoint identities of the intended principals should be included inside the both signatures so that the signed messages are unusable in any context other than this mode between the intended principals.

Methods for fixing authentication failure while keeping a deniability feature will be discussed in §11.2.4.

11.2.3.5 Other security analysis on IPSec and IKE

Several researchers have conducted security analysis work on IKE.

Meadows, using her NRL Protocol Analyzer (an automated exhaustive flaw checker, to study in §15.5.2 [Mea96b], [Mea96a]), has discovered that the Quick Mode (an IKE Phase 2 exchange) is vulnerable to a reflection attack [Mea99].

Ferguson and Schneier has conducted a comprehensive cryptographic evaluation for IPSec [FS00].

A few years back (1996), Bellare made an analysis on a serious problem with IPSec: an option for an IPSec mode in which ciphertext messages are not protected in terms of data integrity [Bel96]. We have seen through an attacking example and known that confidentiality without integrity is a complete missing of point (§10.7.8). We shall further see in a few later chapters (Chapters 13 – 15) that most encryption algorithms cannot provide proper confidentiality protection if ciphertext messages they output are not also protected in terms of data integrity. However, this dangerous option seems to remain unnoticed by the IPSec community (see below), maybe due to the high system complexity in the specifications for IPSec.

11.2.4 A Plausible Deniability Feature in IKE

At the time of writing, IKE Version 2 specification has been published [Kau02b]. A feature which is adopted as an option in IKE v2 is called "plausible deniability" [Hof02] of communications by an entity who may have been involved in a connection with a communication partner.

This feature, which originates from the SIGMA protocol construction of Krawczyk (see an explanation in [Kra]), and Canetti and Krawczyk [CK02], permits an entity to deny "plausibly" the existence of a connection with a communication partner. Offering such a denying-of-a-connection feature at the IP layer is desirable because it permits various fancy privacy services, such as anonymity, to be offered at the higher layers with uncompromised quality. A privacy damage caused at the IP layer can cause irreparable privacy damage at the application layer. For example, an identity connected to an IP address, if not deniable, certainly nullifies an anonymous quality offered by a fancy cryptographic protocol running at the application

level.

The “plausible deniability” feature in the SIGMA design can be described by following two message lines in the position of message lines 5 and 6 in Protocol 11.1:

$$I \rightarrow R : s, \text{ID}_I, \text{Sig}_I(\text{“1”}, s, g^x, g^y), \text{MAC}(g^{xy}, \text{“1”}, s, \text{ID}_I)$$

$$R \rightarrow I : s, \text{ID}_R, \text{Sig}_R(\text{“0”}, s, g^y, g^x), \text{MAC}(g^{xy}, \text{“0”}, s, \text{ID}_R)$$

Here (s is session identifier) both parties can verify the respective signature and then use the shared session key to verify the respective MAC, and hence are convinced that the other end is the intended communication partner. Now, if they dispose of the session key then they cannot later prove to a third party that there was a connection between them.

It is not difficult to see that this construction contains the authentication-failure flaw demonstrated in Example 11.1. Canetti and Krawczyk did anticipate a less interesting form of attack in which Malice simply prevents the final message from reaching I . They suggested a method for preventing this “cutting-final-message attack” by adding a final acknowledgement message from I to R (see Remark 2 in [CK02]). Since now R (who is normally in the server’s position) receives the final message, the “cutting-final-message attack” will be detected by R and hence upon occurrence of the attack, R should reset the state and release the resources. In this way, the protocol is less vulnerable to a denial of service attack. The final acknowledgement may have a useful side-effect of preventing the authentication-failure flaw (depending on the cryptographic formulation of the acknowledgement message). But clearly this method of fixing the protocol is not particularly desirable, since it involves additional traffic and protocol complexity.

Since a deniability feature is useful, we should keep it while fixing the authentication failure flaw. We suggest to augment the SIGMA design into the following one:

$$I \rightarrow R : s, \text{ID}_I, \text{Sig}_I(\text{“1”}, s, g^x, g^y), \text{MAC}(g^{xy}, \text{“1”}, s, \text{ID}_I, \text{ID}_R)$$

$$R \rightarrow I : s, \text{ID}_R, \text{Sig}_R(\text{“0”}, s, g^y, g^x), \text{MAC}(g^{xy}, \text{“0”}, s, \text{ID}_R, \text{ID}_I)$$

Namely, the two principals should still not explicitly sign their identities and so to retain the “plausible deniability” feature, however, they should *explicitly* verify the both intended identities inside the MACs.

Notice that this denying-of-a-connection feature is not in a high quality because a party (call it the traitor) who keeps the session key g^{xy} can later still show to a third party the evidence that a named (authenticated) entity has been involved in this connection. This is clearly possible since the traitor can use exactly the

same verification operations it has used when the two parties were in the authentication connection. That is why the deniability must be prefixed by the modifier “plausible”.

In a later chapter we will introduce a new and practical cryptographic primitive which can provide a deniable authentication service in an absolute sense.

11.2.5 Critiques on IPSec and IKE

The most prominent criticism on IPSec and IKE is on their intensive system complexity and lack of clarity. They contain too many options and too much flexibility. There are often many ways of doing the same or similar things. Kaufman has a calculation on the number of cryptographic negotiations in IKE: 1 MUST, 806,399 MAY [Kau02a]. The high system complexity relates to an extreme obscurity in the system specification. The obscurity is actually not a good thing: it may easily confuse expert reviewers and blind them from seeing security weaknesses, or may mislead implementors and cause them to code flawed implementations.

Ferguson and Schneier regard the high-degree system complexity as a typical “committee effect” [FS00]. They argue that “committees are notorious for adding features, options, and additional flexibility to satisfy various factions within the committee”. Indeed, if a committee effect, i.e., the additional system complexity, is seriously detrimental to a normal (functional) standard (as we sometimes experience), then it shall have a devastating effect on a security standard.

A serious problem with the high-degree flexibility and numerous options is not just an extreme difficulty for reviewers to understand the system behavior, nor just a ready possibility for implementors to code incorrect system, but that some specified options may themselves be dangerous. In §11.2.3.4, we have depicted an optional scenario for Malice to mount a perfect interleaving attack on IKE’s signature based Aggressive Mode, by choosing a signature scheme with message recovery property. Let us now see another example of such dangers.

The example of danger is manifested by an excerpt from an interpretation paper entitled “Understanding the IPSec Protocol Suite” [Alc00]. That paper, published in March 2000, provide explanations on IPSec and IKE at various levels, from a general concept for network security to some detailed features of IPSec and IKE. The following excerpt (page 6 of [Alc00]) explains an optional feature for “Authentication within the encapsulating security payload (ESP)” (an ESP is a cipher chunk which encrypts some confidential data):

The ESP authentication field, an optional field in the ESP, contains something called an integrity check value (ICV) — essentially a digital signature computed over the remaining part of the ESP (minus the authentication field itself). It varies in length depending on the authentication algorithm used. It may also be omitted entirely, if authentication

services are not selected for the ESP.

In this explanation, we can see an option to omit the entire data-integrity protection for a ciphertext. As we have seen in §10.7.8 and shall further see in a few later chapters that encryption without integrity (“authentication” in the excerpt) is generally dangerous, and most encryption algorithms cannot provide proper confidentiality protection without a proper data-integrity protection. Thus, a security problem in IPSec which Bellovin identified and criticized in 1996 (see the final paragraph of §11.2.3.5) remained to be explained as a feature four years later (the IPSec explanation paper was published in March 2000)! We believe that it is the high complexity of the IPSec specifications that contributes to the hiding of this dangerous error.

Aiello et al [ABB⁺02a] criticize IKE from its high complexities in computation and communication. They consider that protocols in IKE are vulnerable to denial of service attacks: Malice and his friends distributed over the Internet can just initiate numerous requests for connections, which include numerous stateful “cookies” for a server to maintain. They proposed a protocol named Just Fast Keying (JFK) and suggest that JFK be the successor of IKE. Blaze disclosed one reason why their protocol should be named JFK [Bla02]:

We decided this was an American-centric pun on the name Ike, which was the nickname of President Eisenhower, who had the slogan “I like Ike”. We don’t like IKE, so we’d like to see a successor to IKE. We call our protocol JFK, which we claim stands for “Just Fast Keying”, but is also the initials of a president who succeeded Eisenhower for some amount of time. We’re hoping not to ever discuss the protocol in Dallas. If there’s ever an IETF in Dallas again^e, we’re not going to mention our protocol at all there.

11.3 The Secure Shell (SSH) Remote Login Protocol

The Secure Shell (SSH) [Ylo95], [Ylo02c], [Ylo02d], [Ylo02a], [Ylo02b] is a public-key based authentication protocol suite which enables a user to securely login to a remote server host over an insecure network, to securely execute commands in the remote host, and to securely move files from one host to another.

11.3.1 The SSH Architecture

The SSH protocol runs between two untrusted computers over an insecure communications network. One is called the remote server (host), the other is called the client from which a user logs on to the server by using the SSH protocol.

^eThe 34th IETF was held in Dallas, Texas in December 1995.

The SSH protocol suite consists of three major components:

- The SSH Transport Layer Protocol [Ylo02d] provides server authentication to a client. This protocol is public-key based. The premise of (i.e., input to) this protocol for the server part is a public key pair called “host key” and for the client part is the public host key. The output from this protocol is a unilaterally authenticated secure channel (in terms of confidentiality and data integrity) *from the server to the client*. This protocol will typically be run over a TCP (Transport Control Protocol) and IP (Internet Protocol) connection, but might also be used on top of any other reliable data stream.
- The SSH User Authentication Protocol [Ylo02a]. This protocol runs over the unilateral authentication channel established by the SSH Transport Layer Protocol. It supports various unilateral authentication protocols to achieve entity authentication *from a client-side user to the server*. For this direction of authentication to be possible, the remote server must have a priori knowledge about the user’s cryptographic credential, i.e., the user must be a known one to the server. These protocols can be public-key based or password based. For example, it includes the simple password based authentication protocol Protocol 10.3. The output from an execution of a protocol in this suite, in conjunction with that from the SSH Transport Layer Protocol, is a mutually authenticated secure channel between the server and a given user in the client side.
- The SSH Connection Protocol [Ylo02b]. This protocol runs over the mutually authenticated secure channel established by above two protocols. It materializes an encrypted communication channel and tunnels it into several secure logical channels which can be used for a wide range of secure communication purposes. It follows standard methods for providing interactive shell sessions.

Clearly, the SSH Connection Protocol is not an authentication protocol and is outside the interest of this book, and the SSH User Authentication Protocol suite can be considered as a collection of applications of standard (unilateral) authentication protocols which we have introduced in Chapter 10 (however notice a point to be discussed in §11.3.4). Thus, we only need to introduce the SSH Transport Layer Protocol.

11.3.2 The SSH Transport Layer Protocol

In the new version of the SSH Protocol [Ylo02c], [Ylo02d], the SSH Transport Layer Protocol applies the Diffie-Hellman key exchange protocol and achieves unilateral authentication from the server to the client by the server signing its key exchange material.

11.3.2.1 Server host key

Each server host has a host key. A host may have multiple host keys using multiple different algorithms. If a server host has keys at all, it must have at least one key using each required public-key algorithm. The current Internet-Draft [Ylo02c] stipulates the default required public-key algorithm to be the DSS (Digital Signature Standard (9.3.4.3)). The default public-key algorithm for the current version in use (in the time of writing [Ylo95]) is the RSA signature (§9.3.1).

The server host key is used during key exchange: the server uses its private key to sign its key exchange material; the client uses the public key to verify that it is really talking to the correct server. For this to be possible, the client must have a priori knowledge of the server's public host key.

SSH supports two different trust models on the server's host key:

- The client has a local database that associates each server host name with the corresponding public part of the host key. This method requires no centrally administered infrastructure (called public-key infrastructure, to be introduced in Chapter 12), and hence no trusted third party's coordination. The downside is that the database for (server-name, host-key) association may become burdensome for the user to maintain. We shall exemplify a realistic method (§11.3.2.2) for a remote user to obtain an authenticated copy of the host (public) key.
- The (server-name, host-key) association is certified by some trusted certification authority (CA) using the technique to be introduced in Chapter 12. The client only needs to know the public key of the CA, and can verify the validity of all host keys certified by the CA.

The second alternative eases the key maintenance problem, since ideally only a single CA's public key needs to be securely stored on the client (security here means data integrity). On the other hand, each host key must be appropriately certified by a CA before authentication is possible. Also, a lot of trust is placed on the central infrastructure.

As there is no widely deployed public-key infrastructure (PKI, Chapter 12) available on the Internet yet, the first trust model, as an option, makes the protocol much more usable during the transition time until a PKI emerges, while still providing a much higher level of security than that offered by older solutions (such as the UNIX^a session commands: `rlogin`, `rsh`, etc).

11.3.2.2 Realistic methods for authenticating a server's public key

A workable method for a user to have an authenticated copy of the server's public key is for the user to bring with her/him a copy of the public key and put it in the

client machine before running the key exchange protocol. For example, when the user is travelling, (s)he can bring with her/him a floppy diskette which contains the server's public key. In the current working version of the SSH Protocol [Ylo95], the server's public key is in a file named `identity.pub`, it should be put in the `~/.ssh` directory of the client machine. Of course, the user should physically secure the public key (e.g., the floppy diskette) in terms of integrity while travelling.

Another realistic method for a user to have an authenticated copy of the server's public key downloaded from an insecure link is to use voice authentication over the telephone.

First, the host public key is downloaded by the user in the client machine via an insecure communication link. Then the user make a phone call to the site of the remote server, where a security administrator reads a hexadecimal "fingerprint" of the host public key to the user, i.e.,

$$\text{"fingerprint"}(\text{host key}) = H(\text{host key})$$

where H is an agreed cryptographic hash function, such as SHA-1. In the SHA-1 case, the whole "fingerprint" has 160 bits and can be read over the phone as 40 hexadecimal characters. Then the user can re-generate the "fingerprint" in the client machine using the downloaded version of the host key and check if the two copies of the "fingerprint" are identical. In this way, the user at the client side and the security administrator at the remote server side use their voices to authenticate the correctness of the host (public) key. In this case, we assume that the user and the security administrator recognize each other's voices.

These means are not secure in a strong sense, but are workable to a quite good degree. They are useful today when PKI is not ready over the Internet.

11.3.2.3 The key exchange protocol

An key exchange connection is always initiated by the client side. The server listens on a specific port waiting for connections. Many clients may connect to the same server machine.

The new version of the SSH Protocol [Ylo02c], [Ylo02d] applies Diffie-Hellman key exchange protocol (§8.2) to achieve session key agreement. In the description of the protocol we use the following notation:

- C : the client;
- S : the server;
- p : a large safe prime;
- g : a generator for a subgroup G_q of $GF(p)$;
- q : the order of the subgroup G_q ;

- V_C, V_S : C 's and S 's protocol versions, respectively;
- K_S : S 's public host key;
- I_C, I_S : C 's and S 's “Key Exchange Initial Message” which have been exchanged before this part begins.

The key exchange protocol is as follows:

1. C generates a random number x ($1 < x < q$) and computes

$$e \stackrel{\text{def}}{=} g^x \pmod{p};$$

C sends e to S ;

2. S generates a random number y ($0 < y < q$) and computes

$$f \stackrel{\text{def}}{=} g^y \pmod{p};$$

S receives e ; it computes

$$K \stackrel{\text{def}}{=} e^y \pmod{p},$$

$$H \stackrel{\text{def}}{=} \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel e \parallel f \parallel K),$$

$$s \stackrel{\text{def}}{=} \text{Sig}_S(H).$$

S sends $K_S \parallel f \parallel s$ to C ;

3. C verifies that K_S really is the host key for S (using any suitable methods, e.g. a certificate or a trusted local database or the method described in §11.3.2.2);
 C then computes

$$K \stackrel{\text{def}}{=} f^x \pmod{p},$$

$$H \stackrel{\text{def}}{=} \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel e \parallel f \parallel K),$$

and verifies the signature s on H ; C accepts the key exchange if the verification passes.

After the key exchange, the communications between the two parties will be encrypted using the agreed session key K . The two parties turn to execute the SSH User Authentication Protocol [Ylo02a] which may be any one of known unilateral authentication technique. After that, the user on the client can request a service using the SSH Connection Protocol [Ylo02b].

11.3.3 The SSH Strategy

One of the goals of the SSH Protocol is to improve security on the Internet in a progressive manner. The permission for the client to use “any suitable methods” (e.g., those given in §11.3.2.2) to verify authenticity of the server’s public key clearly demonstrates SSH’s strategy of quick deployment and supporting backward compatibility.

At the stage way before a public-key infrastructure is ready over the Internet, the improved security from SSH needn’t be a very strong one, but is much stronger and than without. The ease-of-use and quick-to-deploy solution is a great value of SSH and is the reason why the technique has been popularly implemented and widely in use in the UNIX^a and Linux platforms.

From this real-world application of authentication techniques we also see that public-key cryptography forms a vital enabler for the easy solution. The server’s host key in the untrusted environment (e.g., in the client or in the route from the server to the client) only exists in public key form, and so the management of this important key material becomes trivially easy. The problem will become immensely complicated if the protocol is secret key based.

11.3.4 A Caveat

Finally, we should point out a caveat for a user to handle with care of her/his cryptographic credential which is used by the SSH User Authentication Protocol. This credential, which can be public-key based or password based, will be used by the protocol part running on the client machine which is considered part of the untrusted environment. In the current working version of the SSH Protocol [Ylo95], public-key based user credential (i.e., the private key matching the user’s public key) is encrypted under the user’s password and the ciphertext is stored in a file named `identity` in the directory `~/.ssh` of the client machine. The file is read in the protocol execution time by the client part of the protocol which prompts the user to input password. Naturally, the user should make sure that the protocol part running on the client machine is a genuine one. The user should also delete the encrypted private key file from the client machine after use.

11.4 The Kerberos Protocol and its Realization in Windows 2000

Let Alice be an employee of a multi-nation company. She may be served with various kinds of information resources and services. For example, from her “home server”, Alice gets the usual computer network services (i.e., World Wide Web, email, etc); on a “project server”, Alice and her team members will be the exclusive users and the owner of the data related to their work; on an “human-resource server”, Alice may manage her HR related issues, e.g., managing how much percentage of her

next month salary should be invested for company share purchase; if Alice is a manager, she may need to update her subordinates' performance review records on an HR database; from an "intellectual-property server", Alice (as an inventor) may be working on her current patent filing; on an "expenses server", Alice shall often make expenses claims after her business trips. It is not difficult to imagine more examples of services.

In an enterprise environment, a user (an employee or a customer) is usually entitled to using enterprise-wide distributed information services. These services are usually maintained by various business units in the enterprise. As a result, the various information servers can operate in different geographical locations (even around the globe). Speaking in terms of network organization, these servers are in different **network domains**. For secure use of these services (all examples we have listed in the previous paragraph involve seriously sensitive information), a user needs various credentials for her/him to be authenticated before a service can be granted. However, it would be unrealistic and uneconomic to demand a user to maintain several different cryptographic credentials, whether in terms of memorizing various passwords, or in terms of holding a number of smartcards.

A suitable network authentication solution for this environment is the Kerberos Authentication Protocol [MNSS87], [JC93]. The basic idea is to use a trusted third party to introduce a user to a service by issuing shared session key between the user and the server. This idea is due to Needham and Schroeder [NS78] and is illustrated in the Needham-Schroeder Authentication Protocol (Protocol 2.4). As the original Needham-Schroeder protocol is flawed (see §2.5.4.2), Kerberos uses essentially a timestamp version of the Needham-Schroeder protocol.

Now consider that Alice in Protocol 2.4 is in the position of a user who shares a long-term secret key with a trusted third party (Trent in that protocol). Also consider that Bob in that protocol is in the position of a server who also shares a long-term secret key with the trusted third party. When Alice wants to use Bob's service, she can initiate a protocol run with Trent and ask from Trent for a cryptographic credential good for accessing Bob's service. Trent can provide a ("ticket granting") service by issuing a session key to be shared between Alice and Bob, and securely delivers the session key inside two "tickets" encrypted under the long-term secret keys which Trent shares with Alice and with Bob, respectively. That's the idea.

Windows 2000, an important operating system now widely in use in an enterprise network environment, uses the Kerberos Authentication Protocol (based on Version 5 [JC93]), as its network authentication basis.

Kerberos is created by Project Athena at the Massachusetts Institute of Technology (MIT) as a solution to network security problems. MIT has developed the Kerberos Version 5 as a free software (with source code available) which can be downloaded from MIT's Web site <<http://web.mit.edu/kerberos/www/>>. However, due to the exportation control on cryptographic products regulated by the

government of the United States of America, at the time of writing, this distribution of Kerberos executables is only available to the citizens of the USA located in the USA, or to Canadian citizens located in Canada.

The Kerberos Protocol Version 5 is slightly more complex than the Needham-Schroeder Authentication Protocol (the timestamp-fixed version). Let us now introduce Kerberos Protocol Version 5.

11.4.1 A Single-sign-on Architecture

The Kerberos Authentication Protocol consists of a suite of three sub-protocols called exchanges^f. These three exchanges are:

1. the Authentication Service Exchange (AS Exchange): it runs between a “client” *C* and an “authentication server” *AS*;
2. the Ticket-Granting Service Exchange (TGS Exchange): it runs between *C* and a “ticket granting server” *TGS* after the AS Exchange;
3. the Client/Server Authentication Application Exchange (AP Exchange): it runs between *C* and an “application server” *S* after the TGS Exchange;

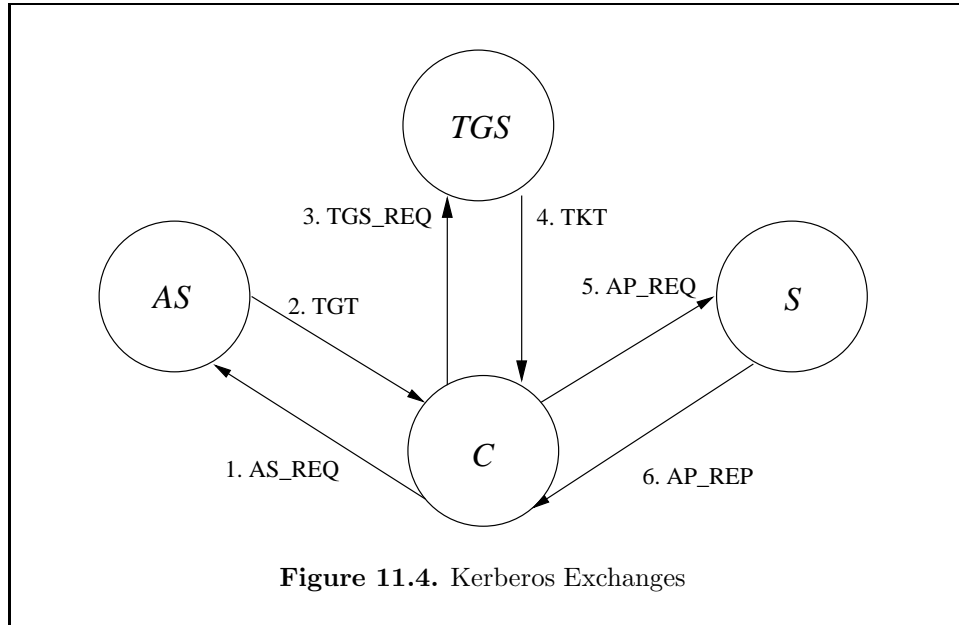
Each of these three exchanges is a two-message exchange protocol. These exchanges has the sequential dependent relation listed above which can be illustrated as a three-headed creature^g in Figure 11.4.

Kerberos has five principals who operate in these three exchanges and these principals have the following roles:

- *U*: a User (a human being) whose actions in the protocols are always performed by her/his client process; so *U* only appears in the protocols as a message. Each user memorizes a password as her/his **single-sign-on** credential for using the Kerberos system.
- *C*: a Client (a process) which makes use of a network service on behalf of a user. In an AS Exchange, in which *C* is initiated by *U*, *C* will need *U*’s Kerberos system credential. This user credential is given to *C* as it prompting *U* to key-in her/his password.
- *S*: an application Server (a process) which provides an application resource to a network client *C*. In an AP Exchange, it receives an “application request”

^fThe suite contains a much bigger number of auxiliary sub-protocols for various specialized tasks, such as password changing, ticket renewal, error handling, etc., however, we shall only describe the three main protocols which provide authentication functions.

^gThe name Kerberos comes from Greek mythology; it is the three-headed dog that guarded the entrance to Hades.



(AP_REQ) from *C*. It responds with “application reply” (AP_REP) which may entitle *C* an application service.

An AP_REQ contains *C*’s credential called a “ticket” (TKT) which in turn contains an application session key $K_{C,S}$ temporarily shared between *C* and *S*.

- **KDC:** Key Distribution Center. KDC is a collective name for the following two authentication servers:

- **AS:** an Authentication Server. In an AS Exchange, it receives a plaintext “authentication service request” (AS_REQ) from a client *C*. It responds with a “ticket granting ticket” (TGT) which can later be used by *C* in a subsequent TGS Exchange.

Initially, *AS* shares a password with each user it serves. A shared password is set up via a single-sign-on means outside the Kerberos system.

A TGT supplied to a client *C* as the result of an AS Exchange has two parts. One part is for *C* to use and is encrypted under a key derived from a user’s single-sign-on password. The other part is for a “Ticket Granting Server” (to be described in the *TGS* item below) to use and is encrypted under a long-term key shared between *AS* and the latter. Both parts of a TGT contain a ticket session key $K_{C,TGS}$ to be shared between *C* and a “Ticket Granting Server”.

- *TGS*: a Ticket Granting Server. In an TGS Exchange it receives an “ticket granting request” (TGS_REQ) (which contains a “ticket-granting ticket” TGT) from a client *C*. It responds with a “ticket” (TKT) which entitles *C* to use in a subsequent AP Exchange with an application server *S*.

Similar to a TGT, a TKT has two parts. One part is for a client *C* to use and is encrypted under a ticket session key $K_{C,TGS}$ (which has been distributed to *C* and *TGS* in TGT). The other part is for an application server *S* to use and is encrypted under key $K_{S,TGS}$ which is a long-term key shared between *S* and *TGS*.

Both parts of a TKT contain a new application session key $K_{C,S}$ to be shared between *C* and *S*. The application session key is the cryptographic credential for *C* to run a subsequent AP Exchange with *S* to get an application service from *S*.

11.4.1.1 Why is KDC divided into two sub-servers *AS* and *TGS*?

We shall see in a moment that the roles of *AS* and *TGS* are actually very similar: both are collectively referred to as a key distribution center (KDC).

The reason to divide KDC into two similar roles is the consideration that the system may be used in a very large network “realm” in which application servers belonging to different network domains should be organized as subordinators of different *TGS*’s in different domains. Therefore, even though a fixed user *U* only has a fixed single-sign-on *AS*, (s)he can be served by a plural number of *TGS*’s and consequently by even a larger number of application servers.

11.4.2 The Kerberos Exchanges

Now let us describe each of the three Kerberos exchanges. For ease of exposition of the main idea in the Kerberos Authentication Protocol, we shall only present mandatory protocol messages. For the full description of all protocol message details which include an enormous volume of optional messages, the reader should study [JC93].

11.4.2.1 The Authentication Service Exchange

The AS Exchange concerns only *C* and *AS*:

1. AS_REQ $C \rightarrow AS : U, TGS, \text{Life_time1}, N_1$
2. TGT $AS \rightarrow C : U, T_{C,TGS}, TGT_C$

where

$$T_{C,TGS} = \{U, C, TGS, K_{C,TGS}, \text{Time_start}, \text{Time_expire}\}_{K_{AS,TGS}},$$

$$TGT_C = \{TGS, K_{C,TGS}, \text{Time_start}, \text{Time_expire}, N_1\}_{K_U}.$$

Message 1 is invoked by the user U . The client C informs the authentication server AS using the plaintext `AS_REQ` messages that it wishes to communicate on behalf of the user U with the ticket granting server TGS . A lifetime `Life_time1` (bookkeeping information) and a nonce N_1 (freshness identifier) are also included in the request.

In response, the authentication server AS generates a new ticket session key $K_{C,TGS}$ for sharing between C and TGS ; it then encrypts the ticket session key inside a ticket granting ticket `TGT` and sends it back to C as message 2.

The part of `TGT` for TGS is $T_{C,TGS}$ and is encrypted using the long-term key $K_{AS,TGS}$ shared between itself and TGS , the part of `TGT` for C is T_C and is encrypted the user's password K_U .

Upon receipt of message 2, C can decrypt T_C (it has prompted U for inputting the password K_U). If everything passes validation (be careful about the validation, to discuss in §11.4.3), then C accepts the ticket session key $K_{C,TGS}$ and the ticket $T_{C,TGS}$. C now has a valid “ticket granting ticket” for use with TGS .

A caveat on proper decryption of a Kerberos ticket will be discussed in §11.4.3.

11.4.2.2 The Ticket-granting Service Exchange

The TGS Exchange has a format similar to that of the AS Exchange, except that the client's request message, `TGS_REQ`, now contains an **authenticator** trailing after the plaintext request message.

3. `TGS_REQ` $C \rightarrow TGS : S, \text{Life_time2}, N_2, T_{C,TGS}, A_{C,TGS}$
4. `TKT` $TGS \rightarrow C : U, T_{C,S}, TKT_C$

where

$$T_{C,S} = \{U, C, S, K_{C,S}, \text{Time_start}, \text{Time_expire}\}_{K_{S,TGS}},$$

$$TKT_C = \{S, K_{C,S}, \text{Time_start}, \text{Time_expire}, N_2\}_{K_{C,TGS}},$$

$$A_{C,TGS} = \{C, \text{Client_time}\}_{K_{C,TGS}}.$$

The functionalities of this pair of exchange and actions of principals can be explained analogously to those for the AS Exchange. The only additional item which worths explanation is $A_{C,TGS}$. This is an authenticator. The use of an authenticator is to show the ticket granting server TGS that the client C has used

the ticket session key $K_{C,TGS}$ in `Client_time`. *TGS* should check its local host time to confirm that the difference between `Client_time` and its local time is within an allowable range.

A caveat on a Kerberos authenticator is discussed in §11.4.3.

11.4.2.3 The Application Service Exchange

Finally, in the AP Exchange a client *C* uses the newly obtained application session key $K_{C,S}$ and the ticket $T_{C,S}$ to obtain an application service from an application server *S*.

5. AP_REQ $C \rightarrow S : T_{C,S}, A_{C,S}$

6. AP_REP $S \rightarrow C : A_{S,C}$

where

$$A_{C,S} = \{C, \text{Client_time1}\}_{K_{C,S}},$$

$$A_{S,C} = \{\text{Client_time1}\}_{K_{C,S}}.$$

The meaning this pair of exchange is straightforward.

As we have warned in the descriptions of the previous two exchanges, we shall pay attention to the caveats below.

11.4.3 Caveats

We must discuss two caveats in Kerberos exchanges.

The first one is about careful validation of a Kerberos ciphertext in a decryption time.

When a principal decrypts a ticket, it should validate the decryption. From the structure of a Kerberos ticket, the validation obviously include steps for checking the freshness identifiers and the correctness of the intended identities. However, what is not so obvious is the need of verifying data-integrity of a ciphertext. The importance of the data-integrity verification has been illustrated by several examples in the previous chapter (e.g., §10.7.8), and will be further investigated in §15.2.1.

This caveat applies to all encryption in Kerberos exchanges.

The second caveat is about “authenticator”.

Although the name “authenticator” and its position and usage (trailing a ticket) may suggest that it plays the role of a message authentication code (MAC, see §9.2) for providing a data-integrity protection on the ticket it trails (e.g., $A_{C,TGS}$ with respect to $T_{C,TGS}$), this imagined “protection” is actually absent.

Not only the needed integrity protection on the ticket must be supplied by a proper mechanism (e.g., by a MAC), but also notice: using encryption to create an authenticator is using a wrong cryptographic service. In order to prevent an adversary to modify a `Client_time` in an authenticator, the cipher block of an authenticator itself needs a data-integrity protection!

This caveat applies to all authenticators in Kerberos.

11.5 SSL and TLS

An important authentication protocol, mainly for World Wide Web (Web for short) security, is the Secure Sockets Layer Protocol (SSL) [Hic95], [FKK96]. The term "sockets" refers to standard communication channels linking peer processes on network devices (e.g., on client/server machines). A sockets-layer protocol runs under the application-layer protocols such as the Hypertext Transfer Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), and above the network-layer protocols such as Transport Control Protocol (TCP) and Internet Protocol (IP). When the sockets-layer communications are secured (e.g., in confidentiality and data integrity), communications in all application-layer protocols will be secured in the same manner.

SSL is a commonly-used protocol for managing the security of a message transmission on the Internet. The protocol is originally developed by Netscape Communications Corporation as an integral part of its Web browser (client side software) and Web server. It is later accepted by Microsoft and other Internet client/server developers as well, and evolves into the de facto standard for Web security until it further evolves into the Transport Layer Security (TLS) [DA99]. The latter is an Internet standard for Web security developed by the industrial standardization body Internet Engineering Task Force (IETF).

TLS is based on SSL and is not drastically different from SSL. However, since TLS succeeds SSL as Internet standard for Web security, we shall from now on follow the standard track and only use the term TLS in our description of the Web security protocol.

11.5.1 TLS Architecture Overview

TLS is composed of two layered protocols: the **TLS Record Protocol** and the **TLS Handshake Protocol**. The latter is on top of the former.

The TLS Record Protocol provides secure encapsulation of the communication channel for use by higher layer application protocols. This protocol runs on top of the TCP and IP layers and provides a reliable session connection. It takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC (HMAC, see §9.2.2) for data-integrity, encrypts

(symmetric algorithm) for confidentiality, and transmits the result to the peer communicant. At the receiving end, it receives cipher data blocks, decrypts them, verifies the MAC, optionally decompressed, reassembles the blocks and delivers the result to higher level application processes.

The keys for symmetric encryption and for HMAC are generated uniquely for each session connection and are based on a secret negotiated by the TLS Handshake Protocol.

The TLS Handshake Protocol allows the server and client to authenticate each other, negotiate cryptographic algorithms, agree on cryptographic keys and thereby establish a secure session connection for the TLS Record Protocol to process secure communications for higher level application protocols.

From this TLS architecture description it is clear that the TLS Record Protocol is not an authentication protocol, albeit it is a protocol for achieving secure communications. We therefore should only introduce the TLS Handshake Protocol.

11.5.2 TLS Handshake Protocol

The TLS Handshake Protocol can be considered as a stateful process running on the client and server machines. A stateful connection is called a “session” in which the communication peers perform the following steps:

- They exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- They exchange the necessary cryptographic parameters to allow the client and server to agree on a secret (called “master secret”).
- They exchange certificates and cryptographic information to allow the client and server to authenticate themselves to one another.
- They generate session secrets from the master secret by exchanging random values.
- They verify that their peer has calculated the same security parameters to confirm that the handshake has completed without having been tampered by an attacker.
- The established secure channel is passed on to the TLS Record Protocol for processing higher level application communications.

These steps are realized by four message exchanges which we describe below. In order to achieve a better exposition of the protocol idea we shall only describe a simplified version of the TLS Handshake Protocol by omitting some optional elements. In the protocol description, *C* denotes the client (i.e., the client side Web

browser), S denotes the Web server. If a message is trailed with *, this message is optional.

1. $C \rightarrow S$: ClientHello;
2. $S \rightarrow C$: ServerHello,
ServerCertificate*,
ServerKeyExchange*,
CertificateRequest*,
ServerHelloDone;
3. $C \rightarrow S$: ClientCertificate*,
ClientKeyExchange,
CertificateVerify*,
ClientFinished;
4. $S \rightarrow C$: ServerFinished.

This protocol can be executed with all the optional messages and the ClientKeyExchange message omitted. This is the case when the client wants to resume an existing session.

Now let us provide an overview level explain on the messages exchanged in the TLS Handshake Protocol.

11.5.2.1 Hello message exchange

The client starts session connection by sending a ClientHello message to which the server must respond with a ServerHello message, or else the connection will fail. These two messages establish the following fields: “protocol_version”, “random”, “session_id”, “cipher_suites”, and “compression_methods”.

The field “protocol_version” is for backward compatibility use: the server and client may use this field to inform their peer the version of the protocol it is using.

The field “random” contains random numbers (nonces as freshness identifiers) which are generated by the both sides and are exchanged. It also contains the local time of the each communicant.

The field “session_id” identifies the current session connection. When the client wishes to start a new session connection, ClientHello.session_id should be empty. In this case, the server generates a new session_id, uses this new value in the field ServerHello.session_id, and caches the session_id in its local memory. If ClientHello.session_id is non-empty (when the client wants to resume an existing session), the server should try to find the session_id from its local cache, and resume the identified session.

A point of noticing is the field “cipher_suites”. `ClientHello.cipher_suites` is a list of the cryptographic options supported in the client side machine, sorted with the client’s first preference first. A wide range of public-key and symmetric cryptographic algorithms, digital signature schemes, MAC schemes and hash functions can be proposed by the client. The server selects a single scheme for each necessary cryptographic operation, and informs the client in `ServerHello.cipher_suites`.

11.5.2.2 Server’s certificate and key-exchange material

Following the hello message exchange, the server may optionally send its certificate, if it is to be authenticated. The `ServerCertificate` message, if non-empty, is a list of X.509.v3 certificates (see § 12.2). An X.509 certificate contains sufficient information about the name and the public key of the certificate owner and that about the issuing certification authority (see Example 12.1). Sending a list of certificates permits the client to choose one with the public key algorithm supported at the client’s machine.

Following `ServerCertificate` is `ServerKeyExchange`. It contains the server’s public key material matching the certificate list in `ServerCertificate`. The material for Diffie-Hellman key agreement will be included here which is the tuple (p, g, g^y) where p is a prime modulus, g is a generator modulo p of a large group and y is an integer cached in the server’s local memory (linked to “session_id”).

The server who provides non-anonymous services may further request a certificate from the client using the `CertificateRequest` message, if that is appropriate to its selection of the public-key algorithm from `ClientHello.cipher_suite`.

Now the server will send the `ServerHelloDone` message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response.

11.5.2.3 Client response

If the server has sent the `CertificateRequest` message, the client must send either the `ClientCertificate` message or the `NoCertificate` alert.

The `ClientKeyExchange` message is now sent. The content of this message will depend on the public key algorithm agreed between the `ClientHello` and `ServerHello` messages.

In the case of the client’s `KeyExchangeAlgorithm` being RSA, the client generates a “master_secret” (a 48-byte number) and encrypts it under the server’s certified RSA public key (obtained from the `ServerCertificate`).

If the client has sent a certificate with signing ability, a digitally-signed `CertificateVerify` message will be sent for the server to explicitly verify the client’s certificate.

11.5.2.4 Finished message exchange

The client now sends the ClientFinished message which includes a keyed HMAC (keyed under the “master_secret”) to allow the server to confirm the proper handshake executed at the client side.

In response, the server will send its own ServerFinished message which also includes a keyed HMAC to allow the client to confirm the proper handshake executed at the server side.

At this point, the handshake is complete and the client and server may begin to exchange application layer data.

11.5.3 A Typical Run of the TLS Handshake Protocol

Let us complete our description of the TLS Protocol by exemplifying a typical run of the Handshake Protocol. The execution example is illustrated in Example 11.2.

Example 11.2: A Typical Run of the TLS Handshake Protocol.

1. $C \rightarrow S$: ClientHello.protocol_version = “TLS Version 1.0”,
 ClientHello.random = T_C, N_C ,
 ClientHello.session_id = “NULL”,
 ClientHello.crypto_suite = “RSA: encryption, SHA-1: HMAC”,
 ClientHello.compression_method = “NULL”;
2. $S \rightarrow C$: ServerHello.protocol_version = “TLS Version 1.0”,
 ServerHello.random = T_S, N_S ,
 ServerHello.session_id = “xyz123”,
 ServerHello.crypto_suite = “RSA: encryption, SHA-1: HMAC”,
 ServerHello.compression_method = “NULL”,
 ServerCertificate = point_to(server’s certificate),
 ServerHelloDone;
3. $C \rightarrow S$: ClientKeyExchange = point_to(RSA_Encryption(master_secret)),
 ClientFinished = SHA-1(master_secret || C || N_C, N_S, \dots);
4. $S \rightarrow C$: ServerFinished = SHA-1(master_secret || S || N_S, N_C, \dots).

Figure 11.5.

In this execution of the TLS Handshake Protocol, the client chooses to be anonymous and so is not authenticated to the server, the client chooses to use RSA for

encryption, SHA-1 for computing HMACs. As a result, the server unilaterally authenticates itself to the client. The output from the execution is a unilaterally authenticated channel from the server to the client.

This execution shows a typical example of using the TLS Protocol in a Web-based electronic commerce application, for example, buying a book from an on-line book seller. The output channel assures the client that only the authenticated server will receive its instructions on book purchase which may include confidential information such as its user's bankcard details, the book title, and the delivery address.

11.6 Chapter Summary

In this chapter we have introduced four authentication protocols for real world applications. Although in our description of each protocol (suite) we have taken a great deal of simplification, still, our descriptions show enough engineering complexities. These complexities are due to real-world necessities such as algorithm and parameter negotiation, compatibility for use by a wide range of systems, backward compatibility, ease of use, etc. In the case IPsec and IKE, the need of supporting a general quality of confidentiality also contributed to the high system complexity. From our study in this chapter we know that for any real-world application of authentication protocols, we are not only facing a number of security problems, but also facing a great deal of system engineering problems.

We have also seen that the extreme error-prone nature of authentication protocols inevitably appears in the versions for real world applications. For this reason, we have still not completed our topic on authentication protocols. We will return to this important topic in Chapter 15 on formal analysis techniques.

AUTHENTICATION FRAMEWORK FOR PUBLIC-KEY CRYPTOGRAPHY

12.1 Introduction

In the usual sense of public-key cryptography, a key generation procedure invariantly contains the following step

$$\text{public-key-component} = F(\text{private-key-component}). \quad (12.1.1)$$

Here, $F()$ is some efficient, 1-1 and one-way function; **public-key-component** and **private-key-component** stand for some part of a public and some part of a private key. It is always the case that **public-key-component** be random since one should choose a private key at uniformly random and since $F()$ with the listed properties won't be able to map a uniformly random **private-key-component** to **public-key-component** with some controllable distribution.

With every public key containing a random part, it is obviously necessary that a principal's public key be associated with the principal's identity information in a verifiable and trustworthy way. Clearly, to send a confidential message encrypted under a public key, the sender must make sure that the random-looking public key used really belongs to the intended recipient. Likewise, to establish the origin of a message using a digital signature scheme, the verifier must make sure that the public key used for the signature verification really belongs to the claimed signer.

In general, to use public-key cryptography in real-world applications, we need a mechanism which enables a ready verification of the association between a public key and a principal's identity. Such a mechanism is usually realized in an authentication

framework: it enables the owner of a public key to authenticate toward the system.

12.1.1 Chapter Outline

In the rest of this chapter we will introduce two different ways for establishing authentication framework for public-key cryptography: one is called **public key certification infrastructure (PKI)** (§12.2), and the other, **identity based public-key cryptography** (§12.3).

12.2 Directory-Based Authentication Framework

For a pair of principals who communicate frequently, it need not be difficult for them to securely identify the other party's public key: they can exchange their public keys initially in a physically secure manner, e.g., in a face-to-face meeting, and then store the keys by a secure means. However, this “simple” **key-management** method does not scale up well. In the general setting for an open communications system, communications take place between principals who may never meet before; also in most cases a communication may take place between a pair of principals *once* only. The “simple” key-management method will require each principal to manage an unrealistically huge number of public keys. Moreover, such a method does not really make use of the advantages of public-key cryptography.

In §2.3 we have seen an on-line service offered by a trusted principal for the management of secret keys. The service is a combination of sub-services such as key registration, authentication and name-directory. To use the key-management service, every principal should first establish a one-one and long-term relationship with a trusted server principal (authentication server) by sharing a long-term secret key with the latter. When two (end-user) principals need to conduct a secure communication between them, they can engage in an authentication protocol run together with the authentication server to establish a secure communication channel between them. Thus, each end-user principal only need to manage a single secret key shared with the authentication server. The key-management service introduced in Chapter 2 is for authentication protocols based on secret-key cryptosystems (even though in §2.5.6 we discussed the Needham-Schroeder public-key authentication protocol, the authentication service in that protocol is an on-line one, and hence is still in a secret-key fashion).

The secret-key management service can naturally be extended to the management of public keys. Here the key-management service is called **public-key certification** service, and a trusted server is called a **certification authority (CA)**. A CA is a special principal who is well-known and trusted directly by the principals in the domain it serves, and can also be known and trusted in a bigger domain through an indirect way (we shall discuss more about “trust” in a moment). For each end-user within the domain of a CA, the CA will issue a public-key **certificate**

for certifying the user's public key material. A public-key certificate is a structured data record with a number of data entries which include a uniquely identifiable identity of the holder and her/his public key parameter. A certificate is digitally signed by the issuing CA. Thus the CA's signature on a certificate provides a cryptographic binding between the holder's identity and her/his public key. A principal, after having verified the certificate of another principal, should believe the validity of the binding if she/he trusts the CA in that the CA has issued the certificate only after having properly identified the holder. In this way, the verification principal establishes a secure **key channel** which is directed from the certified public key toward her/him (in fact, toward the system). Kohnfelder first uses the name "public key certificate" [Koh78].

A public-key channel based on a certification service is often called a directory-based channel, as we have illustrated in Figures 7.1 and ???. The certification service is thus also often called a directory service.

Notice that, in comparison with the "trust" required by an authentication server for secret-key based authentication protocols (see §2.3), the "trust" required by a CA is much weaker. Here, the security service provided is message authentication, which can be provided without need of handling any secret (since verifying a CA's signature on a certificate only involves using the CA's public key). Without the need of handling any secret, the service can be provided off-line, that is, a CA need not be engaged in a protocol run with the end-user principals. An important feature of an off-line service is that it can scale up to deal with a very large system. Obviously, a CA's public key used for verifying the certificates that the CA has issued can itself, in turn, be certified by another CA, and so on.

The data entries in a certificate should include the identity information and the public key information of the issuing CA. They should also include some additional information, such as the description on the algorithm to be used for verifying the issuing CA's signature and that to be used by the public key certified, the valid period, condition of the use, etc. Semi-formally, a public-key certificate may be defined as follows:

Example 12.1: (Public Key Certificate)

```
certificate ::=
{
  issuer name;
  issuer information;
  subject name;
  subject information;
  validity period;
}
issuer information ::=
{
```

```
        issuer public key;
        signature algorithm identifier;
        hash function identifier
    }
    subject information ::=
    {
        subject public key;
        public key algorithm identifier
    }
    validity period ::=
    {
        start date;
        finish date
    }
```

□

12.2.1 Certificate Issuance

In the issuance of a certificate, a CA should validate the identity of a principal who requests for a certificate. The validation should of course involve some physical (i.e., non-cryptographic) means of identification, as we usually have to conduct in some business interaction (e.g., in the opening of a bank account). The principal should also prove that she/he knows the private component of the public key to be certified. The proof can either be in the form of user creating a signature on a challenge message, which is verifiable using the public key, or be in the form of a zero-knowledge proof protocol between the user and the CA, with the public key as the common input. Some applications requires the private component of a public key to have certain structure. In such applications, a zero-knowledge protocol can be designed to enable a proof of the needed structure. We shall see later chapter a few zero-knowledge protocols for proof of the structure of a secret.

12.2.2 Certificate Revocation

Occasionally, it may be necessary to revoke a certificate. Compromise of user's private key or change of user information are two examples for this need.

In the case of the directory based certification framework, the root CA should maintain a hot list of the revoked certificates. The hot list may be available on-line. Alternatively, the root CA may issue a “ Δ -revocation list” throughout the system, which only contains newly revoked certificates. The system-wise users can update their local copies of the certificate revocation list whenever they receive a Δ -revocation list.

A revocation of a certificate should be timestamped by the revocation CA. Signatures of a principal issued prior to the date of her/his certificate's revocation should be considered to be still valid (according to application) even if the date of the signature verification is later than the date of the certificate's revocation.

12.2.3 Examples of Public-key Authentication Framework

Now let us see several examples of directory based public-key authentication framework.

12.2.3.1 X.509 public-key certification framework

The standard public-key certification framework, called the X.509 [IT93] certification infrastructure, scales up in a tree hierarchy, called a **directory information tree (DIT)**. In such a tree hierarchy, each node represents a principal whose public-key certificate is issued by its immediate parent node. The leaf nodes are end-user principals. The non-leaf nodes are CAs at various levels and domains; for example, a country level CA has industry, education and government organization domains; each of these domains has many sub-domains, e.g, the education domain has various university sub-domains. The root node is called the **root CA** who is a well-known principal in the whole system. The root CA should certify its own public key. Since each CA is potentially capable of serving a large domain (of CAs or end-users), the depths of a DIT need not be a large number. Two end-user principals can establish a secure communication channel by finding upward in the DIT a CA who is the nearest common ancestor node of them.

12.2.3.2 PGP “web of trust”

Another public-key certification framework which has a large number of amateur users is called a PGP “web of trust” or “key-ring” (PGP stands for “Pretty Good Privacy” which is a secure email software developed by Zimmermann [Zim95]). This authentication model scales up in an un-hierarchical manner. In the PGP “web of trust”, any individual can be a “CA” for any other principals in the system by signing their “key certificates” which is simply a pair $\langle \text{name}, \text{key} \rangle$. Evidently, the signing relationship forms a web structure. Any single “CA” in the web is not well trusted or not trusted at all. The theory is that with enough such signatures, the association $\langle \text{name}, \text{key} \rangle$ could be trusted because not all of these signers would be corrupt. Thus, when Alice wants to establish the authenticity of Bob's key, she should request to see a number of Bob's “key certificates”. If some of the issuing “CA”s of these certificates are “known” by Alice “to some extent”, then she gains a certain level of authenticity about Bob's public key. Alice can demand Bob to provide more “certificates” until she is satisfied of the level of the trust.

12.2.3.3 Simple public key infrastructure (SPKI)

The X.509 public-key certification framework can be viewed as a global on-line telephone book. Each individual user occupies an entry in it and therefore the entry **subject name** in each user's certificate (see Example 12.1) must be a globally distinguished name. Such an authentication framework seems to be adequate for the early years of applications of public-key cryptography: secure communications in terms of confidentiality (i.e., against eavesdropping): the recipient of a confidential message should be uniquely identified together with her/his key.

Since 1990's, applications of public key became much wider to include electronic commerce, remote access and actions (see a list of applications in the Preface). Ellison et al considered that for the newly emerged applications, a globally distinguished name with a key bound to it becomes inadequate [EFL⁺99]. What an application needs to do, when given a public key certificate, is to answer the question of whether the remote key holder is permitted some access, or some authorized action. That application must make a decision. The data needed for that decision is almost never the spelling of a key holder's name. Instead, the application needs to know if the key holder is authorized for some access. This should be the primary job of a public-key certificate.

For this purpose, Ellison et al proposed a directory-based public-key certification framework named SPKI (which stands for "Simple Public Key Infrastructure") [EFL⁺99]. It is also a tree-structured framework, similar to an X.509 key certification framework. However, it contains "authorization" and "delegation" entries which carry authorization and delegation information. A piece of authorization information can be an authorization description which is bound to a public key. Thus, a certificate can directly show to an application whether or not the requester is authorized to perform an action. The delegation information describes the requester's power to delegate authorizations to another person. We may say that SPKI extends X.509 authentication framework to one with authorization and delegation features.

PolicyMaker [BFL96] and SDSI (which stands for "A Simple Distributed Security Infrastructure") [RL96] are other two proposals which consider authorization and policy issues in a authentication framework. PolicyMaker features the descriptions of certificate holder's role and the role-based policy. SDSI features localization naming rules. These features also aim to make a de-centralized authentication and authorization framework.

12.3 Non-Directory Based Authentication Framework

The key generation procedure in (12.1.1) in the usual sense of public-key cryptography renders all public keys random. Consequently, it is necessary to associate a public key to the identity information of its owner in an authentic manner. We have seen that such an association can be realized by a public-key authentication

framework: the tree-hierarchy public-key certification infrastructure (e.g., X.509 certification framework, see §12.2.3). However, to establish and to maintain a tree-hierarchy PKI incur a non-trivial level of system complexity and cost. It has long been desired that the standard public-key authentication framework be simplified.

It is reasonable to think that, if public keys are not random-looking then the system complexity and the cost for establishing and maintaining the public-key authentication framework may be reduced. Imagine, if a public key of a principal is self-evidently associated with the principal's identity information such as name, affiliation information plus electronic and postal mail addresses, then in essence there is no need to authenticate a public key. Indeed, our postal mail systems work properly this way.

Shamir pioneered a public-key cryptosystem in an unusual sense [Sha85]. It enables a great-deal of reduction in the system complexity for key authentication: in essence to one similar to that of a postal mail system. In his unusual public-key cryptosystem, the key generation procedure has the following step

$$\text{private-key} = F(\text{master-key}, \text{public-key}). \quad (12.3.1)$$

This key generation step takes the opposite direction to the key generation step for the usual sense of public-key cryptosystems, see (12.1.1). Of course, in order for a so computed **private-key** to be kept secret, the computation must not be public: it is restricted to a privileged principal (a trusted authority, TA). TA possesses exclusively the secret key **master-key** in order to be able to perform the computation in (12.3.1). Now that **public-key** is an input to the key generation procedure, any bit-string can be **public-key**! Since using identity information as a public key can greatly reduce the complexity of public-key authentication, Shamir suggested that the public keys in his novel public-key cryptosystem be chosen as users' identities and thus he named his scheme **identity-based public-key cryptography**.

It is obvious that the key generation procedure in (12.3.1) is a service offered by TA to a system wide users. The service is essentially an authentication one in nature: the private key that TA creates for a principal in connection to her/his ID as public key provides the principal with a credential for her/his ID-based public key to be recognized and used by other users in the system. Before creating a private key for a principal, TA should conduct a thorough checking on the identity information of the principal. This checking should include some physical (i.e., non-cryptographic) means of identification. Also, TA needs to be satisfied that the identity information supplied by the principal can uniquely pinpoint the principal. A similar identification checking is necessary before a CA issues a public-key certificate to a principal (see §12.2.1).

Now that users' private keys are generated by TA, they have to trust TA absolutely, completely and unconditionally: namely, they must not feel uncomfortable for the situation that TA can read all of their private communications or forge all of their signatures. Therefore, ID-based cryptography should only be suitable for

applications where an unconditional trust is acceptable by the users; for example, in an organization environment in which the employer has the complete ownership of the information communicated to and from the employees; then the employer can play the role of TA.

With a principal's uniquely identifiable identity being directly used her/his public key, during the use of an ID-based public-key cryptosystem there is no need for the user to establish a key channel; namely, the "key channels" in Figures 7.1 and 9.1 are no longer needed. Moreover, ke in Figures 7.1 and kv in Figure 9.1 can be replaced with a string of a piece of self-evident information, for example, a globally distinguishable identity.

12.3.1 Shamir's ID-Based Signature Scheme

In Shamir's ID-based signature scheme there are four algorithms:

- **Setup:** this algorithm generates global system parameters and master-key;
- **User-key-generate:** this algorithm, inputting master-key and an arbitrary bit-string $id \in \{0,1\}^*$, outputs private-key which corresponds to id ; this is an instantiation of (12.3.1);
- **Sign:** a signature generation algorithm; inputting a message and the signer's private key, it outputs a signature;
- **Verify:** a signature verification algorithm; inputting a message-signature pair and id , it outputs True or False.

A concrete description of Shamir's ID-based signature scheme can be given as follows.

System Parameters Setup

TA sets up the following system parameters before opening a key generation center to offer the key generation service:

1. n : the product of two large primes, only TA knows these two primes;
2. e : an integer satisfying $\gcd(e, \phi(n)) = 1$; this value will be used as signature verification exponent similar to that in the RSA signature scheme, however, it is shared by the system-wise users;
3. d : an integer satisfying $ed \equiv 1 \pmod{\phi(n)}$; this value is TA's master-key;
4. $h : \{0,1\}^* \mapsto \mathbb{Z}_{\phi(n)}$; this is a cryptographically strong one-way hash function.

The parameters n, e and the description of the function h are made public to the system-wise users. Since TA is assumed to be the system-wise well-known principal,

the published system parameters will also be well-known by all users in the system (for example, maybe these parameters will be hardwired into every application that uses this scheme).

TA now opens the key generation center.

User Key Generation

Let ID denote user Alice's uniquely identifiable identity. We assume that ID contains a sufficient quantity of redundancy such that it is impossible for any other user in the system to also have ID as her/his identity. Having performed physical identification of Alice and made sure the uniqueness of ID , TA's key generation service is as follows

$$g \equiv ID^d \pmod{n}. \quad (12.3.2)$$

Notice that (12.3.2) is an instantiation of (12.3.1), where d is in place of **master-key**, ID is in place of **public-key** and g is in place of **private-key**. The value g will be Alice's private key. Since with $ed \equiv 1 \pmod{\phi(n)}$ we have $\gcd(d, \phi(n)) = 1$, the uniqueness of ID implies the uniqueness of Alice's private key g .

We should also notice that because ID contains a sufficient quantity of redundancy, it is a recognizable message. The user-key generation procedure is in fact TA's RSA signature on a recognizable message. Therefore, the difficulty for forging an RSA signature on a recognizable message (see §9.3.1.1) forms the security basis to prevent any user from gaining a private key without being physically identified by TA. Shamir also suggested to use $f(ID)$ in place of ID where f is a cryptographically strong hash function. This is a prudential and desirable measure.

Signature Generation

To sign a message $m \in \{0, 1\}^*$, Alice chooses $r \in_U \mathbb{Z}_n^*$, and computes

$$\begin{aligned} t &\stackrel{\text{def}}{=} r^e \pmod{n}, \\ s &\stackrel{\text{def}}{=} g \cdot r^h(t \parallel m) \pmod{n}. \end{aligned}$$

The signature is the pair (s, t) .

Signature Verification

Given the message m and the signature (s, t) , Bob uses Alice's identity ID to verify the signature using the following step:

$$\text{Verify}(ID, s, t, m) = \text{True} \text{ iff } s^e \equiv ID \cdot t^{h(t \parallel m)} \pmod{n}.$$

Algorithm 12.1 summarizes Shamir's ID-based signature scheme.

Algorithm 12.1: Shamir's Identity-based Signature Scheme**System Parameters Setup**

TA sets up:

1. n : the product of two large primes, only TA knows these two primes;
2. e : an integer satisfying $\gcd(e, \phi(n)) = 1$; this value will be used as the public signature verification exponent shared by the system-wise users;
3. d : an integer satisfying $ed \equiv 1 \pmod{\phi(n)}$; this value is TA's master-key;
4. $h : \{0, 1\}^* \mapsto \mathbb{Z}_{\phi(n)}$; this is a cryptographically strong one-way hash function.

TA keeps d as the system private key (master-key), and publicizes the system parameters (n, e, h) .

User Key Generation

Let ID denote user Alice's uniquely identifiable identity. Having performed physical identification of Alice and made sure the uniqueness of ID , TA's key generation service is as follows

$$g \equiv ID^d \pmod{n}.$$

Signature Generation

To sign a message $m \in \{0, 1\}^*$, Alice chooses $r \in_U \mathbb{Z}_n^*$, and computes

$$\begin{aligned} t &\stackrel{\text{def}}{=} r^e \pmod{n}, \\ s &\stackrel{\text{def}}{=} g \cdot r^h(t \parallel m) \pmod{n}. \end{aligned}$$

The signature is the pair (s, t) .

Signature Verification

Given the message m and the signature (s, t) , Bob uses Alice's identity ID to verify the signature using the following step:

$$\text{Verify}(ID, s, t, m) = \text{True} \text{ iff } s^e \equiv ID \cdot t^{h(t \parallel m)} \pmod{n}.$$

Figure 12.1.

12.3.1.1 How does it work?

A **True** case in a signature verification shows that Alice has in possession of both $ID \cdot t^{h(t||m)}$ and its *unique* e -th root modulo n (which is s , and the uniqueness is guaranteed by the fact $\gcd(e, \phi(n)) = 1$).

The construction of $ID \cdot t^{h(t||m)}$ need not be a difficult job; for example, one can choose a random t , construct $h(t || m)$, then $t^{h(t||m)} \pmod n$ and lastly multiply ID to the result. However, given that the value so constructed is *recognizable* due to the involvement of a cryptographically strong hash function in the construction (see the meaning of a recognizable message in §9.3.1.1), it is difficult to extract the e -th root of such value. This way of signature forgery is the RSAP (see Definition 8.4)^a which is assumed to be a hard problem (see Assumption 8.3). It is then assumed that Alice should have in possession of the e -th root of ID , which is her private key issued by TA, and should have used the private key in the construction of the signature pair.

However we must note that we have not provided a valid argument in that to create Alice's signature one has to use the e -th root of ID . Clearly, the difficulty for signature forgery is related to that for constructing $ID \cdot t^{h(t||m)} \pmod n$ and its e -th root modulo n ; this difficulty is certainly related to the details of the hash function used. Similar to the situation for providing security proofs for other digital signature schemes, a rigorous argument on the security for Shamir's ID-based signature scheme requires to reason about the details of a message-recognizability transformation method.

12.3.2 What Exactly does ID-based Cryptography Offer?

In public-key cryptography in the usual sense, for Bob to verify a signature of Alice using her public key, Bob should also verify, separately, the authenticity of Alice's public-key, e.g., by verifying her key certificate (which links Alice's public key with her identity). Namely, Bob should make sure that the key channel from to Alice has been properly established (see Figure 9.1).

It is interesting to realize that in an ID-based signature scheme, there is no need for Bob to perform a separate verification for the proper establishment of a key channel. Here, a **True** case in a signature verification shows Bob two things at the same time:

- the signature has been created by Alice using her private key which is behind her ID; and
- her ID has been certified by TA, and it is the result of TA's certification of her ID that has enabled Alice to create the signature.

^aWe should note that this statement is different from stating: to forge a valid signature for Shamir's ID-based signature scheme is the RSAP.

Being able to simultaneously verifying these two things in one go seems to be the nice feature offered by an ID-based signature scheme. Being able to avoid transmitting a certificate from the signer to the verifier also saves the communication bandwidth. This feature also brands the ID-based cryptography another name: **non-interactive public key cryptography**. We will see in a moment that the non-interaction property will make a better sense in an ID-based encryption system.

Finally we must recap and remember that TA can forge any user's signature! Therefore, Shamir's ID-based signature scheme is not suitable for applications in an open system environment, rather, it is more suitable for those in a closed system in which TA has a legitimate ownership of all information in the whole system. This is unfortunately a very restrictive setting.

A challenging open problem is to design an ID-based signature scheme which is free from this restrictive setting. Another open problem is to design an ID-based signature scheme which features non-interactive key revocation. Key revocation is necessary when a user's private key is compromised.

It seems that without having these two open problems solved, an ID-based signature scheme will have rather limited applications. We shall see in the remainder of this chapter that one of these two problems, free from the need for an absolute trust on TA, can be solved for an ID-based encryption scheme.

12.3.3 ID-Based Encryption from Pairings on Elliptic Curves

Shamir's original ID-based public-key cryptosystem is a digital signature scheme. He also conjectured the existence of ID-based encryption systems. After Shamir's posing of the problem, several ID-based cryptosystems have been proposed [BF01], [Coc01], [HJW00], [MY91], [TI89], [Tan88], [SOK00]. Among them, the proposal of Boneh and Franklin [BF01] is the most practical realization. In the rest of this chapter we shall introduce the scheme of Boneh and Franklin.

This scheme utilizes a special property due to a "pairing-mapping technique" in an abelian group of the points on some elliptic curves. The special property is that, in an elliptic-curve group in which a "pairing-mapping" can be efficiently computed, the **decisional Diffie-Hellman problem** is easy. Let us first introduce the decisional Diffie-Hellman problem and view its easy case.

12.3.3.1 Special groups with easy decisional Diffie-Hellman problem

In Definition 8.1 we have introduced the CDHP (the computational version of the Diffie-Hellman problem). The decisional version of the problem can be defined as follows.

Definition 12.1: (Decisional Diffie-Hellman Problem) (DDHP)

INPUT $\text{desc}(G)$: the description of an abelian group G ;
 $(g, g^a, g^b, g^c) \in G^4$ where g is a generator of the group G ;

OUTPUT YES if $ab \equiv c \pmod{\#G}$.

The DDHP can not be harder than the CDHP. If there exists a CDHP oracle, then on inputting (g, g^a, g^b, g^c) , the oracle can find g^{ab} from the first three elements in the input, and thus can answer the DDHP by checking if the output from the CDHP oracle is equal to g^c .

In the *general case* of abelian groups we do not know more than this relation between these two problems; moreover, we do not know any efficient algorithm to answer the DDHP. The difficulty to answer the DDHP has rendered the problem a standard and widely accepted intractability assumption (see Assumption 13.2) for underlying the security of many cryptographic systems, e.g., [BM90a], [Bra93], [CS98], [NR97], [Sta96].

Nevertheless, Menezes, Okamoto and Vanstone [MOV83] showed that in a special case of groups, there exists an efficient algorithm to “map-in-pair” two group elements in a special group to a non-degenerated element in a group of a finite field. The special group, denoting it by G_1 , is a group of the points on a supersingular elliptic curve, and the mapped-to group, denoting it by G_2 , is a subgroup of a finite field; here we have $\#G_1 = \#G_2$. The “mapping-in-pair” (“pairing” for short) is an efficient algorithm which can reduce the DLP (the discrete logarithm problem, see Definition 8.2) in G_1 to that in G_2 (and therefore the supersingular elliptic curves do not offer a value for the elliptic-curve cryptography). Joux and Nguyen [JN01] further made it clear that using the pairing technique, the DDHP in certain elliptic-curve-based groups can be answered efficiently. We shall see this in a moment.

In Appendix ?? we shall introduce the basics of the arithmetic over points on an elliptic curve and the principle of the elliptic-curve cryptography. There, we shall see that, by defining an addition operation over the points on an elliptic curve, the points will form an additive abelian group. In such a group we can also define the DLP, the CDHP and the DDHP in a similar manner as these problems are defined in groups based on finite fields.

The DLP

INPUT two elements $g, h \in G_1$ with g a group generator;

OUTPUT integer a such that $h = ag$.

The CDHP

INPUT three elements $g, ag, bg \in G_1$ with g a group generator;

OUTPUT element $(ab)g \in G_1$.

The DDHP

INPUT four elements $g, ag, bg, cg \in G_1$ with g a group generator;

OUTPUT YES iff $c \equiv ab \pmod{\#G_1}$.

Moreover, in the general case of elliptic-curve groups all these problems are hard. However, as we have mentioned above, in some special case of elliptic curves, the pairing technique provides an efficient decision algorithm to answer the DDHP, thanks to the observation made by Joux and Nguyen [JN01].

Boneh and Franklin proposed a practical ID-based cryptosystem [BF01] which applies a fact and an assumption we have discussed so far:

Fact: The DDHP in G_1 can be answered with an efficient pairing algorithm;

Assumption: The CDHP and the DLP in G_1 remain to be hard (by the suitable choice of a curve).

To introduce the ID-based cryptosystem of Boneh and Franklin, we only need to know, at a notational level, a few properties of the pairing mapping. We assume that $\#G_1 = \#G_2 = p$ which is a large prime number.

The Pairing. $e : (G_1, +)^2 \mapsto (G_2, \cdot)$ which satisfies the following properties:

1. Bilinear: $e(g + h, u) = e(g, u)e(h, u)$ and $e(g, u + v) = e(g, u)e(g, v)$ for all $g, h, u, v \in G_1$;
2. Non-degenerate: $e(g, g) \neq 1$ for all $g \in G_1$ with $g \neq 1$; observe that since $\#G_1 = \#G_2 = p$ which is prime, so g is a generator of G_1 if and only if $e(g, g)$ is a generator of G_2 ;
3. $e(g, h)$ is efficiently computable for all $g, h \in G_1$.

From these properties, it is easy to show that the DDHP in $(G_1, +)$ can be efficiently answered. First, with the properties 1 and 2, we have, for all $g, h \in G_1$ and for all $a < p, b < p$:

$$e(ag, bh) = e(g, h)^{ab}.$$

Then for the input tuple $(g, ag, bg, cg) \in G_1^4$, since

$$e(ag, bg) = e(g, g)^{ab},$$

$$e(g, cg) = e(g, g)^c,$$

therefore

$$c \equiv ab \pmod{p} \text{ iff } e(ag, bg) = e(g, cg).$$

With the property 3, this decision can be efficiently done.

There are two kinds of pairing techniques, named the Weil pairing and the Tate pairing respectively. These pairing algorithms are out of the scope of this book. The reader may study them from [BF01], [Gal01].

12.3.3.2 The ID-based cryptosystem of Boneh and Franklin

There are four algorithms in the ID-based cryptosystem of Boneh and Franklin:

- **Setup:** this algorithm generates global system parameters and **master-key**;
- **User-key-generate:** this algorithm, inputting **master-key** and an arbitrary bit-string $\text{id} \in \{0, 1\}^*$, outputs **private-key** which corresponds to id ; this is an instantiation of (12.3.1);
- **Encrypt:** a probabilistic encryption algorithm; it encrypts a message under the public key id ;
- **Decrypt:** a decryption algorithm; inputting a ciphertext and **private-key**, it returns the corresponding plaintext.

System Parameters Setup

TA sets up the following system parameters before opening a key generation center to offer the key generation service:

1. Generate two groups $(G_1, +)$, (G_2, \cdot) of prime order p and a mapping-in-pair $e : (G_1, +)^2 \mapsto (G_2, \cdot)$. Choose an arbitrary generator $P \in G_1$.
2. Pick $s \in_U \mathbb{Z}_p$ and set $P_{\text{pub}} \stackrel{\text{def}}{=} sP$; s is in the position of **master-key**.
3. Choose a cryptographically strong hash function $F : \{0, 1\}^* \mapsto G_1$; this hash function is for mapping a user's id to an element of G_1 .
4. Choose a cryptographically strong hash function $H : G_2 \mapsto \{0, 1\}^n$; this hash function determines that \mathcal{M} (the plaintext message space) is $\{0, 1\}^n$.

TA publishes the system parameters and their descriptions

$$(G_1, G_2, e, n, P, P_{\text{pub}}, F, H),$$

and keeps s as the system private key. Since TA is assumed to be the system-wise well-known principal, the published system parameters will also be well-known by

all users in the system (for example, maybe these parameters will be hardwired into every application that uses this scheme).

Notice that the secrecy of the master key s is protected by the difficulty of the DLP in G_1 .

TA now opens the key generation center.

User Key Generation

Let ID denote user Alice's uniquely identifiable identity. We assume that ID contains a sufficient quantity of redundancy such that it is impossible for any other user in the system to also have ID as her/his identity. Having performed physical identification of Alice and made sure the uniqueness of ID , TA's key generation service is as follows

1. Compute $Q_{ID} \stackrel{\text{def}}{=} F(ID)$, this is an element in G_1 , and is Alice's ID-based public key;
2. Set Alice's private key d_{ID} to be $d_{ID} \stackrel{\text{def}}{=} sQ_{ID}$.

Notice that as a hashed value, Q_{ID} should look random. However, with ID containing a sufficient quantity of redundancy and being the pre-image of Q_{ID} via the cryptographically strong hash function F , Q_{ID} is a *recognizable* element according to our agreement regarding the "recognizable message" which we have made in §9.3.1.1. Therefore, there is essentially no difference to view ID as Alice's public key, or to view Q_{ID} as Alice's public key.

We should also notice that Alice's private key is protected by the difficulty of the CDHP in G_1 ; this is because Q_{ID} must be generated by P (a generator element of G_1) and so we can denote it by $Q_{ID} = aP$ for some $a < p$; then from $P, P_{pub}(= sP), Q_{ID}(= aP)$, to find

$$d_{ID} = sQ_{ID} = (sa)P$$

is clearly a CDHP in G_1 .

Encryption

To send confidential messages to Alice, Bob first obtains the system parameters $(G_1, G_2, e, n, P, P_{pub}, F, H)$. As TA is the system-wise well-known principal, these parameters should be either already well-known or easily available to Bob. Using these parameters, Bob then computes

$$Q_{ID} = F(ID).$$

Let the message be blocked into n -bit blocks. Then to encrypt $M \in \{0, 1\}^n$, Bob picks $r \in_U \mathbb{Z}_p$ and computes

$$g_{\text{ID}} \stackrel{\text{def}}{=} e(Q_{\text{ID}}, rP_{\text{pub}}) \in G_2,$$

$$C \stackrel{\text{def}}{=} (rP, M \oplus H(g_{\text{ID}})).$$

The ciphertext is $C = (rP, M \oplus H(g_{\text{ID}}))$. Thus, the ciphertext is a pair comprising of a point in G_1 and a block in $\{0, 1\}^n$. Namely, \mathcal{C} (the ciphertext space) is $G_1 \times \{0, 1\}^n$.

Decryption

Let $C = (U, V) \in \mathcal{C}$ be a ciphertext encrypted using Alice's public key ID. To decrypt C using her private key $d_{\text{ID}} \in G_1$, Alice computes

$$V \oplus H(e(d_{\text{ID}}, U)). \quad (12.3.3)$$

Algorithm 12.2 summarizes the identity-based cryptosystem of Boneh and Franklin.

12.3.3.3 How does it work?

Observe

$$e(d_{\text{ID}}, U) = e(sQ_{\text{ID}}, rP) = e(Q_{\text{ID}}, rP)^s = e(Q_{\text{ID}}, (rs)P) = e(Q_{\text{ID}}, rP_{\text{pub}}) = g_{\text{ID}}.$$

Therefore, the value inside the hash function in (12.3.3) is in fact g_{ID} . Then

$$V \oplus H(e(d_{\text{ID}}, U)) = M \oplus H(g_{\text{ID}}) \oplus H(g_{\text{ID}}) = M$$

since bitwise-XOR-ing is self inverting.

12.3.3.4 Extension to an open system version

We must notice that TA can decrypt every ciphertext message sent to every principal in the system! Therefore, the basic scheme of Boneh and Franklin is not suitable for applications in an open system. However, their basic scheme can be extended to one which is suitable for applications in an open system environment. We describe here an extension method which is a simplified variation of the method discussed in the paper of Boneh and Franklin.

The basic idea is to, of course, use multiple TAs. However, doing so will be interesting only if it won't cause a blowup on the number of individual user's ID, nor on the size of the ciphertext. Here is one way to do it. We describe the case of two TAs. It is trivial to extend to many TAs.

Algorithm 12.2: The Identity-based Cryptosystem of Boneh and Franklin**System Parameters Setup** (performed by TA)

1. Generate two groups $(G_1, +)$, (G_2, \cdot) of prime order p and a mapping-in-pair $e : (G_1, +)^2 \mapsto (G_2, \cdot)$. Choose an arbitrary generator $P \in G_1$.
2. Pick $s \in_U \mathbb{Z}_p$ and set $P_{pub} \stackrel{\text{def}}{=} sP$; s is in the position of master-key.
3. Choose a cryptographically strong hash function $F : \{0, 1\}^* \mapsto G_1$; this hash function is for mapping a user's id to an element of G_1 .
4. Choose a cryptographically strong hash function $H : G_2 \mapsto \{0, 1\}^n$; this hash function determines that \mathcal{M} (the plaintext message space) is $\{0, 1\}^n$.

TA keeps s as the system private key (master-key), and publicizes the system parameters and their descriptions

$$(G_1, G_2, e, n, P, P_{pub}, F, H),$$

User Key Generation

Let ID denote user Alice's uniquely identifiable identity. Having performed physical identification of Alice and made sure the uniqueness of ID, TA's key generation service is as follows

1. Compute $Q_{ID} \stackrel{\text{def}}{=} F(\text{ID})$, this is an element in G_1 , and is Alice's ID-based public key;
2. Set Alice's private key d_{ID} to be $d_{ID} \stackrel{\text{def}}{=} sQ_{ID}$.

Encryption

To send confidential messages to Alice, Bob first obtains the system parameters $(G_1, G_2, e, n, P, P_{pub}, F, H)$. Using these parameters, Bob then computes

$$Q_{ID} = F(\text{ID}).$$

Let the message be blocked into n -bit blocks. Then to encrypt $M \in \{0, 1\}^n$, Bob picks $r \in_U \mathbb{Z}_p$ and computes

$$\begin{aligned} g_{ID} &\stackrel{\text{def}}{=} e(Q_{ID}, rP_{pub}) \in G_2, \\ C &\stackrel{\text{def}}{=} (rP, M \oplus H(g_{ID})). \end{aligned}$$

The ciphertext is $C = (rP, M \oplus H(g_{ID}))$.

Decryption

Let $C = (U, V) \in \mathcal{C}$ be a ciphertext encrypted using Alice's public key ID. To decrypt C using her private key $d_{ID} \in G_1$, Alice computes

$$V \oplus H(e(d_{ID}, U)).$$

Figure 12.2.

System Parameters Setup

Let parameters $(G_1, G_2, e, n, P, F, H)$ be identical to those defined in §12.3.3.2. Let further

$$\begin{aligned} P_1 &\stackrel{\text{def}}{=} s_1 P, \\ P_2 &\stackrel{\text{def}}{=} s_2 P \end{aligned}$$

such that the tuple

$$(P, P_1, P_2)$$

is in the position of

$$(P, P_{\text{pub}})$$

in §12.3.3.2, that is, s_1 and s_2 are the two master keys of TA_1 and TA_2 , respectively.

Thus, $(G_1, G_2, e, n, P, P_1, P_2, F, H)$ is the system-wise public parameters. These parameters can be “hardwired” into applications.

User Key Generation

Let ID denote user Alice’s uniquely identifiable identity. For $i = 1, 2$, the key generation service provided by TA_i is as follows:

1. Compute $Q_{\text{ID}} \stackrel{\text{def}}{=} F(\text{ID})$, this is an element in G_1 , and is Alice’s unique ID-based public key;
2. Set Alice’s private key $d_{\text{ID}}^{(i)}$ to be $d_{\text{ID}}^{(i)} \stackrel{\text{def}}{=} s_i Q_{\text{ID}}$.

Alice has her private key to be the sum:

$$d_{\text{ID}} \stackrel{\text{def}}{=} d_{\text{ID}}^{(1)} + d_{\text{ID}}^{(2)}.$$

If the two TAs do not collude, then this private key is not known to them.

Notice that Alice has a single public key: ID .

Encryption

To send confidential messages to Alice, Bob first obtains the system parameters $(G_1, G_2, e, n, P, P_1, P_2, F, H)$. Using these parameters, Bob then computes

$$Q_{\text{ID}} = F(\text{ID}).$$

Let the message be blocked into n -bit blocks. Then to encrypt $M \in \{0, 1\}^n$, Bob picks $r \in_U \mathbb{Z}_p$ and computes

$$g_{\text{ID}} \stackrel{\text{def}}{=} e(Q_{\text{ID}}, r(P_1 + P_2)) \in G_2,$$

$$C \stackrel{\text{def}}{=} (rP, M \oplus H(g_{\text{ID}})).$$

The ciphertext is C . Thus, the ciphertext is a pair comprising of a point in G_1 and a block in $\{0, 1\}^n$. Namely, \mathcal{C} (the ciphertext space) is $G_1 \times \{0, 1\}^n$.

Decryption

Let $C = (U, V) \in \mathcal{C}$ be a ciphertext encrypted using Alice's public key ID. To decrypt C using her private key $d_{\text{ID}} \in G_1$, Alice computes

$$V \oplus H(e(d_{\text{ID}}, U)).$$

Notice that

$$\begin{aligned} e(d_{\text{ID}}, U) &= e(s_1 Q_{\text{ID}} + s_2 Q_{\text{ID}}, rP) \\ &= e(s_1 Q_{\text{ID}}, rP) e(s_2 Q_{\text{ID}}, rP) \\ &= e(Q_{\text{ID}}, (rs_1)P) e(Q_{\text{ID}}, (rs_2)P) \\ &= e(Q_{\text{ID}}, r(P_1 + P_2)) \\ &= g_{\text{ID}}. \end{aligned}$$

So Alice has recovered g_{ID} and hence is able to decrypt:

$$V \oplus H(e(d_{\text{ID}}, U)) = M \oplus H(g_{\text{ID}}) \oplus H(g_{\text{ID}}) = M$$

since bitwise-XOR-ing is self inverting.

Discussions

- Comparing with the single TA case, computations for encryption and decryption doubled. But there is no increase on the number of Alice's ID, or the size of ciphertext.
- While colluding TAs can do the decryption too, not any single of them can. When more TAs are used, confidence of no corruption becomes evident. It is easy to see that increasing the number of TAs, the number of Alice's ID and the size of the ciphertext remains unchanged. However, the computations in encryption and in decryption increase linearly proportional to the number of TAs.
- With several TAs are used, to perform decryption of an end user's ciphertext requires the full collusion of all TAs. If we trust that at least one of the TAs is honestly trustworthy, then eavesdropping by TAs is prevented. Thus, this extended IBE scheme is suitable for applications in an open system environment.

12.3.4 Non-interaction Property: Authentication Without Key Channel

When Bob is to send a confidential message to Alice using a cryptographic technique in the usual sense, he should first establish a key channel from him to her (see Figure 7.1). In public-key cryptography in the usual sense, a key channel can be a directory based: for example, based on the sender verifying the public-key certificate of the recipient. Thus, the sender should first make a request to the recipient for obtaining her/his public-key certificate. Considering applications in an open system in which principals cannot memorize a link between a public key and its owner, it is necessary for the sender and recipient to have some interaction(s) in order to establish a key channel before sending a confidential message encrypted under the public key of the recipient.

It is interesting to realize that in an ID-based public-key cryptosystem, the notion of key-channel establishment is not only unnecessary, but in fact is impossible! Even if Bob does make a request for Alice to send her ID-based public key, there will be *no way* for Bob to verify whether the ID received from her is indeed her public key. All Bob needs to do is to treat the acquired ID as a valid public key of Alice and goes ahead to use it to encrypt the message. If in the end Alice is able to decrypt the ciphertext sent by Bob, then her ID is indeed her public key. Thus, as long as each ID is sufficiently precise in terms of uniquely pinpointing an individual (this has to be made sure by TA, or TAs), then there is no need for the sender to be in an interaction with the recipient before using an ID-based encryption algorithm for sending confidential messages. This is why an ID-based public-key cryptosystem is also called non-interactive public key cryptosystem.

The impossibility to directly confirm that a public key belongs to a principal, i.e., the absence of a publicly verifiable key certificate, will enable an interesting application of ID-based public-key cryptography: the e-spy problem. We shall see this application in a latter chapter after we have introduced **zero-knowledge protocols**.

12.3.5 Two Open Questions for Identity-based Public-key Cryptography

First, consider the user-key generation procedure

$$\text{private-key} = F(\text{master-key}, \text{public-key}).$$

Here a user submits **public-key** of the user's choice which can be potentially malicious, however TA will simply serve this computation unconditionally and hand **private-key** back to the user. Notice that for the cryptosystem to be identity based, i.e., to be non-interactive or non-certificate-based, the function $F()$ needs to be deterministic. Thus, the procedure for user-key generation has no any random input. In other words, each user's **private-key** is a deterministic image of **master-key**.

This computation is in general considered to be a dangerous one (in terms of cryptanalysis against **master-key**) and, after the well-known critique of Goldwasser-Micali to the deterministic trapdoor function model of Diffie-Hellman [GM84a], has been widely avoided in the standard applications of public-key cryptography (e.g., by TA adding his random input).

ID-based public-key cryptography with probabilistic **private-key** is thus an interesting thing to pursue.

Secondly, a challenging open problem is to design an ID-based cryptosystem which features non-interactive key revocation, which is necessary if an end-user's private key is compromised.

12.4 Chapter Summary

In this chapter we have introduced authentication framework techniques for public-key cryptography. These techniques include directory based certification framework using a hierarchically organized certification authorities, web-of-trust based non-hierarchical authentication framework, and identity based technique in which a public key can be non-random and so self-authenticated.

Recent progress in identity based public-key cryptography not only provides practical and convenient ways of authenticating (recognizing) public keys, but also opens some new kinds of authentication services. We will introduce some these services in a later chapter.

Part V

**FORMALISM
APPROACHES TO
SECURITY
ESTABLISHMENT**

Systems' complexity, in particular, that due to communications among systems and system components, has been the main cause of failures that are introduced into computing and communications systems. Cryptographic systems (algorithms and protocols), which are usually complex systems, are open to a further cause of failures: attacks. While ordinary computing and communications systems work in a friendly environment (a user will try carefully not to input or send invalid data to a program or to a communication partner in order to avoid system crash), cryptographic systems are meant to work in a hostile environment. In addition to possible failures in an ordinary way, cryptographic systems may also fail due to deliberate abnormal use. They are subject to all imaginable attacks which can be launched not only by an external attacker who may interact with the system without being invited, but also by a legitimate user (attacker from inside). Often cryptographic systems, even designed by security experts, are vulnerable to failures. The long hidden flaws in Needham-Schroeder protocols are a well-known lesson on unreliability of security experts [NS78].

Formal methodologies for systems analysis involve systematic procedures for the analysis task. The systematic procedures base their foundations squarely on mathematics in order to preserve rigorousness and generality in the modelling and construction of complex systems and in the observation and reasoning about of their behavior. These procedures either develop systems in a systematic manner so that the desired system properties are evidently demonstrable, or examine systems via systematic search so that errors in systems are uncovered.

The error-prone nature of cryptographic systems has raised a wide consensus for these systems to be developed and/or analyzed by formal methodologies. This part contains three chapters on the topics of formalism approaches to the development and the analysis of cryptographic systems.

FORMALISM FOR RIGOROUSNESS - SECURITY DEFINITIONS AND FORMALLY PROVABLE SECURITY

13.1 Introduction

Since secrecy is at the heart of cryptography, let us now re-consider security notions for encryption algorithms. We shall later see that the confidentiality-oriented security notions established in this chapter will have a wider generality for basing other kinds of security services.

So far we have confined ourselves to a very basic security notion for confidentiality which is described in Property 8.2: we only face a passive attacker and such an attacker may break a target cryptosystem only in the “all-or-nothing” sense.

In reality, however, attackers are most likely to be active while without losing their passive ability of eavesdropping: they may modify a ciphertext or calculate a plaintext in some unspecified ways and send the result to an unwitting user to get *oracle services* (see §8.3.2 for the meaning of oracle services). Therefore, a security notion which only deals with passive attackers is not strong enough. We need to anticipate Malice, our active and clever attacker (review §2.2 for the power of Malice).

Moreover, in many applications, plaintext messages may contain *a-priori* information which can be guessed easily. For example, such *a-priori* information can be a value from a known range of salaries, a candidate’s name from several known candidates in a voting protocol, one of several possible instructions, or even

one-bit information about a plaintext (e.g., a BUY/SELL instruction to a stock broker, or HEADS/TAILS in a remote “Coin Tossing” protocol). To guess such information about a plaintext encrypted under a one-way-trapdoor based public-key encryption algorithm, Malice can simply re-encrypt the guessed plaintext and see if the result matches the target ciphertext. Therefore, a confidentiality notion in “all-or-nothing” sense cannot be sufficiently strong.

To anticipate attacks of various degrees of severity we need various more stringent security notions. In order to establish more stringent security notions, the first step is to *formalize* the problems properly. In the area of cryptographic systems with formally provable security, various attack games have been proposed to model and capture various attack scenarios. Such games are played between Malice and an oracle. The rule of game allows Malice to obtain cryptographic assistance provided by the oracle, virtually on Malice’s demand. We can view that such assistance provides a kind of “cryptanalysis training course” to Malice. A cryptosystem is regarded to be secure with respect to the formal model of an attack game if, even given adequate “cryptanalysis training course”, Malice cannot succeed with satisfaction.

The second aspect of the formalism treatment on security is a rigorous measure on Malice’s satisfaction. In the area of formally provable security for cryptographic systems, security of a cryptographic system concerns a quantitative relation which links the security of the cryptosystem to some intractable problem in the theory of computational complexity. A standard technique for establishing a high degree of confidence in security is to express and translate the satisfaction level of Malice against a target cryptographic system into some values which measure *how fast* and *how often* for one to solve some reputable intractable problems in the theory of computational complexity. Such a translation is an efficient mathematical transformation, or a sequence of such transformations, leading from the alleged successful attack to a solution to a reputable intractable problem. Since we are highly confident about *how slow* and *how infrequent* for us to solve the latter problems, we will also have a definite (high) confidence about Malice’s *dissatisfaction* with the alleged attack on the target cryptosystem.

Due to the diversity of the attack scenarios and the use of several intractable problems as the computational basis for security, the area of cryptographic systems with formally provable security has adopted a language with plenty of jargons. With these jargons we can describe various security properties and requirements with convenience and precision. Here are a few examples of security statements.

- Cryptosystem X is *semantically secure* against a passive eavesdropper of an unbounded computational power, but is *malleable* by a *chosen ciphertext* attacker who only needs to work at *lunchtime*.
- Signature scheme Y is secure for the signers in terms of unforgeability against an *adaptively-chosen-message* attacker with respect to the integer factoriza-

tion problem; however, the formal proof of security is provided under a *random oracle* assumption.

- Electronic auction protocol Z is secure for the bidders in terms of *indistinguishability* of the winner against an *adaptively-chosen-ciphertext* attacker with respect to the decisional Diffie-Hellman problem, regardless of whether the attacker works at lunchtime, at *midnight* or in *small hours*.

In our study of formally provable security in this and the next chapters, the jargons appeared in these security statements will be defined or explained. Security statements such as those listed here will become much more meaningful after our study.

13.1.1 Chapter Outline

We begin in §13.2 with an introduction to the main theme of this chapter: formalism treatment on security which involves formal modelling of attack scenarios and precise measuring of the consequences. The formal treatment will show clearly inadequacy of the “all-or-nothing” based security notion we have introduced in Chapter 8. A strengthened security notion, “semantic security”, which means hiding any partial information about a message, will be introduced in §13.3. The inadequacy of semantic security will be exposed in §13.4. The exposure leads to several further steps of strengthening security notions: “chosen ciphertext security”, “adaptively chosen ciphertext security”, and “non-malleability”. These strengthened security notions and their relations will be studied in §13.5.

13.2 A Formal Treatment for Security

Speaking at a very abstract level, “formally provable security” for a cryptographic system is an affirmative *measure* on the system’s strength in resisting an attack. Secure systems are those in which Malice cannot do something bad *too often* or *fast enough*. Thus, the measure involves to establish success probabilities and computational cost.

To provide a concrete view of such a measure let us look at an “attack game” played between Malice and an “oracle” who models an innocent user of a cryptographic system in which the user should be inevitably in interactions with Malice. This game provides a formal treatment of a computational view of what we mean by security; it also makes a first step in our process of strengthening the security notion for cryptosystems (from what has so far been confined in Property 8.2).

Let Malice be the attacker and let \mathcal{O} denote an oracle in our game of attack. In the context of confidentiality, the target of Malice is a cryptosystem. Thus, “something bad” means that the game results in a breach of confidentiality regarding the target cryptosystem.

Protocol 13.1: Chosen Plaintext Attack

PREMISE

- i) Malice and an oracle \mathcal{O} have agreed on a target cryptosystem \mathcal{E} of plaintext message space \mathcal{M} and ciphertext message space \mathcal{C} ;
- ii) \mathcal{O} has fixed an encryption key ke for \mathcal{E} .

1. Malice chooses distinct messages $m_0, m_1 \in \mathcal{M}$ and sends them to \mathcal{O} ;
 (* The messages m_0, m_1 are called **chosen plaintext** messages. Malice has so far been in a “**find-stage**” for preparing m_0, m_1 ; he should of course prepare them in such a way that he hopes the encryption of them is easily recognizable. *)

2. If these two messages are not the same length, \mathcal{O} will pad the shorter one into the same length of the other;
 (* E.g., using a dummy string 0^d where d is the difference between the message lengths. *)

\mathcal{O} tosses a fair coin $b \in_U \{0, 1\}$ and performs the following encryption operation

$$c^* = \begin{cases} \mathcal{E}_{ke}(m_0) & \text{if } b = 0 \\ \mathcal{E}_{ke}(m_1) & \text{if } b = 1 \end{cases}$$

\mathcal{O} sends $c^* \in \mathcal{C}$ to Malice;

(* The ciphertext message c^* is called a *challenge ciphertext*. As convention in the area of provable security, a ciphertext with a “*” superscript is always considered as a challenge ciphertext. *)

(* Remember, c^* is a random variable of two random input values: fair coin b and an internal random operation of \mathcal{E} . *)

3. Upon receipt of c^* , Malice must answer either 0 or 1 as his working out of \mathcal{O} ’s coin tossing.
 (* Malice is now in a “**guess-stage**” for an educated guess of \mathcal{O} ’s coin tossing. An answer other than 0 or 1 is not allowed. *)

Figure 13.1.

Let us use Definition 7.1 to provide the syntactic notation for the target cryptosystem which has an encryption algorithm \mathcal{E} , plaintext space \mathcal{M} and ciphertext space \mathcal{C} . Nevertheless, we should make an important notice here: the encryption

algorithm \mathcal{E} is now probabilistic, that is, it has an internal random operation which has certain probability distribution and will cause the ciphertext output to be a random variable of this distribution. For example, if a plaintext message is encrypted under an encryption key twice, the two resultant ciphertexts will be different with an overwhelming probability.

Protocol 13.1 specifies an attack game.

In the attack game \mathcal{O} challenges Malice to answer the following question:

From which of the two experiments (ensembles) $\mathcal{E}_{ke}(m_0)$, $\mathcal{E}_{ke}(m_1)$ comes the challenge ciphertext c^* ?

We consider that Malice is a probabilistic polynomial-time distinguisher defined in Definition 4.14. This is not only because \mathcal{O} 's output is probabilistic, but also because Malice is *polynomially bounded* and therefore may wish to use a probabilistic polynomial-time (PPT) algorithm if he thinks such algorithms may be more efficient than deterministic ones (this is usually true as we have witnessed many times in Chapter 4). Denote by Adv the advantage of Malice for making a distinction. By Definition 4.14, Adv should be the difference between Malice's probabilistic distinguishing of the experiments $\mathcal{E}_{ke}(m_0)$ and $\mathcal{E}_{ke}(m_1)$:

$$\text{Adv} = \left| \text{Prob}[0 \leftarrow \text{Malice}(c^* = \mathcal{E}_{ke}(m_0))] - \text{Prob}[0 \leftarrow \text{Malice}(c^* = \mathcal{E}_{ke}(m_1))] \right|. \quad (13.2.1)$$

The probability space should include the probabilistic choices made by \mathcal{O} , by Malice and the internal random operation of the encryption algorithm. Also notice that Malice's answer is based not solely on the challenge ciphertext c^* , but also on the two chosen plaintext messages (m_0, m_1) . Only because of so, his answer is regarded as an "educated guess". However, for clarity in exposition, we have omitted (m_0, m_1) from Malice's input.

We must notice that there is an additional clue for Malice to "improve" his educated guess: \mathcal{O} tosses a fair coin. The advantage formulation (13.2.1) does not explicitly express how Malice could have used this clue, though *implicitly* we know that each probability term in (13.2.1) can never exceed $\frac{1}{2}$ since, e.g., the event " $c^* = \mathcal{E}_{ke}(m_0)$ ", can only occur with probability $\frac{1}{2}$. We should explicitly express Malice's use of this clue in his advantage formulation. Applying the conditional probability (Definition 3.3) with noticing the equal probability of $\frac{1}{2}$ for both cases of \mathcal{O} 's coin-tossing, (13.2.1) can be rewritten to

$$\text{Adv} = \left| \frac{1}{2} \text{Prob}[0 \leftarrow \text{Malice}(c^*) \mid c^* = \mathcal{E}_{ke}(m_0)] - \frac{1}{2} \text{Prob}[0 \leftarrow \text{Malice}(c^*) \mid c^* = \mathcal{E}_{ke}(m_1)] \right|. \quad (13.2.2)$$

By rule of the game, Malice is not allowed to answer other than 0 or 1, and hence the event of a wrong (correct) answer is complementary to that of a correct

(wrong) answer. So applying Property 5 of probability (§3.2), we have

$$\text{Adv} = \left| \frac{1}{2} \text{Prob}[0 \leftarrow \text{Malice}(c^*) \mid c^* = \mathcal{E}_{ke}(m_0)] - \frac{1}{2} (1 - \text{Prob}[0 \leftarrow \text{Malice}(c^*) \mid c^* = \mathcal{E}_{ke}(m_0)]) \right|,$$

that is,

$$\text{Prob}[0 \leftarrow \text{Malice}(c^*) \mid c^* = \mathcal{E}_{ke}(m_0)] = \frac{1}{2} \pm \text{Adv}. \quad (13.2.3)$$

The formulation (13.2.3) is often used to express an algorithmic advantage on top of that for a sheer guessing of fair coin tossing (probability $\frac{1}{2}$). So if $\text{Adv} = 0$, then Malice's probabilistic answer will be exactly the distribution of tossing a fair coin. Of course, we should not be too cynical about Malice's algorithmic advantage and should generously consider (i) $\text{Adv} > 0$ and (ii) the positive sign prefixing Adv . Clearly, (13.2.3) will also hold if 0 is replaced with 1.

From (13.2.3) we can also see that Malice's advantage can never exceed $\frac{1}{2}$ since a probability value cannot be outside the interval $[0, 1]$. Indeed, given that \mathcal{O} has exactly $\frac{1}{2}$ probability to have encrypted either of the two plaintexts, Adv formulated in (13.2.1) as the probability difference for *joint* events can never exceed $\frac{1}{2}$. The reader might be wondering what (13.2.3) would look like if \mathcal{O} tosses a *biased* coin, say one with $\frac{1}{4}$ probability to encrypt the plaintext query m_0 and $\frac{3}{4}$ probability to encrypt m_1 . Hint: replace $\frac{1}{2}$ in (13.2.2) with respective biased probability values and see how (13.2.3) will change. We will then realize that it *is* possible for Malice's advantage to exceed $\frac{1}{2}$, *provided* that \mathcal{O} tosses a biased coin.

We say that the target cryptosystem is secure against the attack game in Protocol 13.1 if $\mathcal{E}_{ke}(m_0)$ is indistinguishable from $\mathcal{E}_{ke}(m_1)$. Following Definition 4.15, this means there should exist no PPT distinguisher for any $\text{Adv} > 0$ as a non-negligible quantity. Equivalently, any Malice who can be successful to make a distinction, his Adv must be a negligible quantity. Here “negligible” is measured with respect to a security parameter of the target encryption scheme which is usually the size of the key material. We can consider Adv for any polynomially bounded Malice (i.e., any PPT algorithm) to be a slow-growing function of Malice's computational resources. Here “slow-growing” means that even if Malice adds his computational resources in a tremendous manner, Adv will only grow in a marginal manner so that Malice cannot be very happy about his “advantage”. This is exactly what we meant when we mentioned in the beginning of this chapter that Malice cannot do something bad too often or fast enough.

Since our argument has followed exactly Definition 4.15 for **polynomial indistinguishable** experiments, the new security notion we have just established can be named **polynomial indistinguishability of encryption**. Moreover, because the indistinguishability is between the two plaintexts chosen by Malice, the precise name for this new notion is **security with respect to polynomially indistinguishable chosen plaintext attack**. It is usually shorten to **IND-CPA security**.

Now that in the IND-CPA attack game in Protocol 13.1, Malice has freedom to choose plaintext messages, and is only required to answer sheer one-bit information about the chosen plaintexts: “Is the encrypted plaintext m_0 or m_1 ?”, the difficulty for Malice to break the target cryptosystem is drastically reduced from that to break the cryptosystem in the “all-or-nothing” sense of security (defined in Property 8.2.(i)). Indeed, all “textbook” public-key encryption algorithms (see §8.6 for the meaning of “textbook encryption”) we have introduced so far are insecure under IND-CPA. It is easy to see this for the RSA and Rabin cryptosystems since they are deterministic and thereby allow Malice to pinpoint m_0 or m_1 by re-encryption. We shall further see in §13.3.5 that the ElGamal cryptosystem specified in Algorithm 8.3, which provides a probabilistic encryption algorithm, is no longer secure under IND-CPA too.

With the difficulty for an attack being reduced, the security requirement on cryptosystems should be strengthened. To reduce the difficulty for Malice to attack cryptosystems, or speaking equivalently, to strengthen the security notion for cryptosystems, and to do so with formal rigorousness, is the main topic for this chapter.

13.3 Semantic Security — the Debut of Provable Security

The IND-CPA security notion which we have just defined is originally introduced by Goldwasser and Micali [GM84a]. They used **semantic security** to name this security notion. This notion means that a ciphertext does not leak any useful information about the plaintext (if we may regard that the length of the plaintext is not a piece of useful information) to any attacker whose computational power is polynomially bounded. They observed that, in many applications, messages may contain certain *a-priori* information which can be useful for an attack. For example, a ciphertext may only encrypt a simple instruction such as “BUY” or “SELL”, or one of the identities of a handful known candidates who are being voted. Goldwasser and Micali pointed out that public-key cryptosystems which are based on direct applications of one-way trapdoor functions are in general very weak for hiding such messages. We shall see that their critique does apply to each of the public-key cryptosystems which we have introduced in Chapter 8.

The need for this rather strong security notion is very real. The failure of a mental poker protocol provides a good illustration on the weakness of public-key cryptosystems as direct applications of one-way trapdoor functions. Let us first review the mental poker protocol of Shamir, Rivest and Adleman [SRA80].

13.3.1 The SRA Mental Poker Protocol

Alice lives in New York and Bob lives in London. They have never met, but they wish to play poker across the Atlantic. The same authors of the RSA cryptosystems

made this possible: Shamir, Rivest and Adleman proposed a protocol called “SRA mental poker” [SRA80].

Mental poker is played like ordinary poker however the cards are encoded into messages so that the card game can be played in communications. In order to play a poker game, Alice and Bob should first deal the cards fairly. Here “fair” means the following requirements:

- i) The deal must distribute all possible hands with equal probability and should not allow the same card to appear in two hands simultaneously;
- ii) Alice and Bob must know the cards in their own hand, but neither can have any information about the other’s hand;
- iii) Both Alice and Bob must be viewed as potential cheaters who cannot be relied upon to follow the rules of the protocol;
- iv) Alice and Bob should both be able to verify that a preceding game has been fairly played.

The idea behind the SRA mental poker is to make use of a cipher with the commutative property. In such a cipher, a message can be doubly encrypted by Alice and Bob using their respective secret keys and the resultant ciphertext must also be doubly decrypted by both of them. Let

$$C = E_X(M) \quad \text{and} \quad M = D_X(C)$$

denote the encryption and decryption algorithms performed by principal X . The commutative property of the cipher is that the following equations holds for any message M in its plaintext space:

$$\begin{aligned} M &= D_A(D_B(E_A(E_B(M)))) \\ &= D_B(D_A(E_B(E_A(M)))) \end{aligned} \tag{13.3.1}$$

That is, the plaintext message can be correctly retrieved even though the sequence of the double decryption can be independent from that of the double encryption.

For simplicity, suppose that Alice and Bob decide to play a game of one-card hand using a deck of three cards. Protocol 13.2 specifies a method for a fair deal of hands.

13.3.2 A Security Analysis Based on a Rough Notion of Security

For the time being let us suppose that the cryptosystem used in Protocol 13.2 is sufficiently strong in both single and double encryption operations. By saying that a cryptosystem is “sufficiently strong”, we mean that, given a plaintext (respectively,

Protocol 13.2: A Fair Deal Protocol for the SRA Mental Poker

PREMISE:

Alice and Bob have agreed on a commutative cipher with the properties in (13.3.1) and they have picked their own secret encryption keys;

They have agree on a deck of three cards M_1, M_2, M_3 .

GOAL:

They achieve a fair deal of a one-card hand for each party satisfying the fairness properties (i)-(iv).

1. Alice encrypts the three cards as $C_i = E_A(M_i)$ for $i = 1, 2, 3$; she sends to Bob these three ciphertexts at a random order;
(* Sending the encrypted cards at a random order models shuffling of the deck. *)
2. Bob picks at random one ciphertext; denoting it by C , he doubly encrypt C as $CC = E_B(C)$; he also picks at random another ciphertext, denoting it by C' ; he sends CC, C' to Alice;
(* CC determines Bob's hand; C' determines Alice's hand; the other encrypted card is discarded. *)
3. Alice decrypts both CC and C' ; the decryption of C' is her hand; the decryption of CC , denoting it by C'' , is returned to Bob;
4. Bob decrypts C'' and thereby obtains his hand.
(* They can now play their mental poker game. *)

Figure 13.2. The SRA mental poker protocol

a ciphertext) without giving the correct encryption key (respectively, decryption key), a polynomially bounded attacker cannot create a valid ciphertext from the given plaintext (respectively, cannot retrieve the plaintext from the given ciphertext). This is a so-called “all-or-nothing” sense of secrecy (review Property 8.2.(i) for the security notion which we have agreed for the cryptosystems introduced in Chapter 8). Under this notion of security we can now provide a security analysis for Protocol 13.2 with respect to the fairness properties (i)-(iv).

After a run of Protocol 13.2:

- Alice and Bob each obtains a hand of a card in $\{M_1, M_2, M_3\}$ with equal probability; this is because Alice has shuffled the deck in Step 1. Notice that it is Alice's interest to shuffle the deck at uniformly random to prevent Bob from having an advantage in choosing his hand. So fairness property (i) is satisfied.
- Each of the two parties knows her/his own hand after double decryption, but does not know the hand of the other party since none of them knows the discarded card. So fairness property (ii) is satisfied.
- It is obvious that the protocol does not rely on any party to be honest. So fairness property (iii) is satisfied.

Fairness property (iv) depends on whether or not the cryptosystems used in the protocol permits a honest verification after a poker game. Shamir et al suggested to use a variation of the RSA cryptosystem (see §??) where the two parties keep both of their encryption and decryption exponents secret before a poker game finishes, and they disclose these exponents to the other party for checking their honest conduct after a game finishes.

Let N be the shared RSA modulus. In this variation, Alice and Bob knows the factorization of N . Let (e_A, d_A) be Alice's encryption and decryption exponents, and (e_B, d_B) be Bob's encryption and decryption exponents. Knowing the factorization of N permits Alice (respectively, Bob) to compute d_A from e_A (respectively, d_B from e_B). They do so by solving the congruence (8.3.1). Then for principal X we have

$$E_X(M) = M^{e_X} \pmod{N}$$

$$D_X(C) = C^{d_X} \pmod{N}.$$

Since the RSA group is commutative, it is trivial to see the holding of (13.3.1). Before finishing of a game, both parties keeps their encryption and decryption exponents secret. Thus, no one can create a valid ciphertext which has been created by the other party; this prevents a party from testing which card has been encrypted under which ciphertext. Also, no one can decrypt a ciphertext which has been created by the other party. Thus, indeed, the cryptosystem is "sufficiently strong" as we have required to be.

It is now clear that after a game finishes, both parties can disclose their encryption and decryption exponents to the other party and thereby they can check that the encryption, double encryption and decryption have all been correctly performed. Thus, fairness property (iv) is satisfied.

In our analysis we have used a rather inadequate and unreasonable notion of security: a "sufficiently strong" cryptosystem means an attacker's inability to create a valid ciphertext from a given plaintext without the correct encryption key, or to decrypt a ciphertext without the correct decryption key. The inadequacy

and unreasonability of this security notion now become apparent. Lipton [Lip79] observed that Protocol 13.2 fails if it uses the variation of the RSA cryptosystem suggested by the original authors of the mental poker game. The failure is due to the cryptosystem's inability for hiding certain *a-priori* information in plaintext messages. Here, the *a-priori* information is the quadratic residuosity. Review §6.4, a number a is a quadratic residue modulo n if $\gcd(a, N) = 1$ and there exists $x < N$ such that

$$x^2 \equiv a \pmod{N}.$$

Notice that because $\phi(N)$ is even, the encryption exponent e and the decryption exponent d which satisfy congruence (8.3.1) must both be odd. Consequently, a plaintext M is a quadratic residue modulo N , i.e., $M \in \text{QR}_N$ if and only if the corresponding ciphertext $C \in \text{QR}_N$, since

$$C \equiv M^e \equiv (x^2)^e \equiv (x^e)^2 \pmod{N},$$

for some $x < N$. That is, the RSA encryption cannot change the quadratic residuosity of the plaintext message. Further review §6.4, we know that with the factorization of N , deciding $C \in \text{QR}_N$ can be easily done: first having C modulo each prime factor of N , then evaluating the Legendre symbol of the results using Algorithm 6.2.

Therefore, if some plaintext card(s) is (are) in QR_N and others (the other) are (is) not, then a party who knows Lipton's trick will have an unfair advantage in a game: (s)he will know exactly which of the cards will never be encrypted, whether under single encryption or double.

We conclude that the SRA mental poker protocol is not secure. To state our conclusion with the formal precision, we say that it is not secure with respect to the IND-CPA model specified in Protocol (Game) 13.1.

13.3.3 Probabilistic Encryption of Goldwasser and Micali

It is possible to fix Protocol 13.2 with respect to Lipton's attack. For example, forcing all cards to be chosen from QR_N provides a specific fix. However, Goldwasser and Micali envisioned a need for a general fix of a much bigger problem: the need of a stronger security notion: semantic security. They described their notion of semantic security to be the following property:

Property 13.1: (Semantic Security) *Whatever is efficiently computable about the plaintext given the ciphertext, is also efficiently computable without the ciphertext.*

They proposed a **probabilistic encryption** scheme which possesses this property. Let us name this scheme the GM cryptosystem. The GM cryptosystem en-

Algorithm 13.1: The Probabilistic Cryptosystem of Goldwasser and Micali

GM Key Setup

To set up a user's key material, user Alice performs the following steps:

- 1) choose two random prime numbers p and q such that $|p| = |q| = k$
 (* e.g., using Algorithm 4.7 with input 1^k *)
- 2) compute $N = pq$;
- 3) pick a random integer y satisfying $\left(\frac{y}{p}\right) = \left(\frac{y}{q}\right) = -1$
 (* thus $y \in J(N) \setminus QR_N$ *)
- 4) publicize (N, y) as her public key material, and keep (p, q) as her private key.

GM Encryption

Suppose that Bob wants to send a binary string $m = b_1b_2 \cdots b_\ell$ to Alice. Then:

```

for (  $i = 1, 2, \dots, \ell$  )
{
  Bob picks  $x \in_U \mathbb{Z}_N^*$ ;
  if ( $b_i == 0$ ) Bob sets  $c_i \leftarrow x^2 \pmod{N}$ 
  else Bob sets  $c_i \leftarrow yx^2 \pmod{N}$ 
}
Bob sends Alice:  $E_N(m) \leftarrow (c_1, c_2, \dots, c_\ell)$ .
```

GM Decryption

Suppose that Alice receives an ℓ -tuple ciphertext $(c_1, c_2, \dots, c_\ell)$. Then Alice performs:

```

for (  $i = 1, 2, \dots, \ell$  )
{
  if ( $c_i \in QR_N$ )  $b_i \leftarrow 0$ 
  else  $b_i \leftarrow 1$ ;
}
Alice sets  $m \leftarrow (b_1, b_2, \dots, b_\ell)$ .
```

Figure 13.3.

crypts the entire message bit by bit, where the difficulty for finding an encrypted single bit from a ciphertext c is that for deciding whether $c \in \text{QR}_N$ or $c \in \text{J}_N(1) \setminus \text{QR}_N$, where $\text{J}_N(1) = \{x \mid x \in \mathbb{Z}_N^*, \left(\frac{x}{N}\right) = 1\}$.

The GM cryptosystem is specified in Algorithm 13.1.

13.3.3.1 How does it work?

Observing the encryption algorithm it is easy to see that the plaintext bit 0 is encrypted to a ciphertext in QR_N .

For the plaintext bit 1, the corresponding ciphertext is $c = yx^2$. Noticing $\left(\frac{y}{p}\right) = \left(\frac{y}{q}\right) = -1$, we have (due to the multiplication property of Legendre symbol, see Theorem 6.16):

$$\left(\frac{c}{p}\right) = \left(\frac{yx^2}{p}\right) = \left(\frac{y}{p}\right) \left(\frac{x^2}{p}\right) = (-1) \times 1 = -1$$

and

$$\left(\frac{c}{q}\right) = \left(\frac{yx^2}{q}\right) = \left(\frac{y}{q}\right) \left(\frac{x^2}{q}\right) = (-1) \times 1 = -1$$

and therefore

$$\left(\frac{c}{N}\right) = \left(\frac{c}{p}\right) \left(\frac{c}{q}\right) = (-1) \times (-1) = 1.$$

That is, the plaintext 1 is encrypted to a ciphertext in $\text{J}_N(1) \setminus \text{QR}_N$.

The decryption algorithm works properly because knowing p, q , Alice can decide whether $c_i \in \text{QR}_N$ or $c_i \in \text{J}_N(1) \setminus \text{QR}_N$, respectively, and hence can retrieve the plaintext bit by bit correctly.

It is not difficult to see that encryption of an ℓ -bit message b takes $O_B(\ell(\log_2 N)^2)$ bit operations; this is the time complexity for encryption. The encryption algorithm has a message expansion ratio of $\log_2 N$: one bit of plaintext is expanded into $\log_2 N$ bits of ciphertext.

Since computing Legendre symbol mod p and mod q with $|p| = |q| = k$ can be done in $O_B(k^2)$ bit operations (review the discussion after Algorithm 6.2 on careful realization of Jacobi-symbol algorithm), decryption of $(c_1, c_2, \dots, c_\ell)$ requires $O_B(\ell(\log_2 N)^2)$ bit operations. This is the time complexity for decryption.

The “bit-by-bit” fashion of encryption means that the GM cryptosystem is highly inefficient.

13.3.4 The Security of the GM Cryptosystem

Somewhat like a \mathcal{ZPP} algorithm, the encryption algorithm of the GM cryptosystem is an error-free randomized algorithm: the random operations in the encryption

algorithm can introduce no any error into the ciphertext but achieve the following important function:

Distributing the plaintext bit 0 uniformly (that is, correctly) over QR_N and the plaintext bit 1 uniformly over $J_N(1) \setminus \text{QR}_N$.

The both distributions are uniform. This is because, for the plaintext bit 0, squaring maps from \mathbb{Z}_N^* onto QR_N , and for the plaintext bit 1, multiplying- y to an element in QR_N is a permutation from QR_N onto $J_N(1) \setminus \text{QR}_N$. Thus, picking $x \in_U \mathbb{Z}_N^*$ in the encryption algorithm means picking either a uniform element in QR_N if the plaintext bit is 0, or a uniform element in $J_N(1) \setminus \text{QR}_N$ if the plaintext bit is 1.

To express it formally, we say that the difficulty of the GM cryptosystem is that for deciding the quadratic residuosity problem (QRP) which is formally specified in Definition 6.2. The QRP is a well-known hard problem in number theory (review the discussion we provided after Definition 6.2). We have the following assumption on its intractability.

Assumption 13.1: (Quadratic Residuosity Assumption, QRA) *Let \mathcal{IG} be an integer instance generator that on input 1^k , runs in time polynomial in k , and outputs a $2k$ -bit modulus $N = pq$ where p and q are each a k -bit uniformly random odd prime.*

We say that \mathcal{IG} satisfies the quadratic residuosity assumption (QRA) if, for $N \leftarrow \mathcal{IG}(1^k)$, ensembles QR_N and $J_N(1) \setminus \text{QR}_N$ are polynomially indistinguishable where polynomial indistinguishability follows Definition 4.14.

It is clear that the availability of the public key N places an upperbound for the difficulty of the QRP since it suffices for an attacker to factor N and then apply the GM decryption algorithm to solve the QRP. Therefore, the GM cryptosystem assumes the attacker to be polynomially bounded. That is why semantic security for encryption algorithms is also called **polynomial indistinguishability of encryptions**.

If the QRA truly holds, then we can consider that, viewed by a polynomially bounded attacker, the GM encryption algorithm distributes a plaintext bit uniformly over the ciphertext space $J_N(1)$. The uniform distribution of the ciphertext means that an attempt for such an attacker to guess the plaintext from the corresponding ciphertext is a completely senseless thing to do. This is exactly what Goldwasser and Micali mean by expressing their notion of semantic security in the form of Property 13.1.

We can re-state the notion of semantic security as follows.

Definition 13.1: (Semantic Security, Secure with Respect to the Indistinguishable Chosen Plaintext Attack, IND-CPA Security) *A cryptosystem*

with a security parameter k is said to be semantically secure (IND-CPA secure) if after the attack game in Protocol 13.1 being played with any polynomially bounded attacker, the advantage Adv formulated in (13.2.3) is a negligible quantity in k .

We have the following result for the security of the GM cryptosystem.

Theorem 13.1: *Let k be the size of the two prime factors of an RSA modulus N . The GM cryptosystem with security parameter k is semantically secure (IND-CPA secure) if and only if the QRA holds.* \square

13.3.5 A Semantically Secure Version of the ElGamal Cryptosystem

Similar to the case of the RSA cryptosystem, the ElGamal cryptosystem specified in Algorithm 8.3 does not hide the quadratic residuosity of the plaintext. This is because in that algorithm we have set the public parameters (g, p) such that g generates the whole group \mathbb{Z}_p^* . In such a parameter setting, the quadratic residuosity of a plaintext can be related to that of the corresponding ciphertext. This is shown in the following example.

Example 13.1: Let the oracle \mathcal{O} setup (p, g, y) as the public key material for the ElGamal cryptosystem specified in Algorithm 8.3. Then due to Euler's criterion (Theorem 6.13), $g \in \text{QNR}_p$ (i.e., g is a quadratic non-residue modulo p).

Let Malice be an IND-CPA attacker. He should submit a message $m_0 \in \text{QR}_p$ and $m_1 \in \text{QNR}_p$ (applying Algorithm 6.2, it is easy for Malice to prepare m_0 and m_1 to satisfy these two conditions). Let (c_1^*, c_2^*) be the pair of challenge ciphertext returned from \mathcal{O} ; we have

$$c_2^* = \begin{cases} y^k m_0 \pmod{p} & 50\% \text{ probability,} \\ y^k m_1 \pmod{p} & 50\% \text{ probability.} \end{cases}$$

Now, Malice can pinpoint the plaintext by deciding the quadratic residuosity of y , c_1^* and c_2^* . There are a few cases to consider.

Let us first consider $y \in \text{QR}_p$. This case is very easy. The plaintext is m_0 if and only if $c_2^* \in \text{QR}_p$. This is due to the multiplicative property of the Legendre symbol given in Theorem 6.16.(ii).

The case $y \in \text{QNR}_p$ has two sub-cases which are also very easy. The first sub-case of $c_1^* \in \text{QR}_p$ will cause $y^k \in \text{QR}_p$ (because now k is even), and thereby the decision is identical to that in the previous paragraph. The reader may complete the second sub-case of $c_1^* \in \text{QNR}_p$ by noticing that now k is odd. \square

As usual, having seen where the problem is, it is relatively easy to fix it. If we restrict the cryptosystem to working in QR_p , then the attack in Example 13.1 will no longer work. Algorithm 13.2 specifies a fix.

Algorithm 13.2: A Fix of the ElGamal Cryptosystem**Public Parameter Setup**

Let G be an abelian group with the following description:

1. find a random prime number q with $|q| = k$;
2. test primality of $p = 2q + 1$, if p is not prime, go to 1;
3. pick a random generator $h \in \mathbb{Z}_p^*$; set $g = h^2 \pmod{p}$;
4. let $\text{desc}(G)$ be such that $G = \langle g \rangle$;
 (* The group generated from g , see Definition 5.10. *)
5. let (p, g) be the public parameters for the ElGamal cryptosystem;
6. let G be the plaintext message space.

(* The rest part is the same as that of Algorithm 8.3 *)

First of all we should notice that Algorithm 13.2 will terminate because there are plenty of prime numbers p such that $(p - 1)/2$ is also prime (7, 11, 23, 39, 47 are several examples). Such a prime is called a **safe prime**.

Next, by the Fermat's little theorem, $\text{ord}_p(g) = q$ which is a large prime; therefore, the group $G = \langle g \rangle$ has a large order. This is a necessary requirement for the DLA to hold (Assumption 8.2).

Moreover, by the Euler's Criterion (Theorem 6.13) we know $g \in \text{QR}_p$ and therefore $G = \text{QR}_p$ (the reader may answer why by noticing Theorem 5.2). So for chosen plaintexts $m_0, m_1 \in \text{QR}_p$, the numbers g, y, c_1^*, c_2^* are all quadratic residues modulo p . Consequently, the quadratic-residuosity attack demonstrated in Example 13.1 will no longer work since now all cases of quadratic residuosity testing will output the YES answer.

The stipulation of the plaintext space being $G = \text{QR}_p$ can cause no trouble for message encoding (in encryption time) and decoding (in decryption time). For example, for any message $m < p$, if $m \in \text{QR}_p$, then we have done; if $m \notin \text{QR}_p$, then $-m = p - m \in G$. This is because from

$$(-1)^{(p-1)/2} = (-1)^q = (-1)^{\text{odd number}} = -1 \pmod{p},$$

we have

$$(-m)^{(p-1)/2} = (-1)^{(p-1)/2} m^{(p-1)/2} = (-1)(-1) = 1 \pmod{p}$$

and therefore $-m \in \text{QR}_p = G$ follows from the Euler's Criterion.

For the fixed version of the ElGamal cryptosystem, Malice now faces a different decisional problem. Upon receipt of the challenge ciphertext (c_1^*, c_2^*) after having submitted m_0, m_1 for encryption, he can compute from c_2^* :

$$c_2^*/m_0 = \begin{cases} y^k \equiv g^{xk} \pmod{p} & 50\% \text{ probability,} \\ y^k(m_1/m_0) \pmod{p} & 50\% \text{ probability.} \end{cases}$$

Notice that in the first case, the tuple

$$(g, y, c_1^*, c_2^*/m_0) = (g, g^x, g^k, g^{xk}) \pmod{p}$$

is a Diffie-Hellman tuple, while in the second case it is not. So Malice should ask himself:

$$\text{Is } (g, y, c_1^*, c_2^*/m_0) \pmod{p} \text{ a DH tuple?}$$

or

$$\text{Is } (g, y, c_1^*, c_2^*/m_1) \pmod{p} \text{ a DH tuple?}$$

That is, the IND-CPA game actually challenges Malice to answer the DDHP in G (see Definition 12.1).

If Malice can answer the DDHP in G correctly, then given the challenge ciphertext pair he can of course pinpoint the plaintext, i.e., the coin tossing of \mathcal{O} , correctly. Conversely, because $(g, y, c_1^*, c_2^*/m_0) \pmod{p}$ and $(g, y, c_1^*, c_2^*/m_1) \pmod{p}$ are random tuples generated by g , so if he can pinpoint the plaintext correctly then he can answer the DDHP in $G = \langle g \rangle$ correctly. So IND-CPA security for the ElGamal cryptosystem using the public parameters in Algorithm 13.2 is precisely the difficulty for answering the DDHP in G (Theorem 13.2).

In the *general case* of abelian groups (which includes G defined in Algorithm 13.2) we do not know any efficient algorithm to answer the DDHP. The difficulty has rendered the DDHP a standard and widely accepted intractability. The reader is referred to Boneh's survey article [Bon97] for further study of this problem.

Assumption 13.2: (Decisional Diffie-Hellman Assumption, DDHA) *Let \mathcal{IG} be a group instance generator that on input 1^k , runs in time polynomial in k , and outputs (i) $\text{desc}(G)$ (the description of an abelian group G) with $|\#G| = k$, (ii) a group generator $g \in G$.*

We say that \mathcal{IG} satisfies the decisional Diffie-Hellman assumption (DDHA) if for $(\text{desc}(G), g) \leftarrow \mathcal{IG}(1^k)$, ensembles (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are polynomially indistinguishable where polynomial indistinguishability follows Definition 4.14.

We have established the following result for the fixed version of the ElGamal cryptosystem.

Theorem 13.2: *The ElGamal cryptosystem using the public parameters in Algorithm 13.2 is IND-CPA secure if and only if the DDHA holds.* \square

13.3.6 Semantically Secure Cryptosystems Based on Rabin Bits

After the work of Goldwasser and Micali in cryptosystems with semantic security, several public-key encryption schemes with semantic security and form improvements the GM cryptosystem have been proposed by several authors. These include, Blum and Micali [BM82], Yao [Yao82] and an efficient scheme by Blum and Goldwasser [BG85].

The main idea in these improvements is the notion of a CSPRB generator (see the previous subsection). Such a generator is a program which takes as input an k -bit random seed and produces as output a k^t -bit number, where $t > 1$ is fixed. The output produced by a CSPRB generator is of a high quality in the following sense: if the k -bit seed is totally unknown, then the output k^t -bit number cannot be distinguished from truly random number of the same length by any statistical test which runs in polynomial in k time.

Now, to encrypt an ℓ -bit message m , the sender sends the exclusive-or of m with an ℓ -bit output string pr of a CSPRB generator on a k -bit input seed s along with a public-key encryption of s , that is

$$(c_1, c_2) = (\mathcal{E}_{pk}(s), m \oplus ps). \quad (13.3.2)$$

The legitimate recipient (i.e., the owner of the public key pk) can decrypt c_1 and obtain the seed s . This will enable the recipient to re-generate the ℓ -bit pseudo-random bit string ps from the CSPRB generator, and thereby to retrieve m from c_2 by the exclusive-or operation.

A CSPRB-generator-based encryption scheme has a much improved efficiency from the “bit-by-bit” fashion of encryption. An ℓ -bit plaintext message is now expanded to an $(\ell + k)$ -bit ciphertext message instead of ℓk -bit as in the case of “bit-by-bit” fashion of encryption. The improved time and space complexities are compatible to those of the block-based encryption algorithms such as RSA, Rabin and ElGamal.

13.3.6.1 Semantic security of a CSPRB based encryption scheme

If the seed s is a uniformly random k -bit string and if the block-based deterministic encryption algorithm \mathcal{E}_{pk} (with security parameter k) forms a permutation over

its message space^a, then the first ciphertext block c_1 in (13.3.2) is permuted from a uniformly random number and hence itself is uniformly random. Thus, it can provide no *a-priori* nor *a-posteriori* information about the plaintext for an attacker to exploit. Further because of the strength of the CSPRB generator, the pseudo-random string ps generated from it using the seed s plays the role of the internal random operation of the encryption scheme. Consequently, the exclusive-or between m and ps provides a semantically secure encryption of m .

In the case of the efficient CSPRB-generator-based encryption scheme by Blum and Goldwasser (the **BG cryptosystem**, [BG85]), c_1 in (13.3.2) is $s^{2^i} \pmod N$ where $i = \left\lfloor \frac{\log_2 m}{\log_2 \log_2 N} \right\rfloor + 1$; and the pseudo-random bit string ps is generated from s using the BBS pseudo-random generator (8.8.1) in a block-by-block fashion: each block is the $\log_2 \log_2 N$ least significant bits of an element which is the 2^j -th power of s modulo N ($j = 1, 2, \dots, i - 1$). Notice that the first element in the ciphertext pair is essentially a Rabin encryption of s .

Since the problem of extracting the simultaneous $\log_2 \log_2 N$ least significant bits of a plaintext from a Rabin ciphertext is equivalent to factoring N (review Remark 8.2), the semantic security of the BG cryptosystem can be quantified to being equivalent to factoring the modulus N .

13.4 Inadequacy of Semantic Security

The notion of the IND-CPA security (semantic security) introduced in Definition 13.1 (and in Property 13.1) captures the intuition that any polynomially bounded attacker should not be able to obtain any *a-priori* information about a plaintext message given its encryption. However, this guarantee of plaintext secrecy is only valid when the attacker is passive when facing a ciphertext, i.e., all the attacker does about a ciphertext is eavesdropping.

In §8.3.1 and §?? we have pointed out that many public-key cryptosystems are particularly vulnerable to a so-called **chosen-ciphertext attack** (CCA and CCA2, see Definition 8.3). In CCA and CCA2, an attacker (now he is Malice) may get decryption assistance, that is, he may be in certain level of control of a “decryption box” and so may have some ciphertext of his choice to be decrypted for him even though he does not have in possession of the decryption key. We have treated such an assistance as a “cryptanalysis training course” provided to Malice in order to ease his attack job. These modes of attacks, in particular CCA2, are realistic in many applications of public-key cryptography. For example, some protocols may require a principal to perform decryption operation on a random challenge to form a challenge-response mechanism. For another example, a receiver of an encrypted

^aThe RSA and the ElGamal encryption algorithms trivially form permutations over their message spaces; the Rabin encryption algorithm can be constructed to be a permutation in QR_N if N is a Blum integer, see Theorem 6.18(iv).

email may reveal the plaintext message in subsequent public discussions.

The particular vulnerability to CCA or CCA2 shared by many public-key cryptosystems is due to the generally nice algebraic properties that underlie these cryptosystems. Malice may explore these nice properties and make up a ciphertext via some clever calculations. If Malice is given a decryption assistance, his clever calculations on the chosen ciphertext based on the nice algebraic properties of the target public-key cryptosystem may allow him to obtain messages which should otherwise not be available to him.

In Example 8.9 we have seen a vulnerability of the ElGamal cryptosystem to CCA2. That attack is obviously applicable to the fixed version of the cryptosystem with IND-CPA security, too. The same-style CCA2 attacks are obviously applicable to any IND-CPA secure scheme based on a CSPRB generator (in §13.3.6); in such attacks, c_2 in (13.3.2) is replaced with $c'_2 = r \oplus c_2$ where r is an ℓ -bit random string and plays the same (blinding) role of the random number r in Example 8.9.

The following example shows the vulnerability of the GM cryptosystem to CCA2.

Example 13.2: Let Malice be in a conditional control of Alice's GM decryption box. The condition is quite "reasonable": if the decryption result of a ciphertext submitted by Malice looks random, then Alice should return the plaintext to Malice.

Let ciphertext $C = (c_1, c_2, \dots, c_\ell)$ encrypts plaintext $B = (b_1, b_2, \dots, b_\ell)$ which is from a confidential communication between Alice and someone else (not with Malice!). However, Malice has eavesdropped C and he wants to know B . He now sends to Alice the following "cleverly calculated ciphertext":

$$C' = (yc_1, yc_2, \dots, yc_\ell) \pmod{N}. \quad (13.4.1)$$

In this attack, Malice is making use of the following nice algebraic property:

$$ab \pmod{N} \in \text{QR}_N \text{ iff } \begin{cases} a \in \text{QR}_N & \text{and } b \in \text{QR}_N \\ a \in \text{J}_N(1) \setminus \text{QR}_N & \text{and } b \in \text{J}_N(1) \setminus \text{QR}_N. \end{cases}$$

This property is a direct result of Euler's criterion (see Theorem 6.13).

Thus, because $y \in \text{J}_N(1) \setminus \text{QR}_N$, we can view y to encrypt the bit 1. Then the "multiplying- y " attack in (13.4.1) causes flipping of the bit b_i for $i = 1, 2, \dots, \ell$, that is, the decryption result by Alice will be

$$B' = (b'_1, b'_2, \dots, b'_\ell),$$

where b'_i denotes the complementary of the bit b_i for $i = 1, 2, \dots, \ell$.

This decryption result should look random for Alice. So Alice returns B' back to Malice. Alas, Malice finds B !

Malice can also make B' *uniformly* random (not just "looks random") by using the "multiplier" $Y = (y_1, y_2, \dots, y_\ell)$ instead of (y, y, \dots, y) , where Y is a

GM encryption, under Alice's public key, of a uniformly random ℓ bit-tuple $Z = (z_1, z_2, \dots, z_\ell) \in_U \{0, 1\}^\ell$. It is easy to check

$$B = (b'_1 \oplus z_1, b'_2 \oplus z_2, \dots, b'_\ell \oplus z_\ell). \quad \square$$

In this attack, Alice has provided Malice with an “oracle service” for decryption assistance. Notice that the oracle service need not be an explicit one. The following example shows that without replying Malice's cipher query need not necessarily be a good strategy.

Example 13.3: Suppose that now Alice will no longer return random-looking decryption result back to Malice. For the encrypted message $(c_1, c_2, \dots, c_\ell)$ (e.g., sent from Bob to Alice), Malice can still find the plaintext bit by bit.

For instance, in order to find whether c_1 encrypts 0 or 1, Malice can send to Alice an encrypted question for her to answer (e.g., a question for an YES/NO answer). Malice can encrypt the first half of the question in the usual way, but encrypts the second half of the question using c_1 in place of y in Algorithm 13.1.

If $c_1 \in \text{QR}_N$, then Alice will only decrypt the first half of the question correctly. The decryption of the rest of the question will be all zeros. So she will ask Malice why he only sends an uncompleted sentence. Then Malice knows c_1 encrypts 0. On the other hand, if Alice can answer the question correctly, then Malice knows that c_1 is a non-residue and hence encrypts 1.

Notice that in this way of attack, Malice can even digitally sign all of his messages to Alice to assure her the full and real authorship of these messages. Malice cannot be accused of any wrong doing! \square

From these two ways of active attacks we realize that the GM cryptosystem is hopelessly weak against an active attacker. In fact, the security notion in IND-CPA is hopelessly weak.

13.5 Beyond Semantic Security

Lifting security notion from the “all-or-nothing” secure (Property 8.2) to IND-CPA secure (Definition 13.1) forms a first step in our process of strengthening security notions.

In §13.4 we have seen that a security notion in the IND-CPA sense is not good enough for applications where a user may be tricked to provide an oracle service in the decryption mode. Indeed, in applications of cryptographic systems, it will be impractical to require an innocent user to be always vigilant not to provide an oracle service in the decryption mode. Therefore, stronger security notions are needed.

The next step in our process of strengthening security notions is to consider an attack model called **indistinguishable chosen ciphertext attack (IND-CCA)**. In this attack model, we will further ease the difficulty for Malice to break the target cryptosystems: in addition to the encryption assistance provided in the CPA game (in Protocol 13.1), we will further allow Malice to obtain a conditional assistance in the decryption mode. A formal treatment of the IND-CCA model is based on a game due to Naor and Yung [NY90]. The game is named “**lunchtime attack**” or “**indifferent chosen ciphertext attack**”.

13.5.1 Security Against Chosen Ciphertext Attack

A lunchtime attack describes a real-life scenario of Malice who, in the absence of other employees of an organization (e.g., during a lunchtime), queries the decryption mechanism of the organization, in a hope that the interactions with the decryption box may provide him with a kind of “cryptanalysis training course” which may make him more experienced in a future cryptanalysis of the organization’s cryptosystem. Due to the short duration of lunchtime, Malice does not have enough time to prepare his ciphertext queries to be any function of the answers of the decryption box. Therefore, all ciphers he queries during lunchtime are ones which he had prepared before lunchtime.

This real-life scenario can also be modelled by a game of attack. The game will be played by the same players in the CPA attack game (Protocol 13.1): Malice who may be a disgruntled employee of an organization, and an oracle \mathcal{O} who is now the decryption (and encryption) mechanism of the organization. We shall name the game a **chosen ciphertext attack (CCA)**. The new game is specified in Protocol 13.3.

At a first glance, one may argue that lunchtime attack does not model a realistic attack scenario. Who will be so nice and so naive to play the role of a decryption box and answer Malice’s decryption queries? We should answer this question in three different considerations.

- In many applications of cryptosystems (in particular, in cryptographic protocols), it is often the case that a user (a participant of a protocol) is required, upon receipt of a challenge message, to perform a decryption operation using her private key and then send the decryption result back. This is the so-called challenge-response mechanism (see Chapter 2 and Chapter 10).
- We may have to accept a fact of life: many users are just hopelessly naive and cannot be demanded or educated to maintain a high degree of vigilance in anticipation of any trick launched by bad guys. In fact, it will not be wrong for us to say that stronger cryptosystems and security notions are developed exactly for naive users.
- Malice can even embed decryption queries inside normal and innocent looking

Protocol 13.3: “Lunchtime Attack” (Chosen Ciphertext Attack)

PREMISE

- i) As in Protocol 13.1, Malice and oracle \mathcal{O} have agreed on a target cryptosystem \mathcal{E} for which \mathcal{O} has fixed an encryption key;
- ii) Malice has prepared some ciphertext messages, before lunchtime.

1. Malice sends to \mathcal{O} a prepared ciphertext message $c \in \mathcal{C}$;

2. \mathcal{O} decrypts c and returns the decryption result back to Malice;

(* The ciphertext c is called a **chosen ciphertext** or an **indifferent chosen ciphertext**. It is considered that to return the decryption result back to Malice is to provide him with a “cryptanalysis training course”. Malice can ask for this “training course” to be repeated as many times as he wishes. He may want to consider to use a program to speed up the “training sessions” since lunchtime is short. *)

3. Upon satisfaction of the “decryption training course”, Malice now asks \mathcal{O} to play the CPA game in Protocol 13.1;

(* In this instantiation of the CPA game, the chosen plaintext messages m_0 and m_1 are called **adaptively chosen plaintext** messages. That is, these two messages can be some functions of the entire history of the “decryption training course” provided in Steps 1 and 2. Therefore, Malice’s “find-stage” starts right at the beginning of this protocol and ends upon his receipt of the challenge ciphertext $c^* \in \mathcal{C}$, which encrypts, equally likely, one of his two chosen plaintext messages $m_0, m_1 \in \mathcal{M}$. *)

(* We reasonably assume that Malice can compute adaptively chosen plaintext messages even in the short lunchtime since working on plaintext should be relatively easier than working on ciphertext. *)

(* By now, “lunchtime” is over. So Malice should answer either 0 or 1 as his educated guess on \mathcal{O} ’s coin tossing in the CPA game. However, even the game is over, Malice remains in “guess-stage” until he answers. *)

Figure 13.4.

communications, and in so doing he may get implicit answers to his queries. Examples 8.11 and 13.3 provide vivid active attacks which are very innocent looking. It can be very difficult to differentiate such attacks and legitimate secure communications. Not to answer any questions (encrypted questions or answers) does not constitute a good solution to active attacks but a self-denial from the benefit of the secure communication technology.

The correct attitude toward Malice is to face him straightly and provide him the “cryptanalysis training course” on his demand. The training course can be in encryption or in decryption, at a whole data-block level or at a single bit level. Our strategy is to design strong cryptosystems such that the “cryptanalysis training course” even supplied on demand won’t help Malice to break a target cryptosystem!

Following the same reasoning in §13.2 for deriving Malice’s advantage for breaking the target cryptosystem in the CPA game (Protocol 13.1), we can analogously derive Malice’s advantage in the lunchtime-attack game. The formulation of the advantage is very similar to (13.2.3), except that we should now add the entire history of the chosen ciphertext cryptanalysis training course to the input of Malice. Let Hist-CCA denote this history. Malice’s advantage is the following

$$\text{Prob}[1 \leftarrow \text{Malice}(c^*, m_0, m_1, \text{Hist-CCA}) \mid c^* = \mathcal{E}_{ke}(m_1)] = \frac{1}{2} + \text{Adv.} \quad (13.5.1)$$

To this end we reach a new security notion which is strengthened from the IND-CPA security notion.

Definition 13.2: (Secure with Respect to Indistinguishable Chosen Ciphertext Attack, IND-CCA Secure) *A cryptosystem with a security parameter k is said to be secure with respect to an indistinguishable chosen ciphertext attack (IND-CCA secure) if after the attack game in Protocol 13.3 being played with any polynomially bounded attacker, the advantage Adv formulated in (13.5.1) is a negligible quantity in k .*

Since in lunchtime-attack, the decryption assistance (or, “cryptanalysis training course”) for Malice is provided on top of the CPA game in Protocol 13.1, the new attack game must have reduced the difficulty for Malice’s cryptanalysis task from that of the CPA game. We should therefore expect that some cryptosystems which are IND-CPA secure will no longer be IND-CCA secure.

None of the IND-CPA secure cryptosystems which we have introduced in this chapter has been proven to be IND-CCA secure. Alas, among them, there is a demonstrably *insecure* one! This is the efficient CSPRB-generator-based cryptosystem of Blum and Goldwasser, which we have introduced in §13.3.6.

Example 13.4: To attack the BG cryptosystem in lunchtime attack, Malice should make a chosen-ciphertext query (c, m) where c is a chosen quadratic residue modulo

N , and $|m| = \lfloor \log_2 \log_2 N \rfloor$. Observing the BG cryptosystem described in §13.3.6, we know that Malice will be responded with the following decryption result

$$m \oplus \text{“}\lfloor \log_2 \log_2 N \rfloor \text{ least significant bits of a square root of } c \text{ modulo } N\text{”}.$$

Bit-wise XOR-ing m to the reply, Malice obtains $\lfloor \log_2 \log_2 N \rfloor$ least significant bits of a square root modulo N of c . Remember that c is a chosen quadratic residue modulo N . Review Remark 8.2, providing Malice with $\lfloor \log_2 \log_2 N \rfloor$ least significant bits of a square root of c will entitle him to factor N in probabilistic polynomial-time! \square

Example 13.4 shows that it is indeed the very cryptanalysis training course provided to Malice that empowers him to break the cryptosystem. The break through is so severe and thorough that, it is not a mere disclosure of one ciphertext, but the total destroy of the cryptosystem.

We also realize that the precise basis for the BG cryptosystem to be IND-CPA secure is also the very exact cause for the cryptosystem to be insecure with respect to IND-CCA. This is analogous to the case for the Rabin cryptosystem under the “all-or-nothing” sense of security (see Theorem 8.2).

Naor and Yung proposed a cryptosystem which is provably secure with respect to IND-CCA [NY90]. In that cryptosystem, a plaintext message is encrypted in a bit-by-bit fashion into two ciphertext messages under two different public keys. The encryption algorithm includes a **non-interactive zero-knowledge** (NIZK) proof procedure which enables the sender of a plaintext message to prove that the two ciphertext messages do encrypt the same plaintext bit under the respective public keys (consider the encryption algorithm forms an NP problem with the plaintext and random input to the algorithm as the witness to the NP problem, see discussion in §4.11.1). This proof will be verified in the decryption time (e.g., by \mathcal{O} in the lunchtime-attack game). Passing of the verification procedure in the decryption time implies that the plaintext encrypted under the pair of ciphertext messages is already known to the sender (e.g., known to Malice in the lunchtime-attack game). So serving Malice in “lunchtime” will not provide him with any new knowledge for easing his cryptanalysis job.

Due to a rather high cost of realizing a NIZK proof (verification) for an encryption (decryption) algorithm and the bit-by-bit fashion of encryption and decryption, the cryptosystem of Naor and Yung [NY90] is not intended for practical uses. In a later chapter, we will study the topic of zero-knowledge proof (including NIZK). There we will study a practically efficient zero-knowledge proof protocol which provides a so-called “undeniable signature” scheme. Such a scheme provides a signer with a desired privacy. We will prove that that undeniable signature scheme is precisely IND-CCA secure in terms of indistinguishability in who the signer is. The precise security means that the scheme is IND-CCA secure but *not stronger*.

Lunchtime attack is a quite restrictive attack model in that, the decryption assistance provided to Malice is only available in a short period of time. It is as if

the decryption box would be switched off permanently after “lunchtime”, or Malice would not strike back anymore, not even in “lunchtime” tomorrow. This is not a reasonable or realistic scenario. In reality, naive users will remain permanently naive, and Malice will definitely strike back, probably even as soon as in the afternoon tea-break time! Therefore the security notion in IND-CCA is, again, not strong enough. We need a still stronger security notion.

13.5.2 Security Against Adaptively Chosen Ciphertext Attack

A further step in our process of strengthening security notions is to consider an attack model called **indistinguishable adaptively chosen ciphertext attack (IND-CCA2)**. Rackoff and Simon originally proposed this stronger attack model [RS92].

In this attack model, we will further ease the difficulty for Malice to attack cryptosystems from that in lunchtime attack. In the lunchtime-attack game (see Protocol 13.3), the decryption assistance (or “cryptanalysis training course”) is conditional in that, the assistance will be stopped upon Malice’s submission of the pair of the adaptively chosen plaintext messages; that is, a lunchtime attack stops upon termination of the CPA game (i.e., Protocol 13.1), and from then on the decryption assistance will become permanently unavailable.

In the new attack model we remove this unrealistic condition of a short-period availability of decryption assistance which is somewhat artificially placed in lunchtime attack. Now the decryption assistance for Malice will be permanently available before and after a lunchtime attack. We can imagine this attack scenario as a prolonged lunchtime attack. For this reason, we shall name this new attack model **small-hours attack**. This name describes a real-life scenario that Malice, again as a disgruntle employee of an organization, stays up whole night to play with the decryption mechanism of the organization. Notice that small-hours attack is different from a so-called **midnight attack** which often appears in the literature as another name for lunchtime attack; perhaps as in lunchtime attack, it is considered that the security guards of an organization should have meals rather punctually around midnight.

Since now Malice has plenty of time unnoticed, he will of course play the decryption box in a more sophisticated and more interesting way. In addition to what he can do in lunchtime (in fact at midnight), i.e., adaptively chooses plaintext queries using information gathered from the “decryption training course” and subsequently obtains a corresponding challenge ciphertext, now Malice can also submit *adaptively chosen ciphertext messages after* he receives the challenge ciphertext. Therefore, the adaptively chosen ciphertext messages can somehow be related to the challenge ciphertext of which the corresponding plaintext is chosen by him. Of course, the decryption box is intelligent enough and not to decrypt the exact challenge ciphertext for Malice! This is the only restriction, and is of course a reasonable one.

Without this restriction, Malice can simply ask the decryption box to decrypt the challenge ciphertext for him, and we will not have an interesting game to play! The decryption box is also dummy enough so that it will decrypt a ciphertext which can be related to the the challenge ciphertext in any straightforward way! Any minute change of the challenge ciphertext, such as multiplying 2, or adding 1, will guarantee a decryption service!

Our description on the new attack model is specified in Protocol 13.4.

Protocol 13.4: “Small-hours Attack” (Adaptively Chosen Ciphertext Attack)

PREMISE

As in Protocol 13.1, Malice and oracle \mathcal{O} have agreed on a target cryptosystem \mathcal{E} for which \mathcal{O} has fixed an encryption key.

1. Malice and \mathcal{O} play the lunchtime-attack game in Protocol 13.3;
 (* In this instantiation of the lunchtime-attack game Malice’s “find-stage” is the same as that in Protocol 13.3, which ends upon his receipt of the challenge ciphertext $c^* \in \mathcal{C}$, which encrypts, equally likely, one of his two chosen plaintext messages $m_0, m_1 \in \mathcal{M}$. However, in this instantiation, Malice is allowed to extend his “guess-stage”. The extended “guess-stage” is as follows. *)
2. Malice further computes ciphertext $c' \in \mathcal{C}$ and submits it to \mathcal{O} for decryption;
 (* The ciphertext c' is called an **adaptively chosen ciphertext** or **post-challenge chosen ciphertext**. In contrast, the chosen ciphertext in lunchtime-attack game (Protocol 13.3) is also called **pre-challenge chosen ciphertext**. Step 2 is considered to serve Malice a “decryption training course” *extended* from that of the lunchtime-attack game. Malice can ask for the “extended training course” to be repeated as many times as he wishes. *)
 (* It is stipulated that $c' \neq c^*$, namely, Malice is not allowed to send the challenge ciphertext back for decryption. *)
3. Upon satisfaction of the “extended decryption training course”, Malice must now answer either 0 or 1 as his educated guess on \mathcal{O} ’s coin tossing.

Figure 13.5.

Again, following the same reasoning in §13.2 for deriving Malice’s advantage

for breaking the target cryptosystem in the CPA game (Protocol 13.1), we can analogously derive Malice's advantage to break the target cryptosystem in the small-hours-attack game. The formulation of the advantage is again very similar to (13.2.3), except that we should now add to Malice's input the entire history of the two cryptanalysis training courses, one for the CCA, and one for the "extended CCA". Let Hist-CCA2 denote this whole history. Malice's advantage is the following

$$\text{Prob}[1 \leftarrow \text{Malice}(c^*, m_0, m_1, \text{Hist-CCA2}) \mid c^* = \mathcal{E}_{ke}(m_1)] = \frac{1}{2} + \text{Adv}. \quad (13.5.2)$$

To this end we reach a new security notion which is further strengthened from the IND-CCA security notion.

Definition 13.3: (Secure with Respect to Indistinguishable Adaptively Chosen Ciphertext Attack, IND-CCA2 Secure) *A cryptosystem with a security parameter k is said to be secure with respect to an indistinguishable adaptively chosen ciphertext attack (IND-CCA2 secure) if after the attack game in Protocol 13.4 being played with any polynomially bounded attacker, the advantage Adv formulated in (13.5.2) is a negligible quantity in k .*

We summarize the various IND attack games introduced so far in Figure 13.6.

Since in the small-hours-attack game, the "extended" decryption assistance (or, the "extended cryptanalysis training course") is provided after the lunchtime-attack game in Protocol 13.1, the new attack game must have further reduced the difficulty for Malice to break the target cryptosystem from that in lunchtime-attack. We should therefore expect that some cryptosystems which are IND-CCA secure will no longer be IND-CCA2 secure. In fact, except for the RSA-OAEP which we have specified in Algorithm 9.6, none of the other cryptosystems introduced so far in this book is provably IND-CCA2 secure. We have demonstrated plenty of examples of cryptosystems which are CCA2 insecure, i.e., in "all-or-nothing" sense, and hence they are also IND-CCA2 insecure (see Examples 8.5, 8.7, 8.9, 8.11, 13.2, 13.3).

After having introduced the notion of IND-CCA2 security, Rackoff and Simon proposed IND-CCA2 secure cryptosystems which are also based on NIZK proof. However, they considered the case of an NIZK proof with a *specific prover*. In their IND-CCA2 secure cryptosystems, not only the receiver has public-private key pair, the sender has such a key pair too. Moreover, the sender's public key is certified by a public-key certification infrastructure (see the techniques in §12.2). The sender will not only use the receiver's public key to encrypt a message as usual, but also use his own private key in the construction of a NIZK proof so that the receiver of the ciphertext can verify the proof using the sender's public key. Passing of the NIZK verification implies that it is the specific sender (prover) who has created the plaintext, and hence, returning the plaintext back to the sender will not provide him any information useful for breaking the target cryptosystem. The IND-CCA2 secure cryptosystems of Rackoff and Simon also operate in the bit-by-bit fashion.

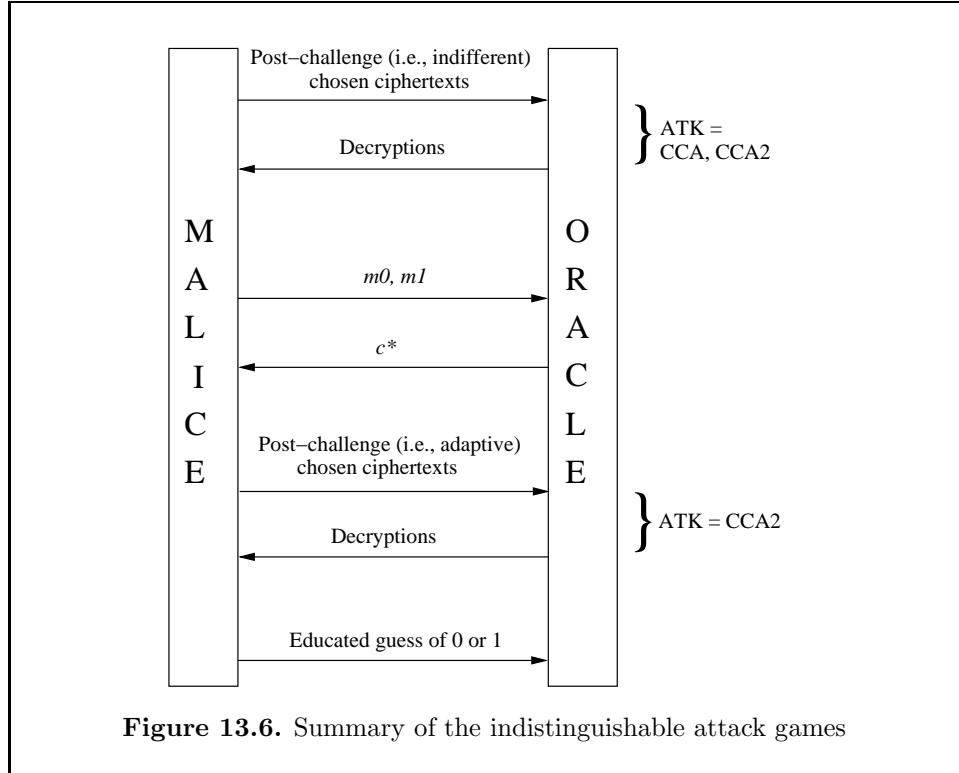


Figure 13.6. Summary of the indistinguishable attack games

13.5.3 Non-Malleable Cryptography

Non-malleable (NM) cryptography [DDN91] strengthens security notions for public-key cryptography in the computational direction. NM is an important requirement that it should not be easy for Malice to modify a plaintext message in a meaningfully controllable manner via modifying the corresponding ciphertext. Dolev et al described the importance of this requirement very well using a contract bidding example.

Suppose that constructing companies are invited by a municipal government to a bid for constructing a new elementary school. The government, which actively progresses its electronic operations, advertizes its public key E to be used for encrypting bids, and opens an email address `e-gov@bid.for.it.gov` for receiving the encrypted bids. Company A places its bid of \$1,500,000 by sending $\mathcal{E}(1,500,000)$ to the `e-gov@bid.for.it.gov`. However, the email is intercepted by Malice, the head of CheapSub, a one-man company specializing selling sub-contracts to some cheap builders. If the encryption algorithm used by the e-government is malleable, then Malice may somehow transform $\mathcal{E}(1,500,000)$ into $\mathcal{E}(15,000,000)$. In so doing, Malice's own bid would get a much better chance to win.

Unlike in the various cases of attacks on indistinguishability (IND) security where attack problems are decisional ones, a malleability attack is a computational problem. The problem is described in Protocol 13.5.

Protocol 13.5: Malleability attack in chosen plaintext mode

PREMISE

As in Protocol 13.1, Malice and oracle \mathcal{O} have agreed on a target cryptosystem \mathcal{E} for which \mathcal{O} has fixed an encryption key pk .

Malice and \mathcal{O} play the following game:

1. Malice sends to \mathcal{O} : $\mathbf{v}, \text{desc}(\mathbf{v})$, where \mathbf{v} is a vector containing a plural number of plaintexts, $\text{desc}(\mathbf{v})$ is a description on the distribution of the plaintexts in \mathbf{v} ;
2. \mathcal{O} creates a valid challenge ciphertext $c^* = \mathcal{E}_{pk}(\alpha)$ where α is created following the distribution of the plaintexts in \mathbf{v} ; \mathcal{O} sends c^* to Malice;
3. Upon receipt of c^* , Malice must output a “meaningful” PPT-computable relation R and another valid ciphertext $c' = \mathcal{E}_{pk}(\beta)$ such that $R(\alpha, \beta) = 1$ holds.

Figure 13.7.

In this malleability attack, because the goal of Malice, given a challenge ciphertext c^* , is not to learn something about the target plaintext α , he need not know α at al. However, for his attack to be successful, Malice must output a “meaningful” relation R to relate the decryptions of c^* and c' .

Malice’s success is also expressed in terms of an advantage. In [DDN91], the authors use the idea of zero-knowledge simulation^b to express this advantage. First, Malice, who is again a PPT algorithm, is given $c^* = \mathcal{E}_{pk}(\alpha)$ and outputs $(\mathcal{E}_{pk}(\beta), R)$ with certain probability. Secondly, a *simulator*, who we denote by ZK-Sim and is a PPT algorithm, is *not* given c^* but will also output a ciphertext \bar{c} with certain probability. (ZK-Sim even ignores the encryption algorithm and the public key!) Malice’s advantage in mounting a malleability attack is the following probability difference

$$\text{NM-Adv} = \text{Prob}[(\mathcal{E}_{pk}(\beta), R) \leftarrow \text{Malice}(\mathcal{E}_{pk}(\alpha), pk, \text{desc}(\mathbf{v}))] - \text{Prob}[(\bar{c}, R) \leftarrow \text{ZK-Sim}]. \quad (13.5.3)$$

^bIn a later chapter we shall study topics of zero-knowledge proof and polynomial-time simulation of such proofs.

The cryptosystem $\mathcal{E}_{pk}()$ with a security parameter k is said to be non-malleable if, for all PPT computable relation R and for all PPT attackers (i.e., Malice and the like), NM-Adv is a negligible function in k . In [DDN91], this security notion is called “**semantic security with respect to relations under chosen plaintext attack**”. We therefore name it **NM-CPA**. NM-CPA intuitively captures the following desirable security quality.

Property 13.2: (NM-CPA Security) *Given a ciphertext from an NM-CPA secure cryptosystem, Malice’s advantage to mount a malleability attack on the cryptosystem does not increase in any PPT discernible way from that to “mount the attack” (i.e., to simulate an attack) without the ciphertext.*

While providing a ciphertext should not ease an attack problem, providing “cryptanalysis training courses” should! Analogous to the cases of IND-CCA and IND-CCA2, a malleability attack can also be eased in the “lunchtime-attack” mode and in the “small-hours-attack” mode. In a malleability attack eased in the “lunchtime-attack” mode, Malice can send plural number of chosen ciphertexts to \mathcal{O} for decryption, but this service will terminate upon Malice requests to receive the challenge ciphertext c^* . In a malleability attack eased in the “small-hours-attack” mode, the decryption service does not terminate after the challenge ciphertext c^* has been sent to Malice. Of course, as we have stipulated in “small-hours” attack game, Malice is not allowed to send c^* back to \mathcal{O} for decryption service.

Consequently, we have **NM-CCA** and **NM-CCA2** security notions.

Due to these problems’ computational nature, we shall not include here a rigorously formal treatment for NM-security notions. The interested reader is referred to [DDN91] for details. It is quite reasonable to anticipate some added care to be in place in the formalization of NM notions. For example, unlike the decisional-problem cases where we do not need to take care of the size of a plaintext message space (there it can even be as small as 2, e.g., as in the GM cryptosystem), a formalization of NM notion must stipulate that the plaintext message space be sufficiently large so that the computation of the relation R will not degenerate into a trivial problem.

In [BDPR98], the authors provide slightly different formalizations for NM security notions which are based on attack games similar to those we have introduced for various IND attacks. The reader may find that the treatment in [BDPR98] to be easier to access as a result of their similarity to the games we have introduced for IND security notions.

Nevertheless, our description on NM security notions does suffice us to capture the idea of NM-security notions with adequate precision. Immediately we can see that, most textbook encryption algorithms which are results of direct applications of one-way trapdoor functions are easily malleable. As we have seen in §8.8, all common one-way (trapdoor) functions underlying popular public-key cryptography

can be inverted by making use of some partial information oracles (e.g., “parity oracle” or “half-order oracle”); the principle of these inverting methods is exactly malleability attacks mounted on the unknown plaintext messages. For example, for the RSA case of $c = m^e \pmod{N}$, Malice knows that the unknown plaintext m can be doubled if he multiply c with $2^e \pmod{N}$.

Dolev et al proposed a public-key encryption scheme which is provable NM-CCA2 secure [DDN91]. The scheme uses a plural number of public/private keys pairs, and encrypts a plaintext message in bit-by-bit manner. The encryption of each plaintext bit also contains an NIZK proof.

13.5.4 Relations Between Indistinguishability and Non-Malleability

The non-malleable security notions are undoubtedly very important. However, due to these problems’ computational nature, formal treatment for non-malleable security notions turns out being rather complex. Consequently, designing a cryptosystem and establishing that it has a non-malleable security is rather a difficult job.

Fortunately, researchers have established a number of important relations between non-malleable security notions and indistinguishable security notions. Moreover, under CCA2, the most useful security notion, non-malleability is found equivalent to indistinguishability. Since formal treatment for IND-CCA2 has been well established, we can achieve provable security in the NM-CCA2 mode by proving security under the IND-CCA2 notion.

Formal proof for relations between security notions can be achieved by constructing a **polynomial-time reduction algorithm**. In the context of relating attacks on cryptosystems, such a reduction (algorithm) “reduces” a target attacking problem (call it “Target Attack”) to another attack (call it “Source Attack”). If a successfully constructed reduction is a PPT algorithm, then “Target Attack” can be successfully mounted based on the successful mounting of “Source Attack”, and the cost for mounting “Target Attack” is bounded by a polynomial in that for mounting “Source Attack”.

Since an attack on a cryptosystem is always based on some appropriate assumptions and requirements (e.g., for an CCA2 attacker to work properly, the attacker should be entitled to pre-challenge and post-challenge cryptanalysis training courses), a reduction algorithm must satisfy an attacker these necessary assumptions and environmental requirements. Often we will use a special agent, who we name **Simon Simulator**, to conduct a reduction. Simon will satisfy an attacker all the assumptions and the entitled requirements by simulating the attacker’s working environment.

Sometimes, Simon himself will become a successful attacker for “Target Attack” as a result that he has been taught by the attacker for “Source Attack” after having interacted with the attacker. In such a reduction, Simon will be “orches-

trating” two attack games in between an attacker and an encryption/decryption oracle. On the one hand, Simon plays the “Source Attack” game with an attacker by simulating the environment for “Source Attack” (i.e., by masquerading as an encryption/decryption oracle to face the attacker). On the other hand, Simon plays the “Target Attack” game with an encryption/decryption oracle, and now he is an attacker. In such a situation, we can consider that the attacker for “Source Attack” is teaching Simon to mount “Target Attack”. Figures 13.8 and 13.9 illustrate such an orchestration conducted by Simon.

Now we are ready to state and prove some useful relations.

13.5.4.1 Non-malleability implying indistinguishability

Let ATK denotes CPA, CCA or CCA2. we can show that if a public-key cryptosystem is NM-ATK secure then it must be IND-ATK secure.

Theorem 13.3: *If a public-key cryptosystem is NM-ATK secure then it is also IND-ATK secure,*

Proof We can prove the theorem by showing that if a public-key cryptosystem \mathcal{E}_{pk} is IND-ATK insecure then it must be NM-ATK insecure.

Suppose \mathcal{E}_{pk} is IND-ATK insecure. Then we have a PPT attacker \mathcal{A} who can break \mathcal{E}_{pk} in the IND-ATK mode with a non-negligible advantage $\text{Adv}(\mathcal{A})$. We let Simon conduct a reduction by using \mathcal{A} to break \mathcal{E}_{pk} in the NM-ATK mode.

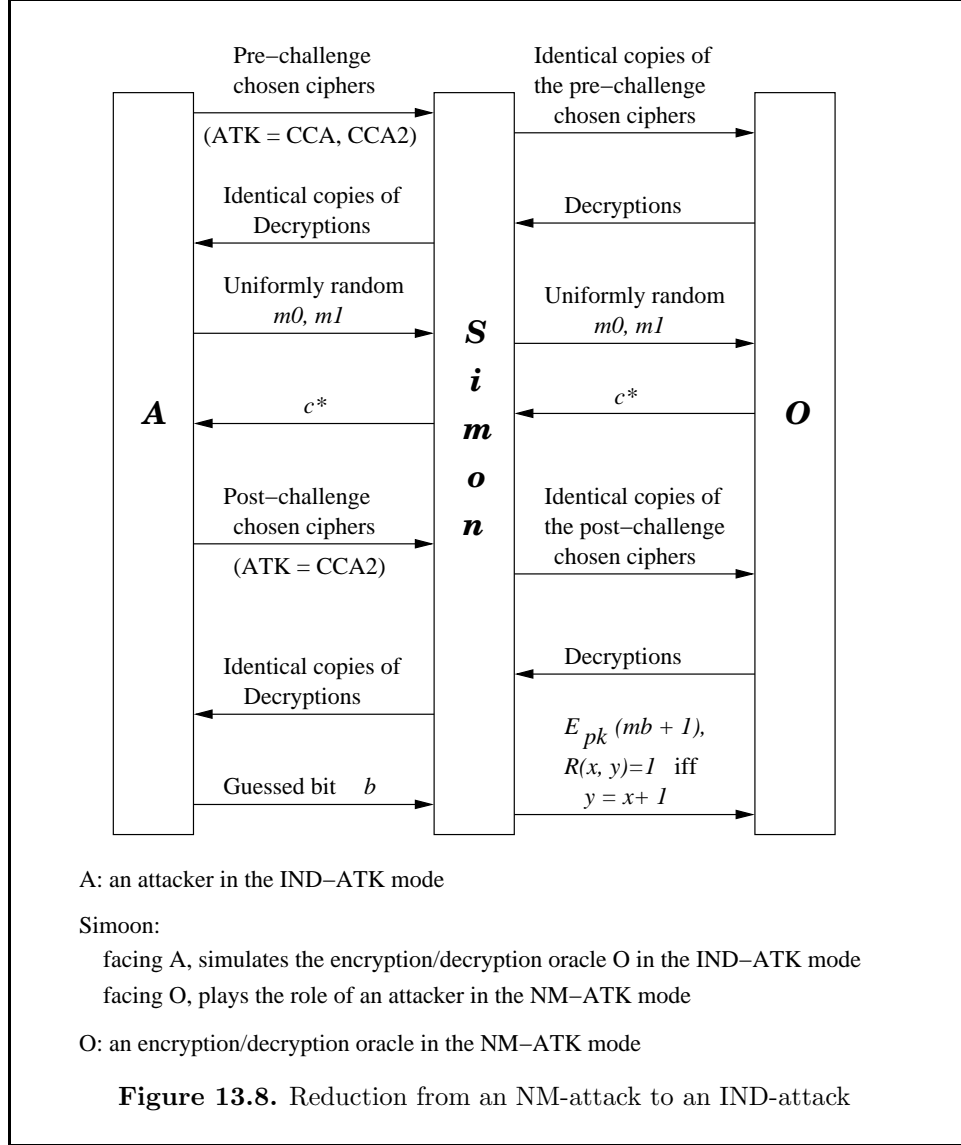
Simon is in orchestration between two attack games. One game is in the IND-ATK mode (i.e., any of the Protocols 13.1, 13.3, or 13.4) in which Simon plays the role of \mathcal{O} in interaction with \mathcal{A} who is in the position of Malice. The other game is the NM-ATK mode (i.e., the ATK version of Protocol 13.5) in which Simon plays the role of Malice in interaction with an encryption oracle (and decryption oracle if $\text{ATK} \in \{\text{CCA}, \text{CCA2}\}$) \mathcal{O} . Figure 13.8 illustrates the reduction for the most general case of $\text{ATK} = \text{CCA2}$. Some of the interactions can be omitted in the other two cases of ATK.

Notice that in the malleability-ATK game (i.e., the right-hand side interactions in Figure 13.8), the description on the distribution of the chosen plaintexts is uniform; therefore \mathcal{O} will have to encrypt a random choice of the chosen plaintexts.

The “educated guess” from \mathcal{A} is $b \in \{0, 1\}$. Simon then has freedom to output $c' = \mathcal{E}_{pk}(m_b + 1)$ and the relation $R(x, y) = 1$ if and only if $y = x + 1$ for all x in the plaintext space. Clearly, with \mathcal{A} being PPT, Simon can output this correct malleability result also in polynomial time.

Since \mathcal{A} answers b with advantage $\text{Adv}(\mathcal{A})$, we have

$$\text{NM-Adv}(\text{Simon}) = \text{Adv}(\mathcal{A}) - \text{Prob}[(\bar{c}, R) \leftarrow \text{ZK-Sim}].$$



Notice that ZK-Sim does not have access to the challenge ciphertext c^* and hence does not have the use of \mathcal{A} ; so for the simulated output ciphertext \bar{c} to correspond a plaintext satisfying R , $\text{Prob}[(\bar{c}, R) \leftarrow \text{ZK-Sim}]$ must be negligible. Hence, $\text{NM-Adv}(\text{Simon})$ is non-negligible as desired. \square

Recall that we have demonstrated numerous attacks in various IND-ATK modes on various cryptosystems. By Theorem 13.3, these cryptosystems are also insecure

in the respective NM-ATK modes.

It is known that there exists cryptosystems which are IND-CPA (respectively, IND-CCA) secure, but are not NM-CPA (respectively, NM-CCA) secure. These cases can be found in [BDPR98].

Among the relations among NM and IND security notions which have been investigated in [BDPR98], the following relation is the most important one.

13.5.4.2 Indistinguishability implying non-malleability under adaptive chosen ciphertext attack

For the case of $\text{ATK} = \text{CCA2}$, the converse of the statement in Theorem 13.3 is also true.

Theorem 13.4: *A public-key cryptosystem is NM-CCA2 secure if and only if it is IND-CCA2 secure.*

Proof Since in Theorem 13.3 we have established $\text{NM-CCA2} \implies \text{IND-CCA2}$, we only need to establish the opposite case: $\text{IND-CCA2} \implies \text{NM-CCA2}$. We can show that if a public-key cryptosystem \mathcal{E}_{pk} is NM-CCA2 insecure then it must be IND-CCA2 insecure.

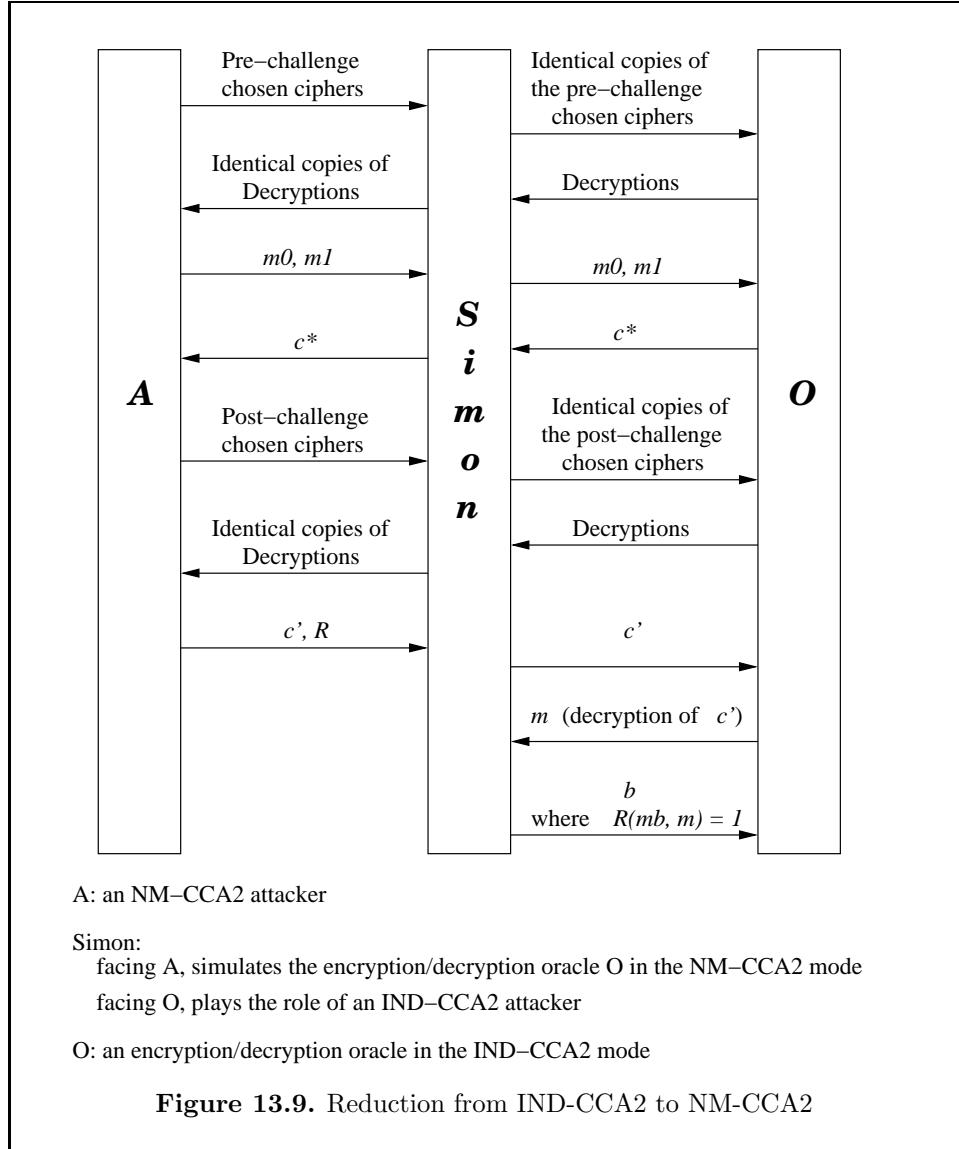
Suppose \mathcal{E}_{pk} is NM-CCA2 insecure. Then we have a PPT attacker \mathcal{A} who can break \mathcal{E}_{pk} in NM-CCA2 with a non-negligible advantage $\text{Adv}(\mathcal{A})$. We let Simon conduct a reduction by using \mathcal{A} to break \mathcal{E}_{pk} in the IND-CCA2 mode.

Figure 13.9 illustrates the reduction orchestrated by Simon. Notice that the reduction is possible exactly because the ciphertext c' output by the malleability attacker \mathcal{A} is different from the challenge ciphertext c^* , and therefore the orchestrator of the two games, Simon who plays the role of Malice in the IND-CCA2 game, can send c' to \mathcal{O} for decryption a post-challenge chosen ciphertext. With the decryption result, Simon can verify the relation between the plaintexts (the relation has been given by \mathcal{A}) and thereby determines the challenge bit b .

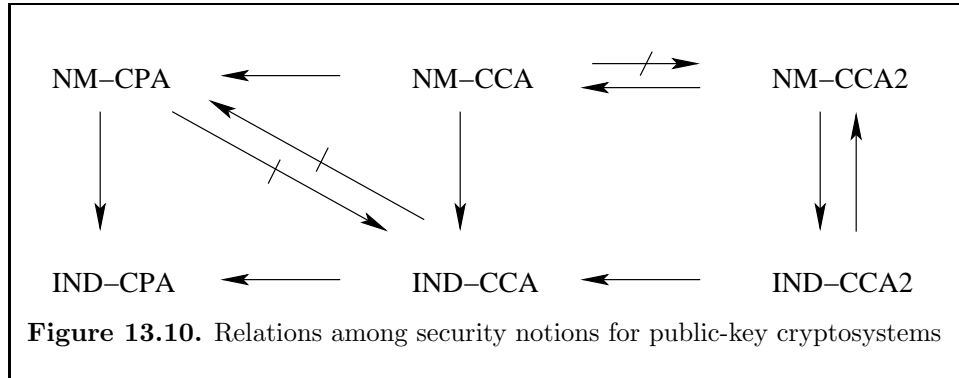
Clearly, since \mathcal{A} is PPT, the two games orchestrated by Simon will terminate in polynomial time, and the advantage of Simon is non-negligible since the advantage of \mathcal{A} is non-negligible. \square

Figure 13.10 summarize the known relations among the security notions we have introduced so far. We have not demonstrated the non-implication cases (those separated by \nrightarrow). The interested reader may study [BDPR98] for details.

Theorem 13.4 tells us that in the context of public-key encryption schemes, we only need to consider IND-CCA2 notion which is easier to handle than NM-CCA2 notion is. Also, due to the equivalence relation between IND-CCA2 and NM-CCA2, it is now generally agreed that IND-CCA2 is *the* right definition of security for public key encryption algorithms for general use.



In the next chapter we shall introduce two practically efficient public-key cryptosystems which are provable IND-CCA2 secure.



13.6 Strong Security Notion for Digital Signatures

13.7 Chapter Summary

In this chapter we have taken step-wise approach to introducing progressively stronger security notions for public-key cryptosystems. We start with a protocol which uses a typical “textbook version” encryption algorithm and seeing its weakness and unsuitability for applications. We then introduce a first-step strengthened security notion: semantic security or indistinguishable encryption under passive attack. Weaknesses of semantic security are exposed, which are followed by further steps of strengthening steps. We finally reach the strongest security notion for public-key cryptosystems: indistinguishable encryption under adaptively chosen ciphertext attack (IND-CCA2) which we consider to be a “fit-for-application” security notion. Finally, we consider the security notion for public-key encryption against a different attacking scenario: non-malleability, and relate the notions of IND-CCA2 and non-malleability.

Nowadays, IND-CCA2 is the standard and “fit-for-application” security notion for public-key cryptosystems. All new public-key encryption schemes should have this security quality. In the next chapter we shall introduce practical public-key cryptosystems which are formally provably secure under the IND-CCA2 notion.

PROVABLY SECURE AND PRACTICALLY EFFICIENT PUBLIC-KEY CRYPTOSYSTEMS

14.1 Introduction

In the previous chapter we have seen that early solutions to IND-CCA2 (equivalently, NM-CCA2) secure public-key cryptosystems have generally rested on applications of non-interactive zero-knowledge (NIZK) proof techniques. Such a proof shows a receiver of a ciphertext that the creator of the ciphertext knows the corresponding plaintext because what is proved is the following NP membership statement:

“The ciphertext c is in language L defined by encryption algorithm \mathcal{E} under public key pk , and the creator of c has in their possession an auxiliary input (i.e., a witness of an NP problem) for the membership proof.”

Here “auxiliary input” for the membership proof consists of the corresponding plaintext and a random input to the encryption algorithm \mathcal{E} ; the random input is necessary since the encryption scheme needs to be semantically secure. If the verification of such a proof outputs “Accept”, the receiver, who may be required or tricked to provide a decryption service, can then be sure that even if the creator of the ciphertext c is Malice (the bad guy), returning the corresponding plaintext to him is returning to him something he already knows, and hence doing so will not help him in anyway should he attack the target cryptosystem.

While this is a sound idea, it is quite an expensive one. The general method for realizing NIZK proof is for the prover (here, the creator of a ciphertext) and the verifier (here, a receiver of that message) to share a *mutually trusted* random string. This demand is way beyond what an encryption scheme should ask for. If we consider that eliminating the need for two parties to share secret information before secure communication constitutes the most important advantage of public-key cryptography^a, then provable security for public-key encryption schemes should not be built at the expense of regressing back to sharing mutually trusted information between communication parties!

In fact, all provable security should achieve is an affirmative measure to ensure that Malice should not be able to do something bad *too often* or *fast enough*. So provable security can be established as long as we can establish the success probabilities and computational cost for Malice to succeed an attack. In the context of achieving a provably secure encryption, to ask for a guarantee that Malice must know the corresponding plaintext of a ciphertext is to ask for too much, and NIZK proof is unnecessary and overkill. In fact, none of the previous public-key encryption schemes which are provably IND-CCA2 secure based on applying NIZK proof techniques is sufficiently efficient for practical use.

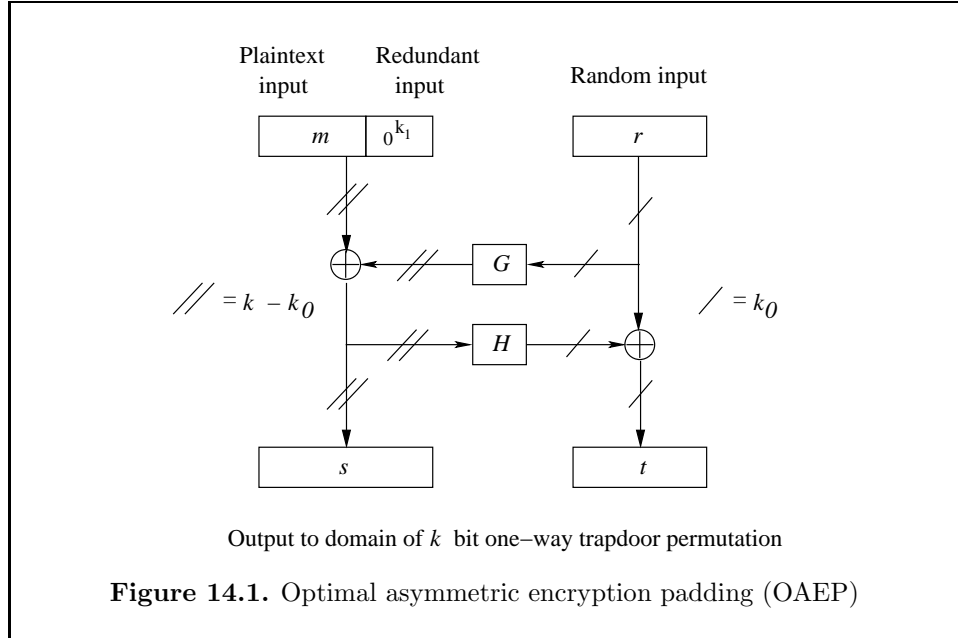
Many practically efficient and provably secure public-key encryption and digital signature schemes have been proposed. These schemes are mainly results of enhancing popular textbook public-key algorithms or digital signature schemes using a message integrity checking mechanism. Here, by textbook public-key algorithms (see §8.6), we mean those which are direct applications of some one-way trapdoor functions such as the RSA, Rabin and ElGamal functions. A message integrity checking mechanism allows us to establish the success probabilities and the computational costs for Malice to be successful in attacking the enhanced scheme.

The cost for using a scheme enhanced this way is at the level of a small constant multiple of that for using the underlying textbook public-key encryption algorithm.

14.1.1 Chapter Outline

In this chapter we shall introduce the two best known public-key encryption schemes which are provably secure against IND-CCA2 and are practically efficient. They are the **Optimal Asymmetric Encryption Padding (OAEP)** [BR95a], [Sho01a], [FOPS01] (§14.2) and the **Cramer-Shoup public-key cryptosystem** [CS98] (§14.3). We shall then conduct an overview on a family of so-called **hybrid cryptosystems** which are combination of public-key and secret-key encryption algorithms, are provably secure against IND-CCA2 and are practically efficient (§14.4). We shall end this chapter with a literature review of practical and provably secure public-key cryptosystems (§14.5).

^aWe have witnessed ways of secure communication even without need for two parties to share public information, see §12.3.



14.2 The Optimal Asymmetric Encryption Padding (OAEP)

The Optimal Asymmetric Encryption Padding (OAEP) is invented by Bellare and Rogaway [BR95a]. This is a **randomized message padding technique** and is an easily invertible transformation from a plaintext message space to the domain of a one-way trapdoor permutation (OWTP). The RSA and Rabin functions are the two best-known OWTP^b. The transformation uses two cryptographic hash functions and takes a plaintext message, an added redundancy and a random value as the input. Figure 14.1 depicts the transformation. Detailed instructions for using the RSA-OAEP scheme (i.e., the OWTP is instantiated under the RSA function) have been specified in Algorithm 9.6.

An OAEP based public-key encryption scheme can be viewed as the following combined transformation

$$\text{Plaintext} \xrightarrow{\text{OAEP}} \text{Domain of OWTP} \xrightarrow{\text{OWTP}} \text{Ciphertext.} \quad (14.2.1)$$

The central idea behind this combined transformation can be explained as follows.

As we have discussed in §8.3.1, usually a mathematical function underlying a textbook public-key algorithm has very well-behaving and public algebraic properties. These algebraic properties are from an underlying algebraic structure in which

^bSee §13.3.6.1 for why the recommended way of using the Rabin encryption algorithm forms OWTP.

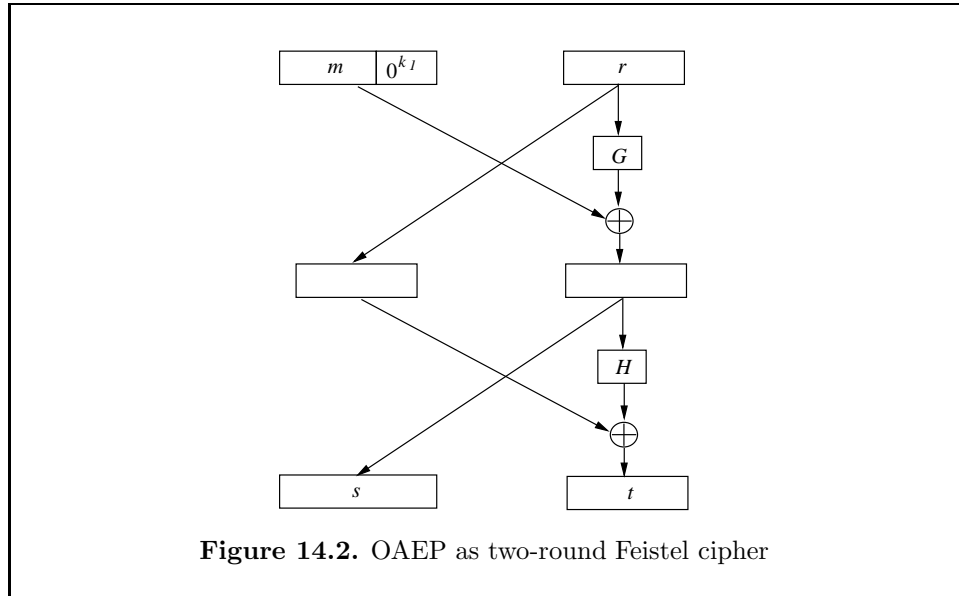


Figure 14.2. OAEP as two-round Feistel cipher

the OWTP is defined (e.g., axioms of a group or a field provide very nice algebraic properties, see Definitions 5.1 and 5.13). A large number of attacks on textbook public-key encryption algorithms we have shown so far (including those on some semantically secure schemes, e.g., the GM cryptosystem, Algorithm 13.1 which is attacked in Examples 13.2 and 13.3) have invariantly shown a general technique for Malice to attack textbook encryption algorithms: manipulating a ciphertext such that the corresponding plaintext can be modified in a controlled way thanks to the nicely-behaving algebraic properties of the OWTP.

Rather differently, the OAEP transformation is constructed by networking cryptographic hash functions with a well-known symmetric crypto-algorithmic structure. Indeed, as shown in Figure 14.2 (compare with Figure 7.3), the OAEP construction can be viewed as a two-round Feistel cipher, with the first round using a hash function G and the second round using a hash function H in places of an “s-box function” for a Feistel cipher; though here the two “s-box functions” are not keyed, and the two “half blocks” can have different sizes.

These two kinds of structures, i.e., the structures of the OWTPs underlying popular public-key cryptosystems and the Feistel-cipher structure of OAEP, have vastly different algebraic properties. For a rough judgement we can apparently see that a former structure has block-wise algebraic properties in a large-order space while the latter structure has bit-wise (i.e., in an order-2 space) algebraic properties. We should therefore have a good hope that the combined transformation (14.2.1) should cause a tremendous difficulty for Malice to modify a plaintext in a controlled way via manipulating the corresponding ciphertext.

Well, a good hope is one thing, a formal proof of security, i.e., a measure on added difficulty for Malice to control plaintext, based on rigorous mathematical reasonings is another. We want the latter.

Formal proof for an OAEP encryption scheme is based on a powerful technique called **random oracle model**. Such a proof assumes that hash functions used in the construction of OAEP have a so-called **random oracle** property (to be discussed in a moment). Given such a property, a random-oracle based proof will construct an efficient transformation which translates an advantage for an alleged attack on the OAEP scheme to a similar (up to polynomially different) advantage for inverting the OWTP used in the scheme. Nevertheless, it is on the other hand widely believed that there exists no efficient algorithm for inverting the OWTP. For example, for the OWTP being the RSA function, the inversion actually solves the RSA problem or breaks the RSA assumption (Definition 8.4, Assumption 8.3). As this is very unlikely, the alleged efficient attack should be same unlikely. That's the idea for formal proof.

14.2.1 Random Oracles

A cryptographic hash function (§9.2.1) or a pseudo-random number generator (see e.g., §8.8.2 for such a generator) have some behavior which is very useful in modern cryptography. First, these functions are efficient: they produce an output in time polynomial in the size of an input. Secondly, these functions are deterministic: evaluation of a function twice using the same input produces the same output. Thirdly, the output values looks random following a widely believed assumption (Assumption 4.2): there exists a function such that, for non-uniform input values, the output values have a distribution which is indistinguishable from the uniform distribution in the output space of the function. We must notice that, the output values of such a deterministic function can in no way be uniform in their output space. This is because, according to Shannon's entropy theory (Theorem 3.2) the entropy of the output value can never be greater than that of the corresponding input value (review §3.6).

In order to reason about the security of a cryptographic scheme or a protocol which uses hash functions or pseudo-random numbers, Bellare and Rogaway make an idealization step to the third property of these functions: the output being uniform rather than being indistinguishable from uniform [BR93]. This seemingly small step of idealization results in a very powerful and imaginary function named **random oracle**.

We regard random oracle a *very powerful* function because of the combination of these three properties, i.e.: *deterministic*, *efficient* and *uniform* output. These three properties mean that a random oracle can relate two independent values efficiently and deterministically. For example, a random oracle RO can efficiently find a relation between two uniformly independent values $RO(x)$ and $RO(x + 1)$ as

a “plus 1” relation in its domain.

We also say random oracle is an imaginary function because from all computational models we know of, there exists no computing mechanism or machinery which can be so powerful. On the one hand, we know how to output uniformly distributed random values efficiently, e.g., by tossing a fair coin, but we do not know how to do it in a deterministic fashion (say, fixing a way to toss a coin would produce a constant and non-trivial output just like the case of a fixed seeding of a pseudo-random number generator always producing a fixed random-looking output). On the other hand, we can also relate a set uniformly independent values deterministically, e.g., by sorting a set of such values so that any two of them has a deterministic relation as the distance between them in the sorted list; but this relation cannot be computed in time polynomial in the size of these random values; in fact, to compute this relation even needs a memory of a size exponential in the size of these random values!

In the real world, if hash functions (or pseudo-random functions) used in a cryptographic scheme or protocol have no “obvious” flaw, then a security proof for such a scheme or protocol using their idealized version can be considered as valid, in particular if the goal of the proof is up to the unproven assumption of polynomial time indistinguishability. Such a proof of security is called a proof based on the random oracle model.

In the case of OAEP (see Figure 14.1), two cryptographic hash functions G and H are used to instantiate two random oracles whose powerful and imaginary behavior will be further described in the proof of the scheme’s security under the IND-CCA2 notion. In a later chapter we will also see random-oracle based proof of security for some digital signature schemes.

Now let us describe random-oracle based proof of security.

14.2.2 Random Oracle Model for Security Proof

In a security proof based on the random oracle model, a special agent, who we have met in the previous chapter (§13.5.4) and given name Simon Simulator, shall be able to, somehow, *simulate* the behavior of *everybody’s* (even including Malice’s) random oracles. We shall see in the next paragraph a perfect way to construct the simulation. So whenever someone wants to apply a random oracle, say G , to a value, say a , (s)he shall unwittingly make a so-called **random oracle query** to Simon; one does this by submitting a to, and subsequently receiving a query result $G(a)$ from, Simon. Simon shall always honestly comply with any query order and duly return a good query result back.

The simulation of a random oracle can be done perfectly in the following way. For oracle G for example, Simon shall maintain a G -list which contains all the pairs $(a, G(a))$ such that a has been queried in the entire history of G . The simulation job is rather mundane: for each query a , Simon shall check whether or not a is

already in the list; if it is, he shall just return $G(a)$ as the query result (that's why *deterministic*); otherwise, he shall invent a new value for $G(a)$ at uniformly random in the range of G , return this new value as the query result (that's why *uniform*) and archive the new pair $(a, G(a))$ in the list. Simon could build his list so that the pairs are sorted by the first element. There is no need to apply a sorting algorithm because each list is initialized to empty at the beginning, and grows as queries arrive. For each query, search through a sorted list of N elements can be done in $\log N$ time (see Algorithm 4.4), i.e., in PPT in the size of the elements (and that's why *efficient*). We have reached the following statement:

Lemma 14.1: *A random oracle can be simulated perfectly in PPT.* □

For an OAEP-based public-key encryption scheme, this way of simulating random oracles enables Simon to compute the left-hand-side transformation in (14.2.1). Now if an attacker, say Malice, constructs a chosen ciphertext c using an OWTP f , then as long as Malice has used Simon's random oracle services in the construction of c , Simon shall be able to "decrypt" c even though he does not have in his possession of the trapdoor information of f . Indeed, the "plaintext" must be in some of his lists. Therefore, in addition to having simulated random oracles, Simon can *also simulate* a decryption oracle^c, i.e., \mathcal{O} in Protocol 13.3 or Protocol 13.4. This is another reason why we have named our special agent Simon Simulator. The simulated "decryption" capability enables Simon to offer proper "cryptanalysis training course" to Malice in IND-ATK games.

If the "training course" is provided at the precise quality (i.e., the simulated "training course" is accurate) and if Malice is educatable and shall thereby end up with a non-negligible advantage in short enough time (PPT) to break the encryption scheme (i.e., in the IND-ATK case, he ends up to relate one of the two chosen plaintexts to the challenge ciphertext), then Simon shall also end up with a successful inversion of the OWTP since he shall have in his possession of the plaintext-ciphertext pair! (Details will be described in §14.2.4.1.)

This does constitute a valid argument, however, only in a world with random oracles!

Nevertheless, due to the good approximation to the random oracle behavior from cryptographic hash functions, this argument provides a convincing heuristic insight for an OAEP enhanced encryption scheme being secure in the real world. Even though we know that it is only an unproven assumption that a cryptographic hash function emulates a random oracle behavior in a PPT indiscernible manner, this assumption has now been widely accepted and used in practice. After all, each of the reputable OWTPs underlying a popular public-key cryptosystem is an unproven but widely accepted assumption.

^cThe reader must not confuse a "decryption oracle" with a "random oracle", they are totally different things. The former can be real, e.g., a naive user tricked by Malice to provide a decryption service, while the latter is an imaginary function.

Goldreich considers (in “§6.2 Oded’s Conclusions” of [CGH01]) that random oracle model based argument for security is a useful test-bed; cryptographic schemes which does not perform well on the test-bed (i.e., cannot pass the sanity check) should be dumped. It is widely agreed that designing a cryptographic scheme so that it is argued secure in the random oracle model is a good engineering principle.

14.2.3 RSA-OAEP

In the case of **RSA-OAEP**, the OWTP is the RSA encryption function.

The RSA-OAEP is computationally very efficient, almost as efficient as the text-book RSA. It also has a very good message expansion ratio: for the usual standard length of 1024-bit RSA modulus, a plaintext message can have a length up to 75% of the modulus (we can conveniently consider this as a rather good signal/noise ratio of 75%). These practically important features have been widely recognized by the practitioners so that the scheme has been accepted as the RSA encryption standard under industrial and international standardization organizations (PKCS#1, IEEE P1363). It has also been chosen to use in the well-known Internet electronic commerce protocol SET [SET97].

So, RSA-OAEP is a very successful public-key encryption scheme. However, for its provable security, success is a son of failure.

14.2.4 A Twist in the Security Proof for RSA-OAEP

The original random oracle based proof for f -OAEP [BR95a] tried to relate an attack on the f -OAEP scheme in the IND-CCA2 mode to the problem of inverting the OWTP f without using the trapdoor information of f . Recently, Shoup has made an ingenious observation and revealed a flaw in that proof [Sho01a]. Moreover, he points out that for f being a general case of OWTP, it is unlikely to exist a random oracle based proof for f -OAEP to be secure against IND-CCA2. Fortunately and very quickly, the danger for us to lose a very successful public-key encryption algorithm standard was over! A closer observation is made by Fujisaki et al [FOPS01] and they find a way to rescue OAEP for f being the RSA function.

Let us now review this dramatic matter. We shall start with studying the original security argument attempted by Bellare and Rogaway. We then describe Shoup’s observation of a flaw in that argument. Finally we shall see the rescue work of Fujisaki et al (Shoup also works out a special case for the same rescue, and we shall see that as well).

14.2.4.1 Random oracle reduction technique

Suppose that an attacker \mathcal{A} , who is a PPT algorithm, can have a non-negligible advantage to break an f -OAEP scheme in the IND-CCA2 mode. Let us construct

an algorithm which will enable our special agent, Simon Simulator, to make use of the IND-CCA2 attacker \mathcal{A} to invert the OWTP f , also with a non-negligible advantage. This algorithm must be efficient (i.e., a PPT one). Thus, Simon efficiently “reduces” his task of inverting f to \mathcal{A} ’s capability of attacking the f -OAEP scheme. The algorithm used by Simon is therefore called a **polynomial time reduction**. Since both \mathcal{A} and the reduction run by Simon are polynomial time, inversion of f as the combination of \mathcal{A} and the reduction conducted by Simon then also runs in polynomial time. It is the belief that inversion of f cannot be done in PPT that should refute the existence of the alleged IND-CCA2 attacker \mathcal{A} on f -OAEP (however, we should be careful about an issue which we shall discuss in §??). A security proof in this style is called **reduction to contradiction** or a **reductionist proof**.

Let us now describe the reduction.

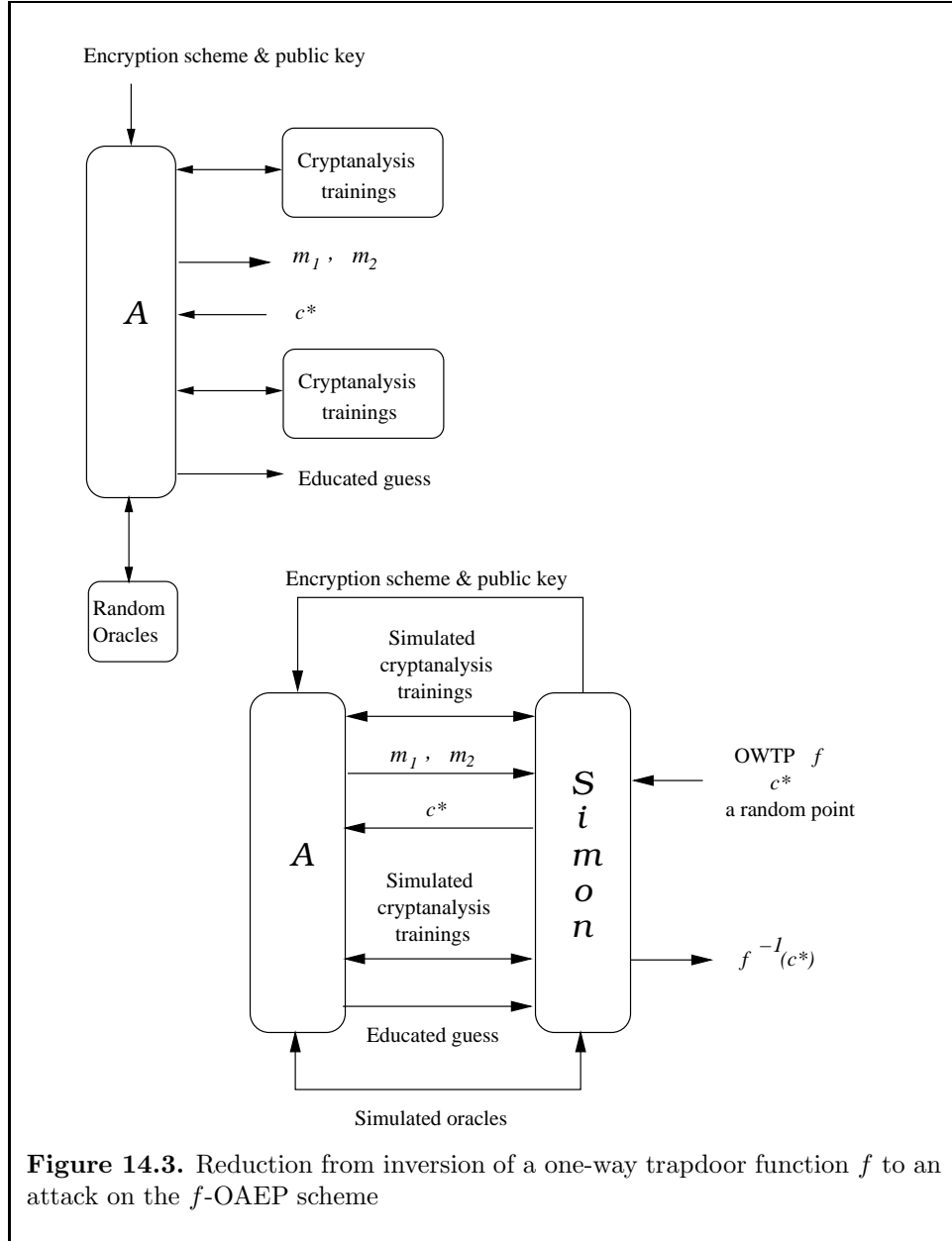
Let Simon be given (the description of) an OWTP f and a uniformly random point c^* in the range of f . Simon wants to uncover $f^{-1}(c^*)$ by using \mathcal{A} as an IND-CCA2 attacker. (Notice that the randomness of c^* is important, or else we are not considering an algorithm.) Simon sends (the description of) the f -OAEP encryption algorithm to \mathcal{A} . In this reduction:

- Simon shall play with \mathcal{A} an IND-CCA2 attack game (i.e., they run Protocol 13.4). In this game, Simon shall impersonate the decryption oracle \mathcal{O} as if he has in his possession a valid decryption box. The impersonation is via simulation.
- Simon shall also provide \mathcal{A} with simulated services for the random oracles G and H used in OAEP (see Figure 14.1). So as we have stipulated in §14.2.2, whenever \mathcal{A} wants to apply G and/or H (e.g., when it wants to prepare a chosen ciphertext in a proper way during the game play), it shall actually make queries to Simon and subsequently gets the respective query results back from Simon. Of course, \mathcal{A} does not have to make random oracle query to Simon if it does not want to prepare chosen ciphertext in a proper way.

Figure 14.3 provides a visual aid for the reduction: Simon has taken over all of the communication links of \mathcal{A} to the external world.

It is vitally important that the simulations provided by Simon must be accurate so that \mathcal{A} cannot feel anything wrong in its communications with the outside world. Only under a precise simulation, \mathcal{A} can be fooled by Simon into thinking that it is in a real attack game. Then \mathcal{A} ’s “educated guess” of the bit b in the simulated game (i.e., in the reduction) should be consistent with that in a real game, and therefore shall provide a consistent mapping under the combined transformation (14.2.1) from m_b to the challenge ciphertext c^* . It is this very mapping that shall enable Simon to invert f : the mapping says that the pre-image $f^{-1}(c^*)$ can be constructed if Simon has recorded all random oracle queries made by \mathcal{A} .

They play the IND-CCA2 attack game as follows.



- i) In \mathcal{A} 's "find stage", Simon shall receive from \mathcal{A} indifferent chosen ciphertexts for decryption (i.e., those in a lunchtime attack). \mathcal{A} has freedom to construct these ciphertexts in any way it wishes; but if it does want to construct them

properly, e.g., via applying the random oracles, then its queries must go to Simon (as shown in Figure 14.3, Simon has taken over all \mathcal{A} 's communication channels to the external world). The ways for Simon to simulate these random oracles will be described in a moment.

- ii) Since Simon receives from \mathcal{A} chosen ciphertexts for decryption, Simon shall answer them to \mathcal{A} by simulating the decryption box (oracle \mathcal{O}). Details for Simon to simulate \mathcal{O} shall also be given in a moment.
- iii) \mathcal{A} shall end its “find stage” by submitting to Simon a pair of chosen plaintexts m_0, m_1 . Upon receipt of them, Simon shall pick a random bit b by tossing a fair coin, and send to \mathcal{A} the “challenge ciphertext” c^* as a *simulated* f -OAEP encryption of m_b .
- iv) Now \mathcal{A} is in its “guess stage”. So it may submit further adaptively chosen ciphertexts for its “extended cryptanalysis training course”. Simon shall serve as in (ii). In case \mathcal{A} makes random oracle queries in its proper construction of the adaptively chosen ciphertexts, Simon shall serve as in (i).

As we have agreed and emphasized many times, \mathcal{A} should not submit the “challenge ciphertext” c^* for decryption. (In fact, were c^* submitted, it would be impossible for Simon to provide a simulated decryption since this is the very ciphertext that Simon needs \mathcal{A} 's help to decrypt.)

Eventually, \mathcal{A} should output its educated guess on the bit b . This is the end of the attack game.

Simulation of Random Oracles. Simon shall simulate the two random oracles G and H used in the OAEP transformation. In the simulation, Simon maintains two lists, called his G -list and his H -list, both are initially set to empty:

G -oracle Suppose \mathcal{A} makes G -query g . Simon shall first search his G -list trying to find g . If g is found in the list, Simon shall provide $G(g)$ to \mathcal{A} . Otherwise, g is fresh; then Simon picks at uniformly random a string $G(g)$ of length k_0 , provides $G(g)$ to \mathcal{A} and adds the new pair $(g, G(g))$ to G -list.

If the query occurs in \mathcal{A} 's “guess stage”, then Simon shall try to invert f at the point c^* as follows. For each $(g, G(g)) \in G$ -list and each $(h, H(h)) \in H$ -list, Simon builds $w = h \parallel (g \oplus H(h))$ and checks whether $c^* = f(w)$. If this holds for some so-constructed string, then $f^{-1}(c^*)$ has been found.

H -oracle Suppose \mathcal{A} makes H -query h . Simon shall first search his H -list trying to find h . If h is found in the list, Simon shall provide $H(h)$ to \mathcal{A} . Otherwise, h is fresh; then Simon picks at uniformly random a string $H(h)$ of length $k - k_0$, provides $H(h)$ to \mathcal{A} and adds the new pair $(h, H(h))$ to H -list.

If the query occurs in \mathcal{A} 's “guess stage”, then Simon shall try to invert f at the point c^* as in the case of G -oracle.

Simulation of the Decryption Box. Simon shall simulate the decryption box (oracle \mathcal{O}) as follows. Upon receipt of ciphertext c from \mathcal{A} for decryption, Simon looks at each query-answer $(g, G(g)) \in G\text{-list}$ and $(h, H(h)) \in H\text{-list}$. For each pair taken from both lists, Simon computes

$$w = h \parallel (g \oplus H(h)),$$

$$v = G(g) \oplus h,$$

and checks

$$c \stackrel{?}{=} f(w)$$

and

Does v have k_1 trailing zeros?

If the both checking steps yield “YES”, Simon shall return the most significant $n = k - k_0 - k_1$ bits of v to \mathcal{A} . Otherwise, Simon shall return REJECT to \mathcal{A} .

Because \mathcal{A} is polynomially bounded, the number of random oracle queries and that of decryption requests made by \mathcal{A} are also polynomially bounded. Hence, Simon can run the simulated game in polynomial time.

14.2.4.2 Accuracy of the simulation

As we have mentioned, for \mathcal{A} to work properly, the accuracy of the simulations is vitally important (is everything)!

First of all, as we have established in Lemma 14.1, the two random oracles have been perfectly simulated.

Now let us examine the accuracy of Simon’s simulation of the decryption box.

Let Simon be given a chosen ciphertext c (either a pre-challenge one or a post-challenge one, i.e., either an indifferently chosen one or an adaptively chosen one). Simon’s simulation for the decryption box is in fact very accurate. Let

$$s \parallel t = f^{-1}(c), \tag{14.2.2}$$

$$r = t \oplus H(s), \tag{14.2.3}$$

$$m \parallel 0^{k_1} = s \oplus G(r) \tag{14.2.4}$$

be the values which are defined by c should c be a valid ciphertext. Below, whenever we say “the correct value”, we mean the value processed by a real decryption oracle \mathcal{O} should c be sent to \mathcal{O} .

If the correct s defined by c in (14.2.2) has not been queried for random oracle H , then the correct $H(s)$ is missing. So in each G -query, we can only have have probability at the level of 2^{-k_0} for r defined in (14.2.3) being correct. So like the missing of the correct value s being queried for H , the correct value r is also missing

from being queried for G (except for probability at the level of 2^{-k_0}). Consequently, as in (14.2.4), value $s \oplus G(r)$ can have probability 2^{-k_1} to have k_1 trailing zeros since this requires s and $G(r)$ to have their k_1 least significant bits identical bit by bit, but the former is missing, and the latter is uniformly random. Notice that in this analysis we have already also considered the case for the correct r having not been queried for G : rejection is correct except for an error probability of 2^{-k_1} .

In summary, we can conclude the following result about the simulated decryption of a chosen ciphertext c :

- if both s and r have been queried for the respective random oracles, then the simulated decryption can correctly construct $f^{-1}(c)$ and thereby further decrypt c in the usual way;
- if *either* s and/or r has not been queried for the respective random oracles, then it is correct for the simulated decryption to return REJECT except for an error probability at the level of $2^{-k_0} + 2^{-k_1}$.

Notice that in the case of *either* s and/or r having not been queried for the respective random oracles, the error probability bound holds in *statistical* sense: namely, the probability bound holds as long as \mathcal{A} has not made the necessary random oracle query regardless of how powerful \mathcal{A} may be.

At this point we can confirm that our argument so far has already shown that f -OAEP is provably secure against IND-CCA (i.e., lunchtime attack or indifferent chosen ciphertext attack). This is because in an IND-CCA attack game the decryption box only works in “find stage”, and we have established that in that stage, the simulated decryption works accurately except for a minute error probability.

We should explicitly emphasize that our argument is solely based on the one-way-ness of f .

14.2.4.3 Incompleteness

Shoup observes that the original OAEP security argument (for IND-CCA2 security) contains a flaw [Sho01a]. Before we go ahead to explain it, let us make it very clear that the OAEP construction is *not* flawed. It is the formal proof described in §14.2.4.2 that has not gone through completely for IND-CCA2 security. A short way to say the incompleteness can be as follows:

The simulated decryption performed by Simon is statistically precise as long as s^* defined by the challenge ciphertext c^* in (14.2.5) is not queried for random oracle H , but the statistical precision falls apart as soon as s^* is queried. However, the possibility for s^* being queried was not considered in the security argument in §14.2.4.2.

In order to explain the incompleteness, let us consider various values defined by the challenge ciphertext c^* . Let

$$s^* \parallel t^* = f^{-1}(c^*), \quad (14.2.5)$$

$$r^* = t^* \oplus H(s^*), \quad (14.2.6)$$

$$m_b \parallel 0^{k_1} = s^* \oplus G(r^*). \quad (14.2.7)$$

The three values (s^*, r^*, m_b) are defined by the challenge ciphertext c^* where b is the coin tossing result performed by Simon.

Let us now imagine that s^* is queried for random oracle H . Of course, in statistics this must be remotely unlikely in \mathcal{A} 's "find stage" since at that point in time it has not yet been given the challenge ciphertext c^* . This is why we have concluded at the end of §14.2.4.2 that the argument there does provide a *valid* proof for f -OAEP being secure in the IND-CCA mode. However, it maybe *possible* for \mathcal{A} to query s^* in its "guess stage" when it has been given the challenge ciphertext c^* .

What is the probability for \mathcal{A} to query s^* in its "guess stage"? Well, we do not know for sure. All we definitely know is that: given \mathcal{A} being allegedly powerful, there is no way for us to deny a possibility for it to query s^* in its "guess stage". Otherwise, why should we have assumed \mathcal{A} being able to guess the bit b in first place? (Nevertheless, the conditional probability bound for s^* having been queried, given that \mathcal{A} answers correctly, can be estimated; the result is non-negligible. We shall provide an estimate in §14.2.4.4.)

As long as \mathcal{A} can query s^* , it can find discrepancy in the simulated attack game. An easy way for us to imagine the discrepancy is as follows. For r^* fixed by (14.2.6), \mathcal{A} may further query r^* . The uniformly random $G(r^*)$ returned back will mean little chance for $G(r^*) \oplus s^*$ to meet either chosen plaintexts. So at this moment \mathcal{A} shall shout: "Stop fooling around!" \mathcal{A} should shout so because it has spotted that the "challenge ciphertext" c^* has nothing to do with any of its chosen plaintexts.

Of course, this "easy" way of finding discrepancy would "cost" \mathcal{A} too much: it would have already disclosed both s^* and $t^* = r^* \oplus H(s^*)$ to Simon and hence would have already helped Simon to invert f at the point $c^* = f(s^* \parallel t^*)$!

Shoup has a better exploitation of this problem. He observes that for some f as an OWTP, \mathcal{A} 's ability to query s^* given c^* suffices it to construct a valid ciphertext without querying r^* for random oracle G (in fact, without ever querying G at all in the entire history of the attack game). Moreover, because the valid ciphertext so constructed is a malleability result of another valid ciphertext, this f -OAEP scheme is NM-CCA2 insecure, and by Theorem 13.4, it is also IND-CCA2 insecure. However, the reduction technique described in §14.2.4.1 shall not help Simon to invert f since Simon's G -list can even be empty (i.e., \mathcal{A} has never queried anything for random oracle G)!

Shoup constructs a k -bit OWTP f for a counterexample. He supposes this permutation "xor-malleable": given $f(w_1)$, w_2 , one can construct $f(w_1 \oplus w_2)$ with

a significant advantage. Notice that this is not an unreasonable assumption. In the security proof for an f -OAEP scheme described in §14.2.4.1 we have only required f being one-way and have never required it being non-malleable. After all, as we have seen in the previous chapter, OWTPs underlying popular textbook public-key encryption algorithms are generally malleable, and the general malleability is the very reason for us to enhance a textbook public-key scheme with the OAEP technique.

To make the exposition clearer, let this f not hide the k_0 most significant bits at all. Then this f can be written as follows

$$f(s \parallel t) = s \parallel f_0(t)$$

where f_0 is a “xor-malleable” $(k - k_0)$ -bit OWTP, i.e., given $f_0(t_1)$, t_2 , one can construct $f_0(t_1 \oplus t_2)$ with a significant advantage. Clearly, this f is still an OWTP.

Now consider f -OAEP encryption scheme instantiated under this f . Remember that for c^* being the challenge ciphertext, values s^* , t^* , r^* and $(m_b \parallel 0^{k_1})$ correspond to c^* under this f -OAEP scheme.

Since \mathcal{A} is a black box, we have freedom to describe how he should construct a valid ciphertext out of modifying another valid ciphertext. Upon receipt of the challenge ciphertext c^* , \mathcal{A} decomposes c^* as $c^* = s^* \parallel f_0(t^*)$. It then chooses an arbitrary, non-zero message $\Delta \in \{0, 1\}^{k-k_0-k_1}$, and computes:

$$s = s^* \oplus (\Delta \parallel 0^{k_1}), \quad v = f_0(t^* \oplus H(s^*) \oplus H(s)), \quad c = s \parallel v.$$

Clearly, in order to construct the new ciphertext c from the challenge ciphertext c^* , \mathcal{A} only needs to query s^* and s for H .

Let us now confirm that c is a valid f -OAEP encryption of $m_b \oplus \Delta$ as long as c^* is a valid f -OAEP encryption of m_b . From $t = t^* \oplus H(s^*) \oplus H(s)$, we have

$$r = H(s) \oplus t = t^* \oplus H(s^*) = r^* \tag{14.2.8}$$

Clearly, (14.2.8) holds even though \mathcal{A} has not queried $r = r^*$ for G (he may not even know r^* because it may not know t^*).

Had this game been played between \mathcal{A} and the real decryption oracle \mathcal{O} , then \mathcal{O} would retrieve r properly by computing

$$f^{-1}(c) = s \parallel t,$$

$$r = H(s) \oplus t.$$

But noticing (14.2.8), \mathcal{O} would have actually retrieved r^* . So \mathcal{O} would further apply hash function G , and would compute

$$G(r) \oplus s = G(r^*) \oplus s^* \oplus (\Delta \parallel 0^{k_1}) = (m_b \oplus \Delta) \parallel 0^{k_1}.$$

Upon seeing the trailing k_1 zeros, \mathcal{O} would return $m_b \oplus \Delta$ as the correct decryption of c . From the returned plaintext $m_b \oplus \Delta$, \mathcal{A} can easily extract m_b and hence break this f -OAEP in the IND-CCA2 mode.

However, for this game being played in the reduction between \mathcal{A} and Simon, because Simon's G -list is empty, Simon shall promptly return REJECT. Now, \mathcal{A} will definitely shout very loudly:

“STOP FOOLING AROUND!”

14.2.4.4 Probability for \mathcal{A} to have queried s^* in its “guess stage”

We have left out a small detail regarding the incompleteness of the original proof of Bellare-Rogaway: the conditional probability for \mathcal{A} to have queried s^* in its “guess stage” given that it can answer the challenge bit correctly. Because this part has involved probability estimation, it may be skipped without causing any trouble in understanding how the security proof for RSA-OAEP works. (In fact, the probability estimation is quite elementary, we will state all rule applications by referring to their origins in Chapter 3).

To start with, we suppose that \mathcal{A} somehow has advantage Adv to guess the challenge bit b correctly after he has had enough adaptive chosen ciphertext training.

During the training, Simon may mistakenly reject a valid ciphertext in a decryption query. This is a bad event because it shows \mathcal{A} low quality of the training course. So let this event be denoted DBad . In §14.2.4.2 our examination on Simon's simulated decryption procedure has concluded that the simulated decryption is accurate or a high-quality one: the probability for inaccuracy is at the level $2^{-k_0} + 2^{-k_1}$. So we have

$$\text{Prob}[\text{DBad}] \approx 2^{-k_0} + 2^{-k_1} \quad (14.2.9)$$

Let further AskG (respectively, AskH) denote the event that r^* (respectively, the event s^*) has ended up in G -list (respectively, in H -list). These two events are also undesirable because they disclose to \mathcal{A} information for it to discover that the challenge ciphertext c^* actually has nothing to do with its “chosen plaintexts” m_0, m_1 . Therefore, let us also call them bad events. Define the event Bad as

$$\text{Bad} = \text{AskG} \cup \text{AskH} \cup \text{DBad}.$$

Now, let $\mathcal{A} \text{ wins}$ denote the event that \mathcal{A} makes a correct guess of the challenge bit b . It is clear that in absence of the event Bad , due to the uniform randomness of the values which the random oracles can have, the challenge bit b is independent from the challenge ciphertext c^* . Thus we have

$$\text{Prob}[\mathcal{A} \text{ wins} \mid \overline{\text{Bad}}] = \frac{1}{2}. \quad (14.2.10)$$

Applying conditional probability (Definition 3.3) we can re-express (14.2.10) into

$$\text{Prob}[\mathcal{A} \text{ wins} \mid \overline{\text{Bad}}] = \frac{\text{Prob}[\mathcal{A} \text{ wins} \cap \overline{\text{Bad}}]}{\text{Prob}[\overline{\text{Bad}}]} = \frac{1}{2}$$

or

$$\text{Prob}[\mathcal{A} \text{ wins} \cap \overline{\text{Bad}}] = \frac{1}{2} \cdot (1 - \text{Prob}[\text{Bad}]). \quad (14.2.11)$$

We should notice that in the event $\overline{\text{Bad}}$ (i.e., in absence of Bad), the simulated random oracles and the simulated decryption box work perfectly and are identical to these true functions. So \mathcal{A} 's attacking advantage should be fully released, and we have

$$\text{Prob}[\mathcal{A} \text{ wins}] = \frac{1}{2} + \text{Adv}. \quad (14.2.12)$$

On the other hand (see the law of total probability, Theorem 3.1),

$$\text{Prob}[\mathcal{A} \text{ wins}] = \text{Prob}[\mathcal{A} \text{ wins} \cap \overline{\text{Bad}}] + \text{Prob}[\mathcal{A} \text{ wins} \cap \text{Bad}]. \quad (14.2.13)$$

If we conjunct (14.2.12) and (14.2.13), we have

$$\text{Prob}[\mathcal{A} \text{ wins} \cap \overline{\text{Bad}}] + \text{Prob}[\mathcal{A} \text{ wins} \cap \text{Bad}] = \frac{1}{2} + \text{Adv}$$

or

$$\text{Prob}[\mathcal{A} \text{ wins} \cap \overline{\text{Bad}}] + \text{Prob}[\text{Bad}] \geq \frac{1}{2} + \text{Adv} \quad (14.2.14)$$

Noticing (14.2.11), the inequality (14.2.14) becomes

$$\frac{1}{2} \cdot (1 - \text{Prob}[\text{Bad}]) + \text{Prob}[\text{Bad}] \geq \frac{1}{2} + \text{Adv},$$

that is

$$\text{Prob}[\text{Bad}] \geq \text{Adv}. \quad (14.2.15)$$

Since $\text{Bad} = \text{AskG} \cup \text{AskH} \cup \text{DBad}$, we have

$$\text{Prob}[\text{Bad}] \leq \text{Prob}[\text{AskG} \cup \text{AskH}] + \text{Prob}[\text{DBad}] \quad (14.2.16)$$

$$= \text{Prob}[\text{AskH}] + \text{Prob}[\text{AskG} \cap \overline{\text{AskH}}] + \text{Prob}[\text{DBad}] \quad (14.2.17)$$

$$\leq \text{Prob}[\text{AskH}] + \text{Prob}[\text{AskG} \mid \overline{\text{AskH}}] + \text{Prob}[\text{DBad}] \quad (14.2.18)$$

where (14.2.16) is due to Probability Addition Rule 1, (14.2.17) is due to Example 3.3, and finally (14.2.18) follows the definition for conditional probability and the fact that a probability value is always less than 1.

Finally, we notice that the uniform randomness of the H oracle, the conditional event $\text{AskG} \mid \overline{\text{AskH}}$ (i.e., given that s^* has not been queried, r^* has been queried) can only occur with probability 2^{-k_0} . We have also known from (14.2.9) that probability

for DBad is also at the level of $2^{-k_0} + 2^{-k_1}$. The inequalities (14.2.15—14.2.18) conclude

$$\text{Prob}[\text{AskH}] \geq \text{Adv} - (2^{-k_0+1} + 2^{-k_1}).$$

Therefore, if Adv is non-negligible in k , so is Prob[AskH].

To this end, we can clearly see that if an attacker is capable of breaking the RSA-OAEP implemented by random oracles in IND-CCA2, then the attacker is capable of partially inverting the RSA function: finding s^* with a similar advantage.

14.2.4.5 Rescue work for RSA-OAEP

The mathematics in §14.2.4.4 actually plays an important role in the rescue work for RSA-OAEP. However, the initial rescue attempt did not rely on it.

Shoup's initial attempt

Fortunately, the malleability property of the RSA function is not so similar to that of a Feistel cipher which bases the OAEP transformation (review Figure 14.2 and our discussion there on the difference in algebraic properties between these two structures). Ironically, the significant difference in algebraic properties (or in malleability properties) between these two structures enabled Shoup to prove that the RSA-OAEP is IND-CCA2 secure provided the RSA encryption exponent is extremely small: for N being the RSA modulus, his proof requires

$$k_0 \leq (\log_2 N)/e. \quad (14.2.19)$$

Let us see why.

Recall that our analysis has concluded that if s^* , which is defined by the challenge ciphertext c^* in (14.2.5), is not queried for oracle H , then the reduction is statistically correct. The only case for the reduction being incorrect is when s^* is queried for H . For this case, we have not considered how Simon should act.

Shoup observes that with s^* being a $(k - k_0)$ -bit string in Simon's H -list, Simon can solve the following equation for the RSA problem:

$$(X + 2^{k_0} I(s^*))^e \equiv c \pmod{N} \quad (14.2.20)$$

where $I(x)$ is the integer value for the string x . This equation is solvable in time polynomial in the size of N using Coppersmith's algorithm [Cop96] provided $X < N^{1/e}$. With X being a quantity at the level of 2^{k_0} and with the restriction in (14.2.19), the condition $X < N^{1/e}$ is met.

Thus, upon given a ciphertext c for decryption service, Simon should, upon failure in decrypting c using the method specified in §14.2.4.1, try to solve (14.2.20) for X using each element in his H -list. If all of these attempts fail, Simon should

reject c . Otherwise, the solution is $X = I(t^*)$; knowing s^* and t^* , Simon should decrypt c in the usual way.

Therefore, for this case of the RSA-OAEP, i.e., for the encryption exponent e satisfying (14.2.20), querying s^* for H has already helped Simon to invert c^* . The question is: what is the magnitude of e satisfying (14.2.20)? For the standard security parameter settings for the RSA-OAEP, we have $N > 2^{1024}$, and $k_0 = 160$ (in order for 2^{-k_0} being negligible), and so $e \leq \frac{1024}{160} = 6.4$. So for the standard security parameter settings, $e = 3$ or $e = 5$ are the only possible cases for encryption exponents (e must be co-prime to $\phi(N)$ which is even). Although using so small exponents we can reach provable security for the RSA-OAEP, it is widely recognized after Coppersmith's work, that one should not use so-small exponent for RSA encryption.

Because the standard security parameter setting for k_0 is already close to the lower bound and that for the size of N cannot be increased dramatically, there is little hope to use this reduction method for any larger e .

Full rescue of Fujisaki et al

Fortunately again, soon after Shoup's analysis, Fujisaki et al [FOPS01] made a further observation and found a way to invert the RSA function for the general case of the encryption exponent.

For the case of the RSA-OAEP, disclosing to Simon s^* as a significantly large chunk of the pre-image of c^* has actually already disclosed too much. Because s^* has $k - k_0$ bits and because $k > 2k_0$, more than half the bits (the most significant bits) of the pre-image of c^* are disclosed. Given such a large chunk of the pre-image, Fujisaki et al applied a brilliant lattice technique which can solve for $T = I(t^*)$ from the equation

$$(2^{k_0} I(s^*) + T)^e \equiv c^* \pmod{N} \quad (14.2.21)$$

for arbitrarily large e . Recall that given a one-bit RSA oracle ("RSA parity oracle", review §8.8.1), we have studied an algorithm (Algorithm 8.4) which applies the one-bit oracle $\log_2 N$ times to invert the RSA function. Exactly the same principle applies here: \mathcal{A} is in fact an "RSA half-or-greater-block oracle" since s^* has more than half the bits of the pre-image of c^* . Using the algorithm of Fujisaki et al, Simon can apply \mathcal{A} twice to obtain two related blocks (half-or-greater-block) of partial pre-image information. These two blocks can be used in the formula (14.2.21) for solving two unknown integers which are smaller than \sqrt{N} . One of these smaller integers is $T(t^*)$, and hence, Simon has inverted the RSA function.

Since Simon has to apply \mathcal{A} twice, he should play with \mathcal{A} twice the reduction-via-attack game: once feeding \mathcal{A} with c^* , and once feeding \mathcal{A} with $\bar{c}^* = c^* \alpha^e \pmod{N}$ for a random $\alpha \in \mathbb{Z}_N^*$. The respective s^* and \bar{s}^* will be in his H -list and his \bar{H} -list, respectively. Let $q = \max(\#(H\text{-list}), \#(\bar{H}\text{-list}))$. From these two lists Simon will deal with no more than q^2 pairs (t, \bar{t}) . One of these pairs will enable Simon to

make two correct equations in the formula (14.2.21) and thereby to invert the RSA function, unless he has chosen a bad α which has a small probability (negligible when $k \gg 2k_0$ which is the case in the RSA-OAEP). Because solving two cases of (14.2.21) can be done in time $O_B((\log_2 N)^3)$, Simon can invert the RSA function in time

$$2\tau + q^2 \times O_B((\log_2 N)^3), \quad (14.2.22)$$

where τ is time bound for \mathcal{A} to perform the IND-CCA2 attack on the RSA-OAEP.

The RSA-OAEP has other two variations for the padding parameter settings. They are PKCS#1 versions 2 and higher [PKC01] and SET [SET97]. In these variations, the known data chunk s^* is positioned in different places in the plaintext chunk. Due to the sufficiently large size of s^* (considerably larger than the half-block size), root-extraction can easily be done by at most twice running of \mathcal{A} . So a variation of the technique of Fujisaki et al will still apply to these variations.

In this way, the RSA-OAEP, and the variations, remain provably secure in the IND-CCA2 mode.

14.2.5 Tightness of “Reduction to Contradiction” for RSA-OAEP

The RSA-OAEP scheme is very efficient. However, the “reduction to contradiction” should not be considered so. Let us now explain this issue.

The formula (14.2.22) shows the time needed by Simon to apply \mathcal{A} twice to invert the RSA function at an arbitrary point. In the term q^2 , q is the number of queries that \mathcal{A} is allowed to make in each instance it is used by Simon (i.e., the number of queries in a cryptanalysis training session which \mathcal{A} entitles in an attacking game). Since \mathcal{A} is assumed to be able to succeed an attack in time τ , we have to satisfy its need of making the number of queries which is also related to τ . Thus, we can only bound q by τ . Consequently, Simon’s time complexity to achieve “reduction to contradiction” expressed in (14.2.22) can be re-expressed in τ as

$$O_B(\tau^2 \times O_B((\log_2 N)^3)).$$

Although τ^2 is a small polynomial in τ , the “reduction to contradiction” is far from satisfactorily tight in the case of RSA application. In the RSA application, we consider \mathcal{A} to be a valid attacker if τ is a feasible value and is far less than a value given by (4.9.1) which is the current champion of the time complexity for factoring an integer N .

An example of a valid attacker is one with $\tau \approx 2^{43}$. This value is at a manageable scope for a dedicated and determined attacker. On the other hand, as we have discussed in §4.9, for N being a 1024-bit composite integer, (4.9.1) is a quantity at the level of 2^{86} . This quantity is currently not manageable even by the dedicated attacker with the use of a vast number of computers running in parallel. Now we see that Simon’s time complexity for inverting the RSA function at a cost level of

2^{86} does not actually lead to a contradiction at all since at that time cost, Simon can invert an RSA function of a 1024-bit modulus without a need of using \mathcal{A} . In other words, the “reduction to contradiction” proof is not a valid one for the case of a 1024-bit RSA modulus.

Since 1024-bit RSA moduli are currently regarded within the safe margin for many secure applications, the invalidity of the security proof for the RSA-OAEP exposes the dissatisfaction of the reduction as a degree-2 polynomial.

The “reduction-to-contradiction” proof is valid for larger RSA moduli, for example, for 2048-bit ones.

14.2.6 A Critique on the Random Oracle Model

Canetti, Goldreich and Halevi hold a rather negative view on the random oracle model based security proof [CGH98], [CGH01]. They show that there exists signature and encryption schemes which are provably secure under the random oracle model, but cannot have secure realizations in the real world implementation. Their basic idea is to devise nasty schemes, upon holding of certain condition (basically, when non-randomness is sensed), it outputs the private signing key for the signature scheme case, or the plaintext message in the case of encryption scheme, otherwise it behaves properly as a signature scheme or an encryption scheme. Such nasty schemes can be proved secure under the random oracle model, but is clearly insecure in real-world implementation.

Their steps to create such nasty schemes are rather involved. More interested reader is referred to [CGH01].

It is interesting that the three authors reach rather different conclusions (even in terms of disagreements) regarding the usefulness of the random oracle methodology. They decide to present their disagreements in the most controversial form by having three separate conclusions, one from each author.

Canetti’s conclusion (§6.1 of [CGH01]) is the most critical: the random oracle model is a bad abstraction and leads to the loss of reductions to hard problems (i.e., it nullifies the elegant idea of “reduction to contradiction”). He considers that to identify any useful, special-purpose properties of random oracle can be alternative directions of research.

Goldreich’s conclusion (§6.2 of [CGH01]) considers the problem with the random oracle model to be incompleteness: it may fail to rule out insecure designs due to some flaws in the implementation of random oracles. He therefore recommends that in currently published work, proofs of security in the random oracle model should not be included (we interpret this as: these proofs should not be considered as real proofs). However, he has a rather optimistic bottom-line: the model has its value in playing the role of a test-bed for conducting the sanity check for cryptographic schemes. He further hopes that in the future the model may show more value to be

recommended.

Halevi's conclusion (§6.3 of [CGH01]) regards that the current success of this methodology is due to pure luck: "all the current schemes that are proven secure in the random oracle model happen to be secure also in the real world for no reason". His bottom line: today's standards should be around the schemes with proof in the random oracle model rather than be around those without.

14.3 The Cramer-Shoup Public-key Cryptosystem

Another well-known provably IND-CCA2-secure and practically efficient public-key cryptosystem is the **Cramer-Shoup public-key cryptosystem** [CS98], named after the inventors Cramer and Shoup.

14.3.1 Provable Security Under Standard Intractability Assumptions

We have just seen the general methodology for formally provable security: to "reduce" a reputable hard problem to an alleged attack (i.e., to make use of an allegedly successful attacker to solve a reputable hard problem). We desire such a "reduction to contradiction" proof to have the following two important properties:

- i) the reduction should be efficient; ideally, an allegedly successful attacker for a scheme should be able to solve a hard problem underlying the scheme in an effort similar to that for mounting the attack;
- ii) the intractability assumptions which are required for a scheme being secure should be as weak as possible; ideally, for a public-key encryption scheme based on an OWTP, the only assumption for the scheme to be provably secure should be the intractability of the OWTP function.

Property (i) has a practical importance: an inefficient reduction, even if it is in polynomial time, may provide no practical relation at all between an attack and a solution to a hard problem. For example, if a reduction relation is a polynomial of degree 8 where the security parameter is the usual case of 1024, then the time complexity for the reduction is at the level of $1024^8 = 2^{80}$. Under such a reduction, while an attacker may enjoy an efficient attack which breaks a scheme as fast as 10^{-6} second, the reduction using this attacker will only solve a hard problem in 38 billion years! Such provable security is certainly useless. In fact, as we have seen in §14.2.5, even a reduction measured by a degree-2 polynomial can already be regarded as invalid for applications using quite standard size of security parameters.

One may think that the desired property (ii) is less practically important since it seems merely to follow a general principle in mathematical proof: if weakening of

assumptions does not restrict the derivation of a proof then a proof should only be based on the weakened assumptions. While pursuing a beautiful proof is certainly an important part of the motivation, the importance of property (ii) is more on the practical side. The importance of (ii) is especially true in the design of cryptographic systems; weaker assumptions are easier to satisfy using more practical and available cryptographic constructions, and consequently, cryptographic systems using weaker assumptions provide a higher security confidence than those using stronger assumptions.

The random oracle based proof does not achieve (ii) to an ideal extent. This is because the proof not only needs the intractability of the RSA function (the RSA assumption, Assumption 8.3), it also needs a very much stronger assumption: hash functions used in the OAEP construction should have the random oracle property. We say that this assumption is very strong, in fact, it is unreasonably strong: as we have discussed in §14.2.1 that there exists no random oracle in the real world; consequently, this assumption is mathematically unsatisfiable. Indeed, speaking in practical terms, what we can obtain from the proof of the RSA-OAEP is that we must use high quality hash functions in the construction of the RSA-OAEP scheme. Unfortunately, this is not an absolute confidence which a mathematical proof should provide.

A formal proof of security for a public-key cryptosystem relying solely on the intractability of the underlying OWTP of the cryptosystem is said to be a proof under **standard intractability assumption(s)**. Such a proof establishes security in the real world: it proves that a cryptosystem cannot be broken without breaking the underlying intractability assumption(s).

There are a number of provably secure (in the IND-CCA2 mode) cryptosystems based on standard intractability assumptions, the NM-CCA2 (equivalent to IND-CCA2) secure scheme of Dolev et al [DDN91] is an example. However, as we have discussed in §13.5.3, the need for using NIZK proof in that scheme makes it unattractive for practical applications.

The Cramer-Shoup public-key cryptosystem [CS98] is the first public-key cryptosystem which is practically efficient and provably IND-CCA2 secure under a standard intractability assumption. So this scheme has the property (i) to the ideal quality. We shall also see that the scheme has a tight “reduction to contradiction” proof of security: a linear reduction.

Let us now introduce the Cramer-Shoup scheme.

14.3.2 The Scheme

The Cramer-Shoup public-key encryption scheme is a CCA2 enhancement of the semantically secure ElGamal encryption scheme (see §13.3.5). As in the case of the semantically secure ElGamal encryption scheme, the standard intractability assumption underlying the security of the Cramer-Shoup scheme is the Decisional

Algorithm 14.1: The Cramer-Shoup Public-key Cryptosystem**Key Setup**

Let G be an abelian group of a large prime order q . The plaintext space is G ;
 (* We assume there exists an encoding scheme to code any plaintext as a bit-string into G and decode it back. Given $\text{desc}(G)$, such an encoding scheme can be easily realized, see, e.g., §13.3.5. *)

To set up a user's key material, user Alice performs the following steps:

1. pick two random elements $g_1, g_2 \in_U G$;
2. pick five random integers $x_1, x_2, y_1, y_2, z \in_U [0, q)$;
3. compute $c \stackrel{\text{def}}{=} g_1^{x_1} g_2^{x_2}$, $d \stackrel{\text{def}}{=} g_1^{y_1} g_2^{y_2}$, $h \stackrel{\text{def}}{=} g_1^z$;
4. choose a cryptographic hash function $H : G^3 \mapsto [0, q)$;
5. publicizes (g_1, g_2, c, d, h, H) as public key, keeps (x_1, x_2, y_1, y_2, z) as private key.

Encryption

To send a confidential message $m \in G$ to Alice, the sender Bob picks random integer $r \in_U [0, q)$ and computes

$$u_1 \stackrel{\text{def}}{=} g_1^r, \quad u_2 \stackrel{\text{def}}{=} g_2^r, \quad e \stackrel{\text{def}}{=} h^r m, \quad \alpha \stackrel{\text{def}}{=} H(u_1, u_2, e), \quad v \stackrel{\text{def}}{=} c^r d^{r\alpha}.$$

The ciphertext is (u_1, u_2, e, v) .

Decryption

To decrypt ciphertext (u_1, u_2, e, v) , Alice performs the following steps:

1. $\alpha \stackrel{\text{def}}{=} H(u_1, u_2, e)$;
2. output $\begin{cases} m \stackrel{\text{def}}{=} e/u_1^z & \text{if } u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = v \\ \text{REJECT} & \text{otherwise} \end{cases}$

Figure 14.4.

Diffie-Hellman Assumption (DDHA, Assumption 13.2). The reader may like to review Definition 12.1 for the Decisional Diffie-Hellman Problem (DDHP).

The Cramer-Shoup cryptosystem is specified in Algorithm 14.1

It is easy to see that part of the ciphertext (u_1, e) is exactly the ciphertext pair of the semantically secure ElGamal cryptosystem. By Theorem 13.2, we already know that the Cramer-Shoup scheme is IND-CPA secure under the DDHA.

Like any CCA2-secure encryption scheme, the decryption procedure has a data-integrity validating step in the decryption procedure. Suppose that the ciphertext has not been altered en route to Alice. Then we have

$$u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = (u_1^{x_1} u_2^{x_2})(u_1^{y_1\alpha} u_2^{y_2\alpha}) = (g_1^{rx_1} g_2^{rx_2})(g_1^{ry_1\alpha} g_2^{ry_2\alpha}) = c^r d^{r\alpha} = v.$$

Upon passing this data-integrity validating step, the rest of the decryption procedure follows that of the semantically secure ElGamal cryptosystem. Later we shall see that this data-integrity validating step is very effective: it virtually stops any hope to construct a valid ciphertext without following the specified encryption procedure.

A reader might want to ask the following question:

“Why is the security of the scheme based solely on the DDHA? Since the data-integrity validating step uses a hash function H , why is the scheme’s security not also based on some hash function property, e.g., the random oracle property?”

Of course, the hash function used in the scheme must not be a weak one. However, we should notice that the security service used in the data-integrity validating step is solely the one-way-ness of the hash function. There is no need to use the random oracle property. For example, hash function $H(x)$ can be implemented by g^x in the same group G , and thereby we will only use the one-way-ness of the discrete logarithm problem (the DLP, see Definition 8.2). The associated intractability assumption is the discrete logarithm assumption (the DLA, see Assumption 8.2) which is not only standard, but also is weaker than the DDHA in the same group; that is, if we use the DDHA in G , the DLA must also be in place in G . It is from this point of view, we say that the security of the scheme can be solely based on the DDHA. In contrast, as we have witnessed in Shoup’s attack on an f -OAEP (§14.2.4.3), a security proof for f -OAEP cannot solely be based on the one-way-ness property, be it that of the hash functions used in the OAEP construction, or that of the underlying intractability.

14.3.2.1 The performance

At first glance, it appears that the Cramer-Shoup cryptosystem is associated with much larger keys and many more exponentiations in comparison with the ElGamal cryptosystem. However, a closer examination will reveal that the difference is not so substantial.

A public key of the scheme consists of five elements in G , increased from a two-element public key in the case of ElGamal. The size of a ciphertext is a quadruple

in G , doubling the size of that of ElGamal. Encryption requires “five” (but in fact, four, see in a moment) exponentiations, increased from two exponentiations in the case of ElGamal. Decryption requires “three” (in fact two) exponentiations, increased from one exponentiation in the case of ElGamal.

Now let us explain why encryption (decryption) only needs four (two) exponentiations instead of five (three) of them as obviously specified in the scheme. This is because the product of two exponentiations in the formulation of $g^x h^y$ can be computed at the cost of a single exponentiation. Algorithm 14.2 specifies this method. It is easy to see that the algorithm terminates in $\max(|x|, |y|)$ steps of the well-known “square-and-multiply” operation and outputs the correct result. Notice that this algorithm and in fact throughout our introduction to the Cramer-Shoup scheme, we have been omitting the presentation of the group operation. Indeed, G can be any abelian group in which the DDHA holds.

Our examination on the performance of the Cramer-Shoup cryptosystem concludes the following result: in both communication bandwidth and computation, the overhead of the Cramer-Shoup cryptosystem is roughly twice that of the ElGamal cryptosystem.

14.3.3 Proof of Security

The reader should be relaxed to know that there is no need of any advanced mathematical knowledge in order to understand the technique for the proof of security for the Cramer-Shoup cryptosystem. A very basic understanding of the group theory which we have introduced in §5.1 plus an elementary fact in linear algebra (we shall state the fact when it is used) will suffice.

Proof of security for the Cramer-Shoup cryptosystem follows the “reduction to contradiction” methodology for formally provable security: “reducing” a hard problem supported by the underlying intractability assumption to an alleged IND-CCA2 attack. In the Cramer-Shoup cryptosystem, the hard problem is the following one:

Let G be a group of a prime order q , and let $(g_1, g_2, u_1, u_2) \in G^4$ be an arbitrary quadruple with $g_1 \neq 1, g_2 \neq 1$. Answer the following question: Is (g_1, g_2, u_1, u_2) a Diffie-Hellman quadruple? That is, whether or not existing integers $a, b \in [0, q)$ such that

$$g_2 = g_1^a, \quad u_1 = g_1^b, \quad u_2 = g_1^{ab}? \quad (14.3.1)$$

Since G is of prime order, $g_1 \neq 1$ is a generator of G (Corollary 5.3) and hence there always exists integers $a, b \in [0, q)$ to satisfy the first two equations in (14.3.1). That is why we have only put the question mark of the third equation. It is routine to check that holding of the three equations in (14.3.1) is equivalent to

$$\log_{g_1} u_1 = \log_{g_2} u_2 \pmod{q}.$$

Algorithm 14.2: Product of Exponentiations

INPUT $g, h \in A$, where A is an algebraic structure;
 x, y : integers in interval $(0, \#A)$;
 $\text{Exp}(u, z)$: single exponentiation which returns u^z ;
 (* e.g., using Algorithm 4.3 for Exp *)

OUTPUT $g^x h^y$.

1. if ($|x| > |y|$)
 - {
 - $u \leftarrow \text{Exp}(g, x \bmod 2^{|x|-|y|})$;
 - (* exponentiation uses the least $|x| - |y|$ significant bits of x *)
 - $x \leftarrow x \div 2^{|x|-|y|}$ (* “ \div ”: division in integers; the operation chops the least significant $|x| - |y|$ bits off x , and hence now $|x| = |y|$ *)
 - }
2. if ($|y| > |x|$)
 - {
 - $u \leftarrow \text{Exp}(h, y \bmod 2^{|y|-|x|})$;
 - $y \leftarrow y \div 2^{|y|-|x|}$
 - }
3. $v \leftarrow gh$; (* below $|x| = |y|$ *)
4. while ($x \neq 0$) do
 - {
 - (a) $u \leftarrow u^2$;
 - (b) if ($x \bmod 2 == 1 \wedge y \bmod 2 == 1$) $u \leftarrow uv$;
 - (c) if ($x \bmod 2 == 1 \wedge y \bmod 2 == 0$) $u \leftarrow ug$;
 - (d) if ($x \bmod 2 == 0 \wedge y \bmod 2 == 1$) $u \leftarrow uh$;
 - (e) $x \leftarrow x \div 2$; $y \leftarrow y \div 2$; (* throw away the least significant bit *)
 - }
5. return(u).

(* total number of “square-and-multiply”: $\max(|x|, |y|)$ *)

Figure 14.5.

By the Decisional Diffie-Hellman Assumption (Assumption 13.2), this question is a hard problem for the general case of an abelian group.

14.3.3.1 A top-level description for the security proof technique

Suppose there exists an attacker \mathcal{A} who can break the the Cramer-Shoup cryptosystem in the IND-CCA2 mode with a non-negligible advantage. We shall construct an efficient reduction algorithm to enable our special agent, Simon Simulator, to answer a Decisional Diffie-Hellman question.

The input to Simon is an *arbitrary* quadruple $(g_1, g_2, u_1, u_2) \in G^4$ where $g_1 \neq 1$ and $g_2 \neq 1$. This quadruple may be a Diffie-Hellman quadruple, if this is the case we denote $(g_1, g_2, u_1, u_2) \in \mathbf{D}$; or it may not be a Diffie-Hellman quadruple, if this is the case we denote $(g_1, g_2, u_1, u_2) \notin \mathbf{D}$.

Using the input values, Simon can construct a public key $PK = (g_1, g_2, c, d, h, H)$ for \mathcal{A} to use, and during the IND-CCA2 attack game played with \mathcal{A} , Simon can also, upon \mathcal{A} 's request, construct a challenge ciphertext $C^* = (u_1, u_2, e, v)$ which encrypts a chosen plaintext $m_b \in_U \{m_0, m_1\}$ (m_0, m_1 are chosen by \mathcal{A} , but the bit b is hidden from \mathcal{A}).

The challenge ciphertext C^* has the following two properties:

- i) If $(g_1, g_2, u_1, u_2) \in \mathbf{D}$, then C^* is a valid Cramer-Shoup ciphertext which encrypts m_b under the public key PK . We shall see the validity of C^* in §14.3.3.3 and §14.3.3.4. Also, whether using the given public key or not, \mathcal{A} can get cryptanalysis training courses which will be precisely simulated by Simon. We shall see the exact precision of the simulated cryptanalysis training courses in §14.3.3.5. So in this case, Simon asks \mathcal{A} to release its attacking advantage to the full capacity.
- ii) If $(g_1, g_2, u_1, u_2) \notin \mathbf{D}$, then the challenge ciphertext C^* encrypts m_b in Shannon's information-theoretical security sense (i.e., perfect encryption, see §??), that is, the ciphertext will be uniformly distributed in the entire ciphertext space. We shall see Shannon's perfect encryption in §14.3.3.4. Moreover, we shall also see in §14.3.3.5 that the quality of Shannon's perfect encryption cannot be compromised by the cryptanalysis training courses delivered to \mathcal{A} . So in this case, \mathcal{A} cannot have any advantage whatsoever!

It is the difference in the respective advantages in these two cases that makes \mathcal{A} a good teacher for Simon to answer the Decisional Diffie-Hellman question.

Let us now construct a "reduction to contradiction".

14.3.3.2 The reduction

The reduction involves the following steps:

1. On input $(g_1, g_2, u_1, u_2) \in G^4$, Simon will construct a public key for the Cramer-Shoup cryptosystem, and send this public key to \mathcal{A} ; the method for the public key construction will be described in §14.3.3.3;
2. Simon will provide \mathcal{A} with needed pre-challenge cryptanalysis training course; the method for Simon to simulate \mathcal{O} 's decryption procedure will be described in §14.3.3.5;
3. Simon will receive from \mathcal{A} a pair of chosen plaintext m_0, m_1 , will flip a fair coin $b \in_U \{0, 1\}$, and will encrypt m_b to construct a challenge ciphertext C^* and will send C^* to \mathcal{A} ; the method for Simon to simulate \mathcal{O} 's encryption procedure will be described in §14.3.3.5;
4. Simon will continue providing \mathcal{A} with needed post-challenge cryptanalysis training course by simulating \mathcal{O} 's decryption procedure;
5. Simon will finally receive from \mathcal{A} an educated guess on the bit b ; upon this time, Simon will be able to answer the question whether $(g_1, g_2, u_1, u_2) \in \mathbf{D}$ or $(g_1, g_2, u_1, u_2) \notin \mathbf{D}$.

Figure 14.6 provides an illustration of the reduction.

14.3.3.3 Public key construction

Using the input quadruple $(g_1, g_2, u_1, u_2) \in G^4$, Simon constructs public-key as follows: he picks

$$x_1, x_2, y_1, y_2, z_1, z_2 \in_U [0, q)$$

and computes

$$c \stackrel{\text{def}}{=} g_1^{x_1} g_2^{x_2}, \quad d \stackrel{\text{def}}{=} g_1^{y_1} g_2^{y_2}, \quad h \stackrel{\text{def}}{=} g_1^{z_1} g_2^{z_2}. \quad (14.3.2)$$

Simon also chooses a cryptographic hash function H . The public key for \mathcal{A} to use is

$$(g_1, g_2, c, d, h, H).$$

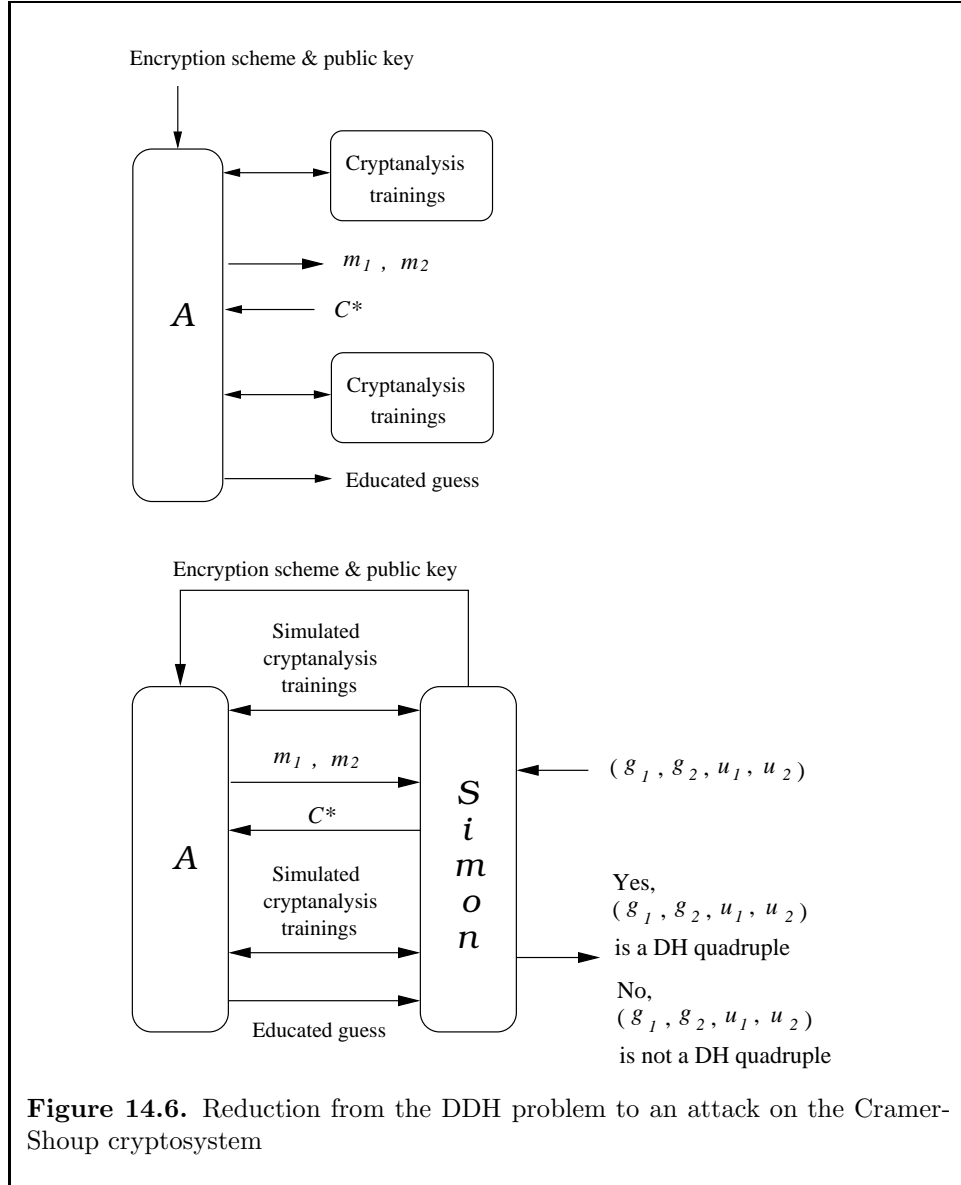
The private key that Simon will use is

$$(x_1, x_2, y_1, y_2, z_1, z_2).$$

The reader may have already noticed that part of the public key, namely the h component, is different from that specified in Algorithm 14.1. Let us explain that this is not a problem. We first show that the h component of the public key constructed by Simon is perfectly valid.

For $h = g_1^{z_1} g_2^{z_2}$ with $g_1 \neq 1$, we note that g_1 is a generator of G (Corollary 5.3) and hence $g_2 = g_1^w$ for some $w \in [0, q)$; thus

$$h = g_1^{z_1 + wz_2} = g_1^z \quad (14.3.3)$$



for $z \equiv z_1 + wz_2 \pmod{q}$. So, indeed, h follows exactly the key-setup procedure in Algorithm 14.1.

The reader might still have the following concern:

Since Simon does not know $w = \log_{g_1} g_2 \pmod{q}$, how can he later use

$z \equiv z_1 + wz_2 \pmod{q}$ in the decryption procedure?

In §14.3.3.5 we shall see that for any ciphertext which is valid with respect to the public key, Simon can indeed correctly use $z \equiv z_1 + wz_2 \pmod{q}$ as the “normal version” of the decryption exponent even he does not have in his possession of z .

14.3.3.4 Simulation of the encryption procedure

Upon receipt of two chosen plaintext messages m_0, m_1 from \mathcal{A} , Simon tosses an unbiased coin $b \in_U \{0, 1\}$, and encrypts m_b as follows:

$$e = u_1^{z_1} u_2^{z_2} m_b, \quad \alpha = H(u_1, u_2, e), \quad v = u_1^{x_1 + y_1 \alpha} u_2^{x_2 + y_2 \alpha}.$$

The challenge ciphertext C^* is (u_1, u_2, e, v) .

The reader may have noticed again that this encryption procedure is different from the normal encryption procedure where e should be computed as

$$e = h^r m_b$$

for some $r \in [0, q)$.

However, this should cause no problem at all. Instead, the difference is vitally important for the proof of security. Let us explain the crux by considering the following two cases:

- i) $(g_1, g_2, u_1, u_2) \in \mathbf{D}$. In this case, because there exists $r \in [0, q)$ such that $u_1 = g_1^r$, $u_2 = g_2^r$, we have

$$u_1^{z_1} u_2^{z_2} = (g_1^r)^{z_1} (g_2^r)^{z_2} = (g_1^{z_1} g_2^{z_2})^r = h^r.$$

So the simulated encryption is exactly a valid Cramer-Shoup encryption under the given public key. This is exactly what we desire for as in this case we want \mathcal{A} to show its attacking advantage in the full capacity.

- ii) $(g_1, g_2, u_1, u_2) \notin \mathbf{D}$. In this case, there exists integers $r_1, r_2 \in [0, q)$ with $r_1 \neq r_2 \pmod{q}$, $u_1 = g_1^{r_1}$ and $u_2 = g_2^{r_2}$. Since $g_1 \neq 1$ is a generator of G , there exists $\log_{g_1} g_2$, $\log_{g_1} h$, $\log_{g_1}(e/m_0)$, and $\log_{g_1}(e/m_1)$.

To make our exposition clearer, we may consider that \mathcal{A} is now (i.e., in this case only) computationally unbounded. Given e in the challenge ciphertext C^* , with its unbounded computational power, \mathcal{A} can see the following two linear equation systems on two unknown integers (z_1, z_2) :

$$\begin{pmatrix} 1 & \log_{g_1} g_2 \\ r_1 & r_2 \log_{g_1} g_2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \log_{g_1} h \\ \log_{g_1}(e/m_0) \end{pmatrix} \pmod{q} \quad (14.3.4)$$

$$\begin{pmatrix} 1 & \log_{g_1} g_2 \\ r_1 & r_2 \log_{g_1} g_2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \log_{g_1} h \\ \log_{g_1}(e/m_1) \end{pmatrix} \pmod{q}. \quad (14.3.5)$$

With $(g_1, g_2, u_1, u_2) \notin \mathbf{D}$, we have $r_1 \neq r_2 \pmod{q}$; also notice $\log_{g_1} g_2 \neq 0 \pmod{q}$ (since $g_2 \neq 1$). So the left-hand side matrix is of the full rank, 2, and so both systems have a unique integer solution for the pair (z_1, z_2) . There is no way for \mathcal{A} to verify which of the two cases is the correct one. Thus, even computationally unbounded, \mathcal{A} can have absolutely no idea whether m_0 or m_1 is encrypted under C^* . So for this case, C^* encrypts m_b in Shannon's information-theoretical security sense, and so \mathcal{A} can have no advantage whatsoever!

We must point out that, so far, the exact Cramer-Shoup encryption in case (i), or Shannon's information-theoretically secure encryption in case (ii) are only true up to the CPA mode. That is, the qualities of the respective cases of the simulated encryption hold if \mathcal{A} is passive. Our attacker is not so feeble! Remember, \mathcal{A} entitles cryptanalysis training courses even after receipt of the challenge ciphertext. For example, if in case (ii) \mathcal{A} can obtain a third linear system in addition to (14.3.4) and (14.3.5), maybe as a result of the CCA2 training course, then we can no longer claim Shannon's information-theoretical security of the encryption.

We shall see in the next section how the qualities of the two cases of the simulated encryption will be maintained throughout the cryptanalysis training courses which an IND-CCA2 attacker enjoys.

14.3.3.5 Simulation of the decryption procedure

Upon receipt of a ciphertext $C = (U_1, U_2, E, V)$ from \mathcal{A} , Simon will first follow the data-integrity validating procedure specified in Algorithm 14.1. If the test yields YES, then the ciphertext is deemed valid. Simon then computes

$$m = E / (U_1^{z_1} U_2^{z_2}), \quad (14.3.6)$$

and returns m to \mathcal{A} as the decryption result. If the test yields NO, then the ciphertext is deemed invalid and Simon will return REJECT as the decryption result.

By Theorem 14.1 (to be stated and proved in a moment), a valid ciphertext $C = (U_1, U_2, E, V)$ implies $(g_1, g_2, U_1, U_2) \in \mathbf{D}$ with probability $1 - \frac{1}{q}$. So there exists $R \in [0, q)$ such that $U_1 = g_1^R, U_2 = g_2^R$. Thus,

$$U_1^{z_1} U_2^{z_2} = g_1^{Rz_1} g_2^{Rz_2} = (g_1^{z_1} g_2^{z_2})^R = h^R. \quad (14.3.7)$$

So we see that the simulated decryption performed by Simon in (14.3.6) is correct, except for a negligible probability $\frac{1}{q}$. This clarifies the doubt we have left over at the

end of §14.3.3.3 on how Simon can properly decrypt without “the normal version” of the private exponent $z \equiv z_1 + z_2 \log_{g_1} g_2 \pmod{q}$.

Simon’s ability to conduct correct decryption for valid ciphertexts permits Simon to offer \mathcal{A} proper cryptanalysis training courses which \mathcal{A} entitles as an IND-CCA2 attacker.

We remain to show that the cryptanalysis training courses will not compromise the perfect qualities for the challenge ciphertext hiding m_b , which we have established in §14.3.3.4.

For any valid ciphertext submitted by \mathcal{A} , the returned decryption result will only confirm \mathcal{A} (14.3.7) in which the integer pair (z_1, z_2) is defined by (U_1, U_2, h^R) *exactly* the same way as the pair is defined by the public key components (g_1, g_2, h) in the third equation in (14.3.2). So no information about z_1, z_2 in addition to what has already been shown in the public key can be obtained by \mathcal{A} . Therefore, if \mathcal{A} submits valid ciphertexts, then the cryptanalysis training courses are useless for it.

In order not to waste the precious cryptanalysis training opportunity, \mathcal{A} must submit ciphertexts such that for $C = (U_1, U_2, E, V)$, it holds $(g_1, g_2, U_1, U_2) \notin \mathbf{D}$. If such a ciphertext passes Simon’s validating step, then a numerical decryption result will be returned to \mathcal{A} and this decryption result might relate to the challenge ciphertext in some way which may only be known to \mathcal{A} . Since \mathcal{A} is supposed to be very clever, we can never be sure, in the case of $(g_1, g_2, U_1, U_2) \notin \mathbf{D}$, how the returned decryption result may relate to the challenge ciphertext. Should \mathcal{A} be confirmed of a hidden relation, then we could no longer claim that the encryption of m_b is exactly under the Cramer-Shoup scheme for case (i), or is information-theoretically secure for case (ii), as we have established in §14.3.3.4 under the CPA mode.

Fortunately, if \mathcal{A} somehow manages to come up with a ciphertext (U_1, U_2, E, V) such that $(g_1, g_2, U_1, U_2) \notin \mathbf{D}$, it will be promptly replied with REJECT. This is due to Theorem 14.1 which we shall state and prove in a very short moment. As we shall see, the rejection probability is at least $1 - \frac{1}{q}$. Notice that in the remaining probability of $\frac{1}{q}$, there is no need for \mathcal{A} to obtain any clue from Simon by taking the trouble to submit a ciphertext; \mathcal{A} can always help itself to guess anything in G correctly with probability $\frac{1}{q}$ since G is only of size q .

Thus, \mathcal{A} cannot use its “cleverness” by submitting bad ciphertexts and hoping not to be rejected.

The probability for constructing a bad ciphertext and escaping rejection can be established as follows.

Theorem 14.1: *Let (g_1, g_2, c, d, h, H) be a public key for the Cramer-Shoup encryption scheme in a group G of a prime order q , where $g_1 \neq 1$ and $g_2 \neq 1$. If $(g_1, g_2, U_1, U_2) \notin \mathbf{D}$, then the successful probability for solving the following problem is bounded by $\frac{1}{q}$ regardless of what algorithm is used:*

Input: *public key* (g_1, g_2, c, d, h, H) , $(U_1, U_2, E) \in G^3$;
Output: $V \in G$: (U_1, U_2, E, V) is a valid ciphertext deemed by the key owner.

Remark 14.1: We have simplified the problem of creating a valid ciphertext as to output the fourth component V from a given triple (U_1, U_2, E) and the public key. Treating V as an input component and outputting any one of the first three ciphertext components is essentially the same problem, however since V is not input to the hash function H , outputting V makes the simplest case. \square

Proof To construct a valid ciphertext from the input values, an algorithm has to output $V \in G$ satisfying

$$U_1^{x_1+y_1\alpha} U_2^{x_2+y_2\alpha} = V \quad (14.3.8)$$

where x_1, y_1, x_2, y_2 is the private key of the owner of the input public key and $\alpha = H(U_1, U_2, E)$.

Since G is of prime order q , $g_1 \neq 1$ is a generator of G (Corollary 5.3). So we can denote $r_1 = \log_{g_1} U_1$, $r_2 = \log_{g_2} U_2$, $w = \log_{g_1} g_2$; there also exists $\log_{g_1} c$, $\log_{g_1} d$ and $\log_{g_1} V$ for any $V \in G$. Combining (14.3.8) with the construction of the public-key components c and d (which have been implicitly verified during the key setup time), we have the following linear system

$$\begin{pmatrix} 1 & 0 & w & 0 \\ 0 & 1 & 0 & w \\ r_1 & r_1\alpha & wr_2 & wr_2\alpha \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \log_{g_1} c \\ \log_{g_1} d \\ \log_{g_1} V \end{pmatrix} \pmod{q}. \quad (14.3.9)$$

Applying Gaussian elimination, the matrix in (14.3.9) is equivalent to

$$\begin{pmatrix} 1 & 0 & w & 0 \\ 0 & 1 & 0 & w \\ 0 & 0 & w(r_2 - r_1) & w\alpha(r_2 - r_1) \end{pmatrix} \pmod{q}. \quad (14.3.10)$$

With $(g_1, g_2, U_1, U_2) \notin \mathbf{D}$, we have $r_1 \neq r_2 \pmod{q}$; also notice $w \neq 0 \pmod{q}$ (since $g_2 \neq 1$ is also a generator of G). So the matrix in (14.3.10) is of the full rank, 3, i.e., the three row vectors are linearly independent. By a simple fact in linear algebra, for any $V \in G$, system (14.3.9) has (non-unique) solutions for $(x_1, y_1, x_2, y_2) \pmod{q}$.

Thus, we have proved that for the input values satisfying the theorem condition, all q elements in G are valid candidates for V , i.e., for the input values, every $V \in G$ makes (U_1, U_2, E, V) a valid ciphertext. However, for the key owner, among these q

possibilities, there is only one single $V \in G$ satisfying her/his choice of the private key components (x_1, y_1, x_2, y_2) . (We should clarify that, even though for any fixed $V \in G$, the integer solutions from system (14.3.9) are not unique, however, it is very clear that any fixed integer tuple (x_1, y_1, x_2, y_2) can only be mapped to a single $V \in G$.) Hence the successful probability for the problem in the theorem statement is established. \square

We have, to this end, completed the security proof for the Cramer-Shoup cryptosystem.

It is easy to observe that the “reduction to contradiction” in this proof is a linear function: \mathcal{A} ’s capability to attack the cryptosystem is identically translated to its capability to distinguish whether or not a given quadruple is in \mathbf{D} . We therefore say that the reductionist proof for the security of the Cramer-Shoup scheme has a tight reduction.

14.4 An Overview of Provably Secure Hybrid Cryptosystems

In §8.7 we have introduced hybrid cryptosystems mainly under efficiency considerations. Hybrid cryptosystems also form a general solution to IND-CCA2 secure and practical public-key encryption. In this section let us conduct an overview of a series of hybrid encryption schemes. As the number of these schemes is not small, we cannot include security proofs; interested reader may study the original papers for details.

A ciphertext output from a hybrid cryptosystem has two components: a **key encapsulation mechanism (KEM)** and a **data encapsulation mechanism (DEM)**. This KEM-DEM pair ciphertext can be written as

$$\text{KEM} \parallel \text{DEM} = \mathcal{E}_{pk}^{\text{asym}}(K) \parallel \mathcal{E}_K^{\text{sym}}(\text{Payload_Message}).$$

Upon receipt of this pair, the receiver should decrypt the KEM block using her/his private key to obtain the ephemeral symmetric key K , and then using this key to decrypt the DEM block to retrieve Payload_Message.

If KEM is output from a provably CCA2 secure asymmetric encryption scheme, then the IND property of the DEM block is then a natural result of the randomness of the ephemeral key. It is not difficult conceive that a KEM-DEM structured hybrid cryptosystem can be IND-CCA2 secure. Indeed, Hybrid schemes in a KEM-DEM structure should be regarded as the most natural approach to public-key encryption with IND-CCA2 security and practical efficiency.

We regard hybrid schemes most natural ones because they can encrypt messages of any length at a low overhead. In applications, data have varied lengths and in most situations have lengths larger than a length fixed by a security parameter in a public-key cryptosystem, e.g., n in the case of the RSA-OAEP or $\log_2(\#G)$ in the case of the Cramer-Shoup. Because public-key cryptosystems have a much

higher overhead than those of a symmetric cryptosystem, it is very likely that in applications a provably secure public-key encryption scheme, such as the RSA-OAEP and the Cramer-Shoup, is only used to form a KEM block in a hybrid scheme, while the encryption of data is done in a series of DEM blocks.

Hybrid encryption schemes include several KEM-DEM schemes proposed by Shoup [Sho01b], a scheme named FO proposed by Fujisaki and Okamoto [FO99b], a scheme named HD-RSA proposed by Pointcheval [Poi99a], a scheme named DHAES proposed by Abdalla, Bellare and Rogaway [ABR98], a variation of the Cramer-Shoup scheme proposed by Shoup [Sho00], and a scheme named REACT proposed by Okamoto and Pointcheval [OP01].

The scheme of Fujisaki and Okamoto takes the following formulation:

$$\mathcal{E}_{pk}^{\text{hybrid}}(m) = \text{KEM} \parallel \text{DEM} = \mathcal{E}_{pk}^{\text{asym}}(\sigma, H(\sigma, m)) \parallel \mathcal{E}_{G(\sigma)}^{\text{sym}}(m),$$

where G, H are **hash functions** (to be introduced in §9.2.1). In this scheme, the decryption result from the KEM block is pair $\sigma, H(\sigma, m)$. The recipient uses σ to “seed” hash function G to obtain a symmetric key $G(\sigma)$; then using it to decrypt the DEM block; finally, the recipient can verify the correctness of the decryption by re-evaluation $H(\sigma, m)$. So this scheme allows the recipient to detect whether the ciphertext has been modified or corrupted en route. The detection of ciphertext alteration is the main technical enabler for a cryptosystem to be secure against active attackers.

The HD-RSA scheme of Pointcheval is based on an intractability problem named **dependent RSA** [Poi99b]: given an RSA ciphertext $A = r^e \pmod{N}$, find $B = (r+1)^e \pmod{N}$. This problem is apparently hard if one cannot find the e -th root of A modulo the composite N (the RSA problem). Then, the KEM block of the HD-RSA scheme is simply $A = r^e \pmod{N}$ for a random $r \in \mathbb{Z}_N^*$. The recipient as the owner of N can of course extract r from A and then construct B . The scheme uses $K = G(B)$ as the symmetric key for the DEM block, as in the hybrid scheme of Shoup and that of Fujisaki-Okamoto.

The DHAES scheme of Abdalla, Bellare and Rogaway [ABR98] is a hybrid scheme where the DEM block also attaches a message authentication code (MAC, see §9.2) as a means for data integrity validation. The two symmetric keys (one for the DEM block and one for the MAC block) are derived from a hash function formulation: $H(g^u, g^{uv})$ where g^u is the KEM block and g^v is the recipient’s public key. Clearly, the owner of the public key g^v can operate the private key v on the KEM block g^u to obtain g^{uv} , and thereby reconstruct $H(g^u, g^{uv})$ for further derivation of the two symmetric keys. Without using the private key v , the problem of decryption seems to be something similar to the computational Diffie-Hellman problem (Definition 8.1). The problem for finding $H(g^u, g^{uv})$ given g^u, g^v is called **Hash Diffie-Hellman (HDH) Problem**.

In DHAES, it is interesting to notice that if g^{uv} is directly used as an encryption multiplier as in the cases of the ElGamal and Cramer-Shoup schemes, then

semantical security will be based on a decisional problem: deciding whether or not $(g, g^u, g^v, g^{uv}(= e/m_b))$ is a Diffie-Hellman quadruple. Now in this hybrid scheme, the use of hash function prevents the easy access to the fourth element in the quadruple, and so the decisional problem seems to have been weakened to a computational problem. Remember, it is desirable to underlie security with intractability assumptions which are as weak as possible. The reader may do an exercise to show that the HDH problem lies in between the CDHP (Definition 8.1) and the DDHP (Definition 12.1). Of course, we must notice that the “weakening of assumption” from the DDHP to the HDH problem is not unconditional: it needs some (hidden from our brief description) assumption on the hash function used. Unfortunately, the hidden assumption should be something very close to a random oracle one.

Shoup’s hybrid scheme [Sho00] is a “weakening of assumption” version for the Cramer-Shoup scheme. In the original Cramer-Shoup scheme (Algorithm 14.1), encryption of message m takes the ElGamal formulation: $h^r m$. In the “weakening of assumption” version in [Sho00], h^r is hidden under a hash function $H(...; h^r)$ to stop the easy testing of the DDHP. The hashed value $H(...; h^r)$ will be used to derive symmetric keys for encoding the DEM block and a data integrity validating mechanism. Shoup uses “hedging with hash” to name his version of “weakening of assumption”.

14.5 Literature Notes on Practical and Provably Secure Public-key Cryptosystems

Damgård originates the work on practical public-key cryptosystems with security resilient to active attackers [Dam92]: thwarting active attackers by including a data-integrity validating procedure in public-key cryptosystems. The method has since then become a general strategy for designing cryptosystems with provable security against active attackers. However Damgård’s schemes (there are two schemes in his original work) are demonstrably insecure in the CCA2 mode (see, e.g., [ZS93b]).

Zheng and Seberry propose practical encryption and digital signature schemes which aim to be secure in the CCA2 mode [ZS93b], [ZS93a]. The general idea in their scheme is to enhance one-way function based textbook public-key schemes (ElGamal based) using hash functions. This is an important idea which is later developed to the random-oracle model for provable security which we have studied in §14.2. The security proof in the IND-CCA2 mode provided in [ZS93a] is based on a non-standard assumption called “sole-samplability of spaces induced by functions” (together with the computational Diffie-Hellman assumption). Soldera discovered that one of the schemes of Zheng and Seberry is actually IND-CCA2 insecure [Sol01], [SSQ02].

In the case of RSA based schemes, upon discovery of the incompleteness in the security proof for the RSA-OAEP (§14.2.4.3), Shoup proposes a modification to OAEP called OAEP+ [Sho01a]. After that, Boneh also proposes modifications to

OAEP, named Simple-OAEP, or SAEP [Bon01]. Recently, Coron et al [CJNP02] propose a yet another padding scheme for RSA named Universal Padding. Essentially, these authors insightfully realize that, with the use of hash functions, data-integrity validation in a padding scheme needn't be based on introducing additional redundancy such as a string of zeros as in the case of the RSA-OAEP. Thus, the plaintext message space is enlarged, which means an improved performance (a better data expansion rate). In addition, Coron et al prove that their Universal Padding scheme is also applicable for signature padding use [CJNP02]. Like the RSA-OAEP, these modified padding schemes are also very efficient, and are provably secure in the IND-CCA2 mode under the random-oracle model. However, because provable IND-CCA2 security for the RSA-OAEP has been re-established (§14.2.4.5) and because the RSA-OAEP has long been the RSA encryption standard, it is not clear whether these new modifications can gain similar momentum as the RSA-OAEP has obtained as the standard for RSA encryption.

Padding techniques for OWTP can result in optimally efficient schemes. However, OWTP is actually a very rare function. RSA and Rabin (over quadratic residues) are probably the only OWTPs among common public-key cryptographic functions. Moreover, the random oracle model based security proof (or argument) for padding schemes have so far fail to derive a tight “reduction to contradiction”. Some researchers consider to devise provably secure schemes for general one-way function based public-key cryptographic functions with more tight reductions for provably security. Some authors devise schemes which extend padding based schemes from OWTP to general one-way trapdoor functions. Many public-key cryptographic functions are not permutations (e.g., the ElGamal function is not a permutation). Therefore such extensions are useful. Fujisaki and Okamoto [FO99a] and Pointcheval [Poi00] propose two such generalized schemes. However, these generalized schemes are not optimally efficient: re-encryption is required in the decryption time as the means to detect errors. Since decryption is often operated in a slow device, such as smart cards, decryption-heavy schemes should be avoided.

Of course, a number of hybrid cryptosystems form a family of provably IND-CCA2 secure and practical public-key encryption schemes. Detailed literature notes of this family have been overviewed in §14.4.

Finally, we should remind the reader that for practical public-key encryption schemes, the data-integrity validating mechanism used in the provably IND-CCA2 secure encryption schemes only provides a security service which we have termed “data integrity without source identification”, or “integrity from Malice” (review §9.4). In most applications in the real world, this notion of security service is inadequate. The common approach to achieving source identification in public-key cryptography is to use digital signatures.

Recently, a novel public-key cryptographic primitive named **signcryption** has emerged. A signcryption scheme combines encryption and signature in one primi-

tive. The motivation of the combination is to achieve efficiency in a provably secure encryption scheme. As this new primitive appeared rather recently (originated by Zheng in 1997 [Zhe97]), researchers have the readiness to apply the provable security strategy in the design of signcryption schemes. We shall study the subject of signcryption in a later chapter.

14.6 Chapter Summary

In this chapter we have described two important public-key cryptosystems which not only have “fit-for-application” security: provably secure under the IND-CCA2 security notion, but also are practically efficient: their efficiency is similar to their “textbook version” counterparts.

In the course of security proof for these cryptosystems, we also introduce and explain several important concepts: random oracle model for security proof (with its limitation discussed), formal proof via reduction to contradiction, and tightness of such a reduction.

We also conduct an overview on various hybrid encryption schemes which combine symmetric and asymmetric encryption techniques and achieve practical public-key cryptosystems.

Finally we provide a literature note to review the development of the subject.

The results provided in this chapter are not only practical schemes, but also strides from previous bit-by-bit based solutions (e.g., those described in the preceding chapter).

FORMAL METHODS FOR AUTHENTICATION PROTOCOLS ANALYSIS

15.1 Introduction

In Chapter 10 we have witnessed a fact that authentication and authenticated key establishment protocols (in this chapter we shall often use authentication protocols to refer to both kinds) are notoriously error prone. These protocols can be flawed in very subtle ways. How to develop authentication protocols so that they are secure is a serious research topic pursued by researchers with different backgrounds; some are cryptographers and mathematicians, others are theoretic computer scientists. It is widely agreed by these researchers that formalism approaches should be taken to the analysis of authentication protocols.

Formalism approaches are a natural extension to informal ones. Formalism can mean many things, ranging over notions such as methodical, mechanical, rule and/or tool supported methods. A formal method usually supports a symbolic system or a description language for modelling and specifying a system's behavior so that the behavior can be captured (i.e., comprehended) and reasoned about by applying logical and mathematical methods in a rigorous manner. Sometimes, a formal method is an expert system which captures human experience or even tries to model human ingenuity. A common characteristic of formal methods is that they take a systematic, sometime an exhaustive, approach to a problem. Therefore, formal methods are particularly suitable for the analysis of complex systems.

In the areas of formal analysis of authentication protocols, we can identify two distinct approaches. One can be referred to as formal reasoning about holding of some desirable, or secure properties; the other can be referred to as systematic search for some undesirable, or dangerous, properties.

In the first approach, a protocol to be analyzed must be very carefully chosen

or designed so that it is already believed or likely to be correct. The analysis tries to establish that the protocol is indeed correct with respect to a set of desirable properties which have also been carefully formalized. Because of the careful chosen protocols to be analyzed, a formal proof is often specially tailored to the target protocol and may hence need to have much human ingenuity involvement, although the proof methodology can be more general. This approach further branches to two schools: a computational school and a symbolic manipulation school. In the former, security properties are defined and measured in terms of probability, and a proof of security or protocol correctness is a mathematician's demonstration of holding of a theorem; the proof often involves a reductionist transformation to a well-believed complexity-theoretic assumption (see Chapters 13 and 14 for the case of provably secure public-key encryption schemes). In the latter school, which consists of theoretic computer scientists in formal methods area, security properties are expressed as a set of abstract symbols which can be manipulated, sometimes by a formal logic system, sometimes by an mechanical tool called a theorem prover, toward a Yes/No result.

The second approach considers that, an authentication protocol, however carefully chosen or designed, or even having gone through a formal proof of correctness (i.e., as a result of the first approach), can still contain error. This is because, "proof of correctness" can only demonstrate that a protocol satisfies a set of *specified* desirable properties; it is still possible that a provably secure protocol can fail if a failure has not been considered in the "proof of security" process. Therefore, in this approach, analysis is in terms of systematic, or exhaustive, search for errors. Formalization of a protocol involves to express the protocol into a (finite) state system which is often composed from sub-state systems of protocol parts run by different principals (including "Malice's part"). An error can be described in general terms, e.g., in the case of secrecy of a message, a bad state can be that the message ends up in Malice's set of knowledge; or in the case of entity authentication, a bad state can be that a wrong identity ends up in the set of an accepted identities of an honest principal. This approach has a close relation with the area of formal analysis of complex systems in theoretic computer science, and hence often applies well developed automatic analysis tools developed there.

In this chapter we shall study these approaches to formal analysis of authentication protocols.

15.1.1 Chapter Outline

The technical part of the chapter starts in formalization of protocol specifications; in §15.2 we shall study a refinement approach to authentication protocols specification. After the specification topic, we shall concentrate on analysis techniques. In §15.3 we shall introduce a proof technique based on a computational model for protocol correctness where a proof is in terms of mathematician's demonstration of holding of a theorem. In §15.4 we shall introduce techniques for arguing protocol security

by manipulation of abstract symbols; a logical approach and a theorem proving approach will be introduced there. In §15.5 we shall introduce formal analysis techniques which model a protocol as a finite-state system and search for system errors. Finally, in §?? we shall summarize advantages and drawbacks of these methods.

15.2 Toward Formal Specification of Authentication Protocols

Let us begin the technical part of this chapter by providing an evidence on a need for a more formalized specification means for authentication protocols. Specification should be an indispensable component in any formal methods for the analysis of complex systems. In the case of complex systems being authentication protocols, we consider that the area of study needs a more precised description on the use of cryptographic transformations.

As we have seen in Chapters 2 and 10, many authentication protocols are designed by solely using encryption, and for this reason, a widely agreed notation for expressing the use of encryption in these protocols is $\{M\}_K$. This notation denotes a piece of ciphertext: its sender must perform encryption to create it while its the receiver, has to perform decryption in order to extract M from it. It is the demonstration of these cryptographic capabilities to the communication partners that prove a principal holding of a secret key and hence prove the holder's identity.

Thus, it seems that the idea of authentication achieved by using encryption is simple enough; there should be no much subtlety here.

However in fact, the simple idea of achieving authentication using cryptographic transformation is often misused. The misuse is responsible for many protocol flaws. In this section, we shall first identify a popular misuse of encryption in authentication protocols; then we shall propose an authentication protocol design method based on a refined specification on the use of cryptographic transformations.

15.2.1 Imprecision of Encryption-decryption Approach for Authentication

In §10.4.1.5 we have listed two “non-standard” mechanisms for construction authentication protocols using encryption. In those mechanisms, a sender generates ciphertext $\{M\}_K$ and sends it to an intended receiver; the correct receiver has a secret key to perform decryption, and subsequently can return to the sender a message component extracted from the ciphertext. The returned message component, often containing a freshness identifier, proves to the sender a lively correspondence of the receiver. This achieves authentication of the receiver to the sender.

Let us name these (non-standard) mechanisms “authentication via encryption-decryption” approach.

An often unpronounced security service which implicitly plays the role in the encryption-decryption approach is *confidentiality*, which must be realized using a reversible cryptographic transformation. However, in many cases of authentication protocols where this approach is used, the really needed security service is actually *not* confidentiality, but *data integrity*, which is better realized using some one-way (i.e., non reversible) transformations. That is why we have labelled such cases *misuse* of cryptographic transformations.

When a misuse of cryptographic transformations takes place, there are two undesirable consequences. Let us now discuss them in detail.

15.2.1.1 Harmful

In a challenge-response mechanism for verifying message freshness, the encryption-decryption approach assists an adversary to use a principal to provide a *decryption oracle services* (see §8.3.2). Such a service may grant an un-entitled cryptographic operation to Malice who otherwise cannot perform himself as he does not have in his possession of the correct cryptographic key.

Decryption oracle service forms a major source of tricks for Malice to manipulate protocol messages and defeat the goal of authentication. Lowe's attack on the Needham-Schroeder Public-key Authentication Protocol (Example 2.3) shows exactly such a trick: in the attacking step 1-7, Malice uses Alice's decryption oracle service to decrypt Bob's nonce N_B for him, and is subsequently able to talk to Bob by masquerading as Alice.

Decryption oracle services also provide Malice with valuable information usable for cryptanalysis, e.g., in chosen-plaintext or chosen-ciphertext attacks. We have seen such tricks in numerous attacking examples in Chapter 13.

The correct cryptographic transformation in a challenge-response based mechanism for a receiver to show a cryptographic credential (possessing the correct key) is for her/him to perform a one-way transformation. In the case of using symmetric cryptographic technique, mechanism 10.4.2 is a more desirable one. If the freshness identifier needs to be kept secret, then mechanism 10.4.1 can be used, however, in that case, Bob should still apply an data-integrity service to protect his ciphertext (reason to be given in §15.2.1.2), which should in fact be achieved using a one-way transformation, that is, the ciphertext in mechanism 10.4.1 still needs a protection based on mechanism 10.4.2. In the case of using asymmetric techniques, mechanism 10.4.3 is standard.

Of course mechanisms 10.4.1 and 10.4.2 also enable the challenger to use the responder to provide an oracle service for creating plaintext-ciphertext pairs:

$$N, \mathcal{E}_K(\dots, N),$$

$$N, \text{MDC}(K, \dots, N),$$

where N is a freshness identifier of the challenger's choice. Nevertheless, considering N being non-secret, providing such a pair can cause far less problem than providing a decryption service.

Moreover, in the second case, the “encryption oracle service” is in fact not in place. Any one-way cryptographic transformation for realizing an MDC has a data compression property (see, e.g., §9.2.1 and §9.2.3 for the data compression property in hash-function based and block-cipher based MDC). The data compression property renders loss of information and that's why the transformation becomes irreversible. The loss of information makes the resultant challenge/response pair to be un-usable in a different context: their usage is fixed as in the context of mechanism 10.4.2; using them in any other context will cause a detectable error.

15.2.1.2 Insufficient

In general, a ciphertext encrypting a confidential message should itself be protected in terms of data integrity. In absence of a data-integrity protection, it seems impossible to prevent an active adversary from manipulating the encrypted messages and defeating the goal of a protocol if the ciphertext is an important protocol message.

Let us now look at this issue using the Needham-Schroeder Symmetric-key Authentication Protocol (the fixed version due to Denning and Sacco, see §2.5.5.1). We assume that the encryption algorithm used in the protocol provides a strong confidentiality protection for any message component inside a ciphertext. However, for the purpose of exposing our point, we shall stipulate that the encryption algorithm does *not* provide any protection in terms of data integrity. This stipulation is not unreasonable. In fact, any encryption algorithm which is not designed to also provide a data-integrity protection can have this feature if the plaintext message contains a sufficient quantity of randomness so that the plaintext extracted from decryption is unrecognizable.

For instance, we may reasonably assume that the encryption algorithm is the AES (§??) with the CBC mode of operation (§7.7.2). The reader may extend our attack to other symmetric encryption algorithms, for example, the one-time pad encryption. We should notice that, regardless of what encryption algorithm is to be used, our attack will not make use any weakness in the algorithm's quality of confidentiality service.

Let us examine the first two steps of the Needham-Schroeder Symmetric-key Authentication Protocol.

1. Alice \rightarrow Trent: $Alice, Bob, N_A$;
2. Trent \rightarrow Alice: $\{N_A, K, Bob, Y\}_{K_{AT}}$;

where $Y = \{Alice, K, T\}_{K_{BT}}$.

Let

$$P_1, P_2, \dots, P_\ell$$

denote the plaintext message blocks for the plaintext message string

$$N_A, K, \text{Bob}, Y.$$

In order for the protocol to suit the needs for general applications, we should reasonably assume that the size of the session key K should be no smaller than the size of one ciphertext block. This is a reasonable assumption since a session key should contain sufficiently many information bits (e.g., for secure keying a block cipher or seeding a stream cipher). The nonce N_A should also be sufficiently large to prevent prediction. Since the nonce N_A starts in P_1 , our assumption on the size of the session key will naturally deduce that the whole plaintext block P_2 will be used solely for containing the session key, or may be P_2 only contains part of the session key.

Notice that although we have related P_2 to K , this is purely for clarity in the exposition; if the session key K is very large, then it may occupy a number of plaintext blocks starting from P_2 . Of course, Malice will know the size of the session key K . Yes, our attack does require Malice to know the size of the plaintext messages and the implementation details. After all, these should not be secret.

Let

$$IV, C_1, C_2, \dots, C_\ell$$

denote the AES-CBC ciphertext blocks corresponding the plaintext blocks P_1, P_2, \dots, P_ℓ (review the CBC mode of operation in §7.7.2). Let further

$$IV', C'_1, C'_2, \dots, C'_\ell$$

be the ciphertext blocks of a previous run of the same protocol between the same pair of principals. Of course, Malice had recorded the old ciphertext blocks.

To attack the protocol in a new run, Malice should intercept the new ciphertext blocks flowing from Trent to Alice:

$$2. \text{ Trent} \rightarrow \text{Malice}(\text{“Alice”}): IV, C_1, C_2, C_3 \dots, C_\ell;$$

Malice should now replace these blocks in the following way:

$$2. \text{ Malice}(\text{“Trent”}) \rightarrow \text{Alice}: IV, C_1, C'_2, C'_3 \dots, C'_\ell;$$

That is, Malice should replace the last $\ell - 1$ ciphertext blocks in the current run with the respective old blocks which he had recorded from an old run of the protocol, and let the manipulated chain of blocks go to Alice as if they were coming from Trent.

The CBC decryption by Alice will return N_A in good order since the decryption result is a function of IV and C_1 . It will return (see “**CBC Decryption**” in §7.7.2)

$$\hat{K} = D_{K_{AT}}(C'_2) \oplus C_1 = K' \oplus C'_1 \oplus C_1$$

as the “new” session key (or the first block of the “new” session key). Here K' is the old session key (or the first block of it) which was distributed in the recorded old run of the protocol. Alice’s decryption of the subsequent ciphertext blocks will return the rest of the $\ell - 1$ plaintext blocks identical to those she had obtained in the old run of the protocol.

Since K' is the old session key, we should not exclude a possibility that Malice may have somehow acquired the old session key already (maybe because Alice or Bob had accidentally disclosed it). Thus, Malice can use \hat{K} (or concatenated with the rest part of the old session key) to talk to Alice by masquerading as Bob.

From this attack we see that, regardless of what Alice may infer from her correct extraction of her freshness identifier N_A , no any other plaintext message returned from Alice’s decryption operation should be regarded as fresh!

There can be numerous ways to implement the encryption-decryption approach in this protocol, each of them may thwart this specific attack, but may be subject to a different attack, as long as the implementation details are not secret to Malice.

Several authentication protocols in two early draft international standard documents [ISO92], [ISO93] follow the wrong idea of CBC realization of encryption providing data-integrity service (general guideline for these protocols to use CBC is documented in [ISO96], [ISO91a]), and of course, these protocols are fatally flawed [MB93], [MB94] as we have demonstrated in this section.

We believe that the correct solution to securing this protocol is to have the ciphertext blocks to be protected under a proper data-integrity service. For example, by applying the message authentication techniques which we have introduced in §9.2.2 and §9.2.3 (manipulation detection code technique). Such a technique essentially is a based on one-way transformation, rather than the encryption-decryption approach.

To this end, we have clearly demonstrated that in the case of authentication protocols applying symmetric cryptographic techniques, the encryption-decryption approach is insufficient for securing authentication protocols.

In authentication using asymmetric cryptographic techniques, the encryption-decryption approach is also insufficient. The Needham-Schroeder Public-key Authentication Protocol (Protocol 2.5) is an example of this approach. Lowe’s attack on that protocol (Example 2.3) provides a clear evidence of the insufficiency. We will see later (§??) that a one-way transformation approach for specifying that protocol will provide a sound fix to that protocol with respect to thwarts Lowe’s attack.

15.2.2 A Refined Specification for Authentication Protocols

In order to specify authentication protocols so that the precisely needed cryptographic services are expressed, Boyd and Mao propose to specify authentication protocols in a more complete manner [MB95]. They take a refinement approach which uses two notations to express the use of cryptographic transformations. Here are the two notations are described:

- $\{M\}_K$ denotes a result of an encryption of the message M using the key K . The security service provided on M is confidentiality: M may only be extracted by a principal who has in possession of K^{-1} which is the decryption key matching K . Notice that the message output from the decryption procedure may *not* be recognizable by the holder of K^{-1} (see §9.3.1.1 for the meaning of a recognizable message).
- $[M]_K$ denotes a result of a one-way transformation of the message M using the key K . The security service provided on M is data integrity with *message source identification* which should use the techniques we have studied in Chapter 9. The message M in $[M]_K$ is not a secret and may be visible from $[M]_K$ even without performing any cryptographic operation. A principal who has in possession of K^{-1} which is the verification key matching K can verify the data-integrity correctness of $[M]_K$ and identify the message source. The verification procedure output YES or NO: in the YES case, $[M]_K$ is deemed to have the correct data integrity and M is deemed to be a recognizable message from the identified source; in the NO case, $[M]_K$ is deemed to have an incorrect data integrity and M is deemed to be unrecognizable.

In practice, $[M]_K$ can be realized by a pair $(M, \text{prf}_K(M))$ where prf_K denotes a keyed pseudo-random function (e.g., a message authentication code in cipher-block-chaining mode of operation, CBC-MAC, see §9.2.3, or a keyed cryptographic hash function, HMAC, see §9.2.2) for the case of symmetric technique realization, or a digital signature algorithm for the case of asymmetric technique realization. These are practically efficient realizations.

The refined notations unifies symmetric and asymmetric cryptographic techniques. In the former case, K and K^{-1} are the same, whereas in the latter case, they are the matching key pair in a public-key cryptographic algorithm.

We should emphasize that the transformation $[M]_K$ not only serves data integrity, but also *message source identification*. If the verification of $[M]_K$ returns YES, then even though the message M may not contain any information about its source, the verifier can identify the correct source based on the verification key in use.

Recently, the importance of coupling ciphertexts (confidentiality service) with data-integrity service becomes more widely adopted. We have seen the idea's general

Protocol 15.1: The Needham-Schroeder Symmetric-key Authentication Protocol in Refined Specification

PREMISE and GOAL: Same as in Protocol 2.4.

1. Alice \rightarrow Trent : $Alice, Bob, N_A$
2. Trent \rightarrow Alice : $[\{K\}_{K_{AT}}, N_A, Alice, Bob]_{K_{AT}},$
 $[\{K\}_{K_{BT}}, T, Alice, Bob]_{K_{BT}}$
3. Alice \rightarrow Bob : $[\{K\}_{K_{BT}}, T, Alice, Bob]_{K_{BT}}$
4. Bob \rightarrow Alice : $[N_B]_K$
5. Alice \rightarrow Bob : $[N_B - 1]_K$

Figure 15.1.

applications in public-key cryptography in Chapter 14. In security community, Aiello et al [ABB⁺02b] use the following notation for refinement of security services:

$$\{M\}_{K_1}^{K_2}.$$

In this notation, the message M is encrypted using key K_1 , as well as protected in data-integrity where the verification key is K_2 .

15.2.3 Examples of Refined Specification for Authentication Protocols

Now let us provide a few examples of authentication protocols specified using the refined notation.

15.2.3.1 The Needham-Schroeder symmetric-key authentication protocol

The first example is the Needham-Schroeder Symmetric-key Authentication Protocol, which is specified in Protocol 15.1.

In the refined specification of the Needham-Schroeder Symmetric-key Authentication Protocol, the need of data integrity service is made explicit. If the messages which Alice and Bob receives from Trent, and those between them have not been altered in the transmission, then the both parties can be sure, after seeing YES output from data-integrity verification, that the session key K has a correct cryptographic association between with their identities and their respective freshness

identifiers. These assure them the correctness of the key sharing parties and the freshness of the session key. It is clear that any unauthorized alteration of the message, such as that we have seen in §15.2.1.2, will be detected.

From this refined specification of the protocol we can see that the confidentiality service is provided at the minimum level: only the session key K is protected in that way. Given the random nature of a key, the minimum use of confidentiality service is desirable because it limits (minimize) the amount of information disclosure which may be useful for cryptanalysis.

15.2.3.2 The Woo-Lam protocol

The second example of a refined protocol specification is that for the Woo-Lam Protocol. The refined specification is given in Protocol 15.2 (cf Protocol 10.2). This example will reveal the effectiveness of using the refined specification to achieve avoiding various attacks. The reasons behind the attacks we have seen on this protocol will become apparent: incorrect cryptographic services implied by the imprecision of the widely agreed notation for expressing the use of encryption in authentication protocols.

We notice that since no message in the Woo-Lam Protocol (Protocol 10.2) is secret, and hence there is no need for providing confidentiality protection on any protocol message. The only needed cryptographic protection in the protocol is data integrity with source identification. Therefore in the refined specification of the protocol we only specify one-way transformation.

Now let us reason that none of the attacks on the original Woo-Lam Protocol or on various “fixes” which we have seen in §10.7 can be applicable to the protocol’s refined version.

First, the parallel-session attack demonstrated in Example 10.6 will no longer work. To see this, let Malice sends to Bob $[N_B]_{K_{MT}}$ in two parallel steps 3 and 3’:

3. Malice(“Alice”) \rightarrow Bob: $[N_B]_{K_{MT}}$

3’. Malice \rightarrow Bob: $[N_B]_{K_{MT}}$

Let us assume that Bob remains inauspicious: he does not check these messages (since they are not meant for him to perform any checking), and simply proceed to send out two messages in two parallel steps 4 and 4’:

4. Bob \rightarrow Trent: $[Alice, N_B, [N_B]_{K_{MT}}]_{K_{BT}}$

4’. Bob \rightarrow Trent: $[Malice, N'_B, [N_B]_{K_{MT}}]_{K_{BT}}$

However, Trent will detect two errors in these two steps. The first error occurs on the verification of the message in step 4: Trent uses K_{AT} to verify $[N_B]_{K_{MT}}$

Protocol 15.2: The Woo-Lam Protocol in Refined Specification

PREMISE and GOAL: Same as in Protocol 10.2;

CONVENTION:

Abort run if any one-way transformation verification returns NO.

1. Alice \rightarrow Bob: $Alice$;
2. Bob \rightarrow Alice: N_B ;
3. Alice \rightarrow Bob: $[N_B]_{K_{AT}}$;
4. Bob \rightarrow Trent: $[Alice, N_B, [N_B]_{K_{AT}}]_{K_{BT}}$;
 (* Note: Bob includes N_B in his part of the one-way transformation since *himself* is the source of this freshness identifier. *)
5. Trent \rightarrow Bob: $[N_B]_{K_{BT}}$;
6. Bob accepts if the integrity verification of $[N_B]_{K_{BT}}$ returns YES, and rejects otherwise.

Figure 15.2.

and this of course returns NO, and so the run with step 4 will be aborted. The second error occurs on the verification of the message in step 4': Trent finds that the two one-way transformations use different nonces, and hence this run will have to be aborted too (otherwise, which of the two nonces Trent should return to Bob?)

Finally, it is trivial to see that the reflection attack in Example 10.7 will no longer work on the refined specification too. This is because, if Malice reflects message line 4 in message line 5, the verification step performed by Bob in step 6 will certainly return NO.

Now it is clear that the fundamental reason for the original Woo-Lam Protocol to be flawed is its misuse of cryptographic services.

15.2.3.3 The Needham-Schroeder public-key authentication protocol

Finally, let us look at the refined protocol specification example on a public-key application: the Needham-Schroeder Public-key Authentication Protocol.

Protocol 15.3: The Needham-Schroeder Public-key Authentication Protocol

PREMISE and GOAL: Same as in Protocol 2.5.

1. Alice \rightarrow Bob : $\{N_A, Alice\}_{K_B}$
2. Bob \rightarrow Alice : $\{N_A, N_B\}_{K_A}$
3. Alice \rightarrow Bob : $\{N_B\}_{K_B}$

Figure 15.3.

Following our discussion in §2.5.6.3, the Needham-Schroeder Public-key Authentication Protocol can be presented in three steps of message transitions. This simplified version is specified in Protocol 15.3.

Here, $\{\dots\}_{K_A}$ and $\{\dots\}_{K_B}$ denote the encryption using Alice's and Bob's public keys, respectively, and so, they can only be decrypted by Alice and Bob, respectively. Thus, upon receipt of message line 2, Alice should believe that only Bob can have decrypted message line 1 and subsequently returned her nonce N_A . Likewise, upon receipt of message line 3, Bob should believe that only Alice can have decrypted message line 2 and subsequently returned his nonce N_B . Thus, it is rather reasonable to expect that, upon termination of a run, both parties should have achieved lively identification of the other party and sharing the two nonces.

However, Lowe's attack in Example 2.3 refutes these "reasonable" believes. Now, after our convincing arguments in §15.2.1 against the imprecision of achieving authentication via encryption-decryption approach, we are able to review Lowe's attack on the original protocol from a new angle: it is the missing of the correct cryptographic service that has been the cause of the attack. The attack will disappear if the protocol is specified in a refined precision. Protocol 15.4 specifies one case.

In the refined specification, $[N_A, Alice]_{K_A}$ denotes a message of which the verification of data integrity (and message source identification) should use Alice's public key K_A . Hence, the transformation $[N_A, Alice]_{K_A}$ can be Alice's signature. So the message in step 1 is Alice's nonce, signed by Alice then encrypted under Bob's public key. Likewise, the message in step 2 can be one signed by Bob and then encrypted under Alice's public key.

Because the second message is signed by Bob, Lowe's attack in Example 2.3 can no longer work on the protocol in the refined version. Although Malice can initiate

Protocol 15.4: The Needham-Schroeder Public-key Authentication Protocol in Refined Specification

PREMISE and GOAL: Same as in Protocol 2.5.

1. Alice \rightarrow Bob : $\{[N_A, Alice]_{K_A}\}_{K_B}$
2. Bob \rightarrow Alice : $\{N_A, [N_B]_{K_B}\}_{K_A}$
3. Alice \rightarrow Bob : $\{[N_B]_{K_A}\}_{K_B}$

Figure 15.4.

a run with Bob by masquerading as Alice using Alice's signature (Alice has sent her signature to Malice, and he can decrypt to retrieve the signature and re-encrypt it using Bob's public key), however, now Malice cannot forward Bob's response to Alice as he did in the attack on the original protocol since then Alice will detect an error when she tries to verify Malice's signature.

Although in §2.5.6.4 we have suggested a fix for Lowe's attack on the Needham-Schroeder Public-key Authentication Protocol: adding identities of the intended communication partner in the encrypted message, we now know that that fix is not necessarily correct. Indeed, if the encryption algorithm is malleable (see §13.5.3), then there is no guarantee for the decrypting principal to be sure about the correctness of the identity revealed from decryption. Clearly, the correct fix is to use correct cryptographic services, and the refined protocol specification helps to identify and specify the correct services.

The Needham-Schroeder Public-key Authentication Protocol can also be refined in a different version, encryption-then-sign, as specified in Protocol 15.5.

Again, Lowe's attack on the original protocol won't work on Protocol 15.5: now Malice cannot even initiate a masquerading run with Bob.

15.3 A Computational View of Correct Protocols — the Bellare-Rogaway Model

In Chapters 13 and 14 we have been familiar with the idea of provable security under a computational model which originates from the seminal work of Goldwasser and Micali [GM84a]. There, a security property (one of several confidentiality qualities) is argued under a given attacking scenario (one of several attacking games each of which models, with sufficient generality and precision, one of several typ-

Protocol 15.5: Another Refined Specification of the Needham-Schroeder Public-key Authentication Protocol

PREMISE and GOAL: Same as in Protocol 2.5.

1. Alice \rightarrow Bob : $[\{N_A\}_{K_B}, Alice]_{K_A}$
2. Bob \rightarrow Alice : $[\{N_A, N_B\}_{K_A}]_{K_B}$
3. Alice \rightarrow Bob : $[\{N_B\}_{K_B}]_{K_A}$

Figure 15.5.

ical behaviors of a real-world attacker against public-key encryption schemes). A proof of security for public-key encryption schemes with respect to an alleged attack involves to demonstrate an efficient transformation (called a polynomial-time reduction) leading from the alleged attack to a major breakthrough to a well-believed hard problem in computational complexity. It is the wide belief on the unlikelihood of the major breakthrough that should refute the existence of the alleged attack, that is, a proof is given by contradiction.

Therefore, a formal proof of security under the computational model consists of the following three steps:

- i) Formal modelling of the behavior of protocol participants and that of an attacker: the modelling is usually given in the form of an attacking game played between the attacker and the target.
- ii) Formal definition of security goal: success for the attacker in the attacking game is formulated here, usually in terms of (a non-negligible) probability and (an affordable) time complexity formulations.
- iii) Formal demonstration of a polynomial-time reduction, leading from an alleged attack on a given target to an unlikely breakthrough in the theory of computational complexity; the formal demonstration of the reduction is a mathematician's proof which shows holding of a theorem.

Bellare and Rogaway are the first researchers who initiate a computational-model approach to proof of security for authentication and authenticated key establishment protocols [BR94]. In their seminal work, they model attacks on authentication and authenticated key establishment protocols, design several simple protocols (entity authentication and authenticated key agreement) and then conduct

proofs that their protocols are correct. Their proof leads from an alleged successful attack on a protocol to the collapse of pseudo-randomness, i.e., the output of a pseudo-random function can be distinguished from that of a truly random function by a polynomial-time distinguisher; in other words, the existence of pseudo-random functions is denied.

The reader may now like to review our discussions in §4.10 leading to Assumptions 4.1 and 4.2. These assumptions are the foundations for modern cryptography. They imply that the result of the reduction should either be false or a major breakthrough in the foundations for modern cryptography. As the former is more likely the case, the reduction derives a contradiction as desired.

We shall only introduce the simplest case in the initial work of Bellare and Rogaway: proof of security for a two-party entity authentication protocol based on shared symmetric key [BR94]. Nevertheless, this simple case suffices us to view the working principle of the computational model for proof of protocol correctness.

Our introduction to the original work of Bellare and Rogaway on authentication protocols follows the three steps in the computational model for provable security. In §15.3.1 we formally model the behavior of protocol participants and that of Malice. In §15.3.2 we provide a formal definition on the security goal of mutual entity authentication. In §15.3.3 we demonstrate a proof by reduction to contradiction for a mutual entity authentication protocol.

15.3.1 Formal Modelling of the Behavior of Principals

The protocols considered in [BR94] are two-party ones. Each of the two participants of a protocol has its part as a piece of efficiently executable code with input and output values. A protocol is composed by the communications between these two parts on the input and output values. However we should notice that “communications” here may go through Malice and may be subject to his manipulation of the communicated values.

Thus, we use two steps to describe an abstract protocol: first, an efficiently executable function owned by a protocol participant; and secondly, the composition via communications.

15.3.1.1 Formalization of the protocol part owned by an honest participant

Formally, this part of an abstract protocol is specified by a polynomial-time function Π on the following input values:

- 1^k : The security parameter — $k \in \mathbb{N}$ (review §4.7 and Definition 4.7 for the reason why we write k in the unary representation).

- i : The identity of the principal who owns this part of the protocol; let us call this principal “the owner”; $i \in I$ where I is a set of principals who share the same long-lived key.
- j : The identity of the intended communication partner of the owner; $j \in I$.
- K : The long-lived symmetric key (i.e., the secret input) of the owner; in our case of two-party protocols based on shared symmetric key, K is also the long-lived key for j .
- $conv$: Conversation so far — $conv$ is a bit-string; this string grows with the protocol runs; new string is concatenated to it.
- r : The random input of the owner — the reader may consider r as a nonce created by the owner.

Since $\Pi(1^k, i, j, K, conv, r)$ runs in polynomial-time in the size of its input values (notice that 1^k is of size k), we may consider K , r of size k , and $i, j, conv$ of size polynomial in k .

An execution of $\Pi(1^k, i, j, K, conv, r)$ will yield three values:

- m : The next message to send out — $m \in \{0, 1\}^* \cup \{\text{“no message output”}\}$; this is the public message to be transmitted over the open network to the intended communication partner.
- δ : The decision for the owner — $\delta \in \{\text{Accept, Reject, No-decision}\}$; the owner decides whether to accept or reject or remain undecided regarding the claimed identity of the intended communication partner; an acceptance decision usually does not occur until the end of the protocol run, although a rejection decision may occur at any time. Once a decision other than “No-decision” is reached, the value of δ will no longer change.
- α : The private output to the owner — $\alpha \in \{0, 1\}^* \cup \{\text{“no private output”}\}$; the reader may consider that an agreed session key as a result of an acceptance run is a private output to the owner.

From this formal modelling of a protocol part, we can see that Bellare and Rogaway model entity authentication protocols using important protocol ingredients: cryptographic operations, participants’ identities, freshness identifies, and the conversed messages (review §10.4 for the meanings of these protocol ingredients).

We sometimes use $\Pi(1^k, I)$ to denote an abstract protocol for the entities in set I .

For any given pair $i, j \in I$ (i.e., for two principals who share a long-lived symmetric key) and for $s \in \mathbb{N}$, we denote by $\Pi_{i,j}^s$ to mean player i attempting to

authenticate player j in a session which i considers to be labelled by s . This attempt may be initiated by i , or may be a response to a message from the intended communication peer j . In fact, we shall generally (and conveniently) treat that this attempt is always a response to an *oracle query* made by Malice. This generalization is formulated by formalization of communications.

15.3.1.2 Formalization of communications

Bellare and Rogaway follow the communication model of Dolev and Yao [DY81] (see §2.2): Malice, the attacker, controls the entire communication network.

Given Malice's network observation capability which we have been familiar with in Chapters 2 and 10, Malice can observe a series of $\Pi_{i,j}^s, \Pi_{j,i}^t$ for any given pair $i, j \in I$ (i.e., who share a long-lived symmetric key) even if the executions of these functions are not the making of himself. However, as an active attacker, Malice can do much more than passive observation. He can conduct as many *sessions* as he pleases among the honest principals, and he can persuade a principal (e.g., i) to start a protocol run as if it is run with another honest principal (e.g., j).

For the reason that Malice is a powerful active attacker, we conveniently let Malice own $\Pi_{i,j}^s, \Pi_{j,i}^t$ as *oracles* in a black-box style (for $i, j \in I, s, t \in \mathbb{N}$). This means, Malice can query $\Pi_{i,j}^s$ by supplying i with input values $(i, j, s, conv)$, and he can query $\Pi_{j,i}^t$ likewise. When Malice queries oracle $\Pi_{i,j}^s$ using input $(i, j, s, conv)$, i will add to Malice's input its own secret input K and its random input r , and so $\Pi^s(1^k, i, j, K, conv, r)$ can be executed. After the execution, i will send out an output message m (if there is one), or "no message output", and the decision δ , and will keep the private output α to itself. The outbound output results will of course be obtained by Malice, and so he may proceed his attack further.

Under the Dolev-Yao threat model of communications, before an oracle reaches the "Accept" decision, it always considers that any query it receives is from Malice.

Without loss of generality, it is harmless to consider that there always exists a particularly friendly kind of attacker, called a "benign adversary", who restricts its action to choosing a pair of oracles $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$ and then faithfully conveying each flow from one oracle to the other, with $\Pi_{i,j}^s$ beginning first. In other words, the first query an benign adversary makes is $(i, j, s, "")$ (where $""$ denotes an empty string), generating response $m_1^{(i)}$; the second query he makes is $(j, i, t, m_1^{(i)})$, generating response $m_1^{(j)}$; and so forth, until both oracles reach the "Accept" decision and terminate. Therefore, a benign adversary behaves just like a wire linking between i and j . We shall later see that for a provably secure protocol, Malice's behavior, if he wishes to have the targeted principals to output the "Accept" decision, will be restricted to that of a benign adversary.

In a particular execution of a protocol, Malice's t -th query to an oracle is said to occur at time $\tau = \tau_t \in \mathbb{R}$. For $t < u$, we demand that $\tau_t < \tau_u$.

15.3.2 The Goal of Mutual Authentication: Matching Conversations

Bellare and Rogaway define a notion of **matching conversations** (notice the plurality) as the security goal of mutual entity authentication.

A conversation of oracle $\Pi_{i,j}^s$ is a sequence of timely ordered messages which $\Pi_{i,j}^s$ has sent out (respectively, received), and as consequent responses, received (respectively, sent out). Let $\tau_1 < \tau_2 < \dots < \tau_R$ (for some positive integer R) be a time sequence recorded by $\Pi_{i,j}^s$ when it converses. The conversation can be denoted by the following sequence:

$$\text{conv} = (\tau_1, m_1, m'_1), (\tau_2, m_2, m'_2), \dots, (\tau_R, m_R, m'_R).$$

This sequence encodes that at time τ_1 oracle $\Pi_{i,j}^s$ was asked m_1 and responded with m'_1 ; and then, at some later time $\tau_2 > \tau_1$, the oracle was asked m_2 , and responded with m'_2 ; and so on, until, finally, at time τ_R it was asked m_R , and responded with m'_R .

We should remind the reader that under the Dolev-Yao threat model of communications, oracle $\Pi_{i,j}^s$ should assume that this conversation has been between it and Malice, unless at time τ_R it has reached “Accept” decision. It is convenient to treat as if all conversations are started by Malice. So if $m_1 = \text{“”}$, then we call $\Pi_{i,j}^s$ an initiator oracle, otherwise, we call it a responder oracle.

Let

$$\text{conv} = (\tau_0, \text{“”}, m_1), (\tau_2, m'_1, m_2), (\tau_4, m'_2, m_3), \dots, (\tau_{2t-2}, m'_{t-1}, m_t)$$

be a conversation of oracle $\Pi_{i,j}^s$. We say that oracle $\Pi_{j,i}^t$ has a conversation conv' which matches conv if there exists time sequence $\tau_0 < \tau_1 < \tau_2 < \dots < \tau_R$ and

$$\text{conv}' = (\tau_1, m_1, m'_1), (\tau_3, m_2, m'_2), (\tau_5, m_3, m'_3), \dots, (\tau_{2t-1}, m_t, m'_t),$$

where $m'_t = \text{“no message output”}$. These two conversations are called matching conversations.

Given a protocol Π , if $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$ both always have matching conversations whenever they are allowed to complete a protocol run (again, we remind that, before reaching the “Accept” decision, each of the oracles thinks to have been running with Malice), then it is clear that Malice has not been able to mount any attack more harmful than being a benign adversary, i.e., acting honestly just like a wire.

Now we are ready to formally pronounce what a secure mutual entity authentication protocol is.

Definition 15.1: We say that $\Pi(1^k, \{A, B\})$ is a secure mutual authentication protocol between A and B if the following statement holds except for a negligible probability in k : oracles $\Pi_{A,B}^s$ and $\Pi_{B,A}^t$ both reach the “Accept” decision if and only if they have matching conversations.

When we prove a protocol to be secure using this definition, it is trivial to see that the existence of matching conversations implies acceptance by both oracles since an oracle accepts upon completing its part of the (matching) conversations. The other direction is the non-trivial step: acceptance by both parties implies the existence of matching conversations. Consequently, the goal for Malice in an attack on a protocol is to have both oracles to accept while they do not have matching conversations. Therefore the following definition is more relevant and will be used in our proof of protocol security:

Definition 15.2: *We say that $\Pi(1^k, \{A, B\})$ is a secure mutual authentication protocol between A and B if Malice cannot win with a non-negligible probability. Here Malice wins if $\Pi_{A,B}^s$ and $\Pi_{B,A}^t$ both reach the “Accept” decision while they do not have matching conversations.*

15.3.3 Protocol *MAP1* and its Proof of Security

Bellare and Rogaway demonstrate their formal proof technique by providing a simple mutual entity authentication protocol named *MAP1* (standing for “mutual authentication protocol one”) and conducting its proof of security. *MAP1* is specified in Protocol 15.6.

In this protocol, Alice begins by sending Bob $A\|R_A$ where R_A is her random nonce of length k . Bob responds by making up a nonce challenge R_B of length k and sending back $[B\|A\|R_A\|R_B]_K$. Alice checks whether this message is of the right form and is correctly labelled as coming from Bob. If it is, Alice sends Bob the message $[A\|R_B]_K$ and accepts. Bob then checks whether the final message is of the right form and is correctly labelled as coming from Alice, and, if it is, he accepts.

If Alice and Bob are interfaced with a benign adversary, then with $\tau_0 < \tau_1 < \tau_2 < \tau_3$, Alice will accept with the following conversation:

$$conv_A = (\tau_0, "", A\|R_A), (\tau_2, [B\|A\|R_A\|R_B]_K, [A\|R_B]_K);$$

and Bob will accept with the following conversation

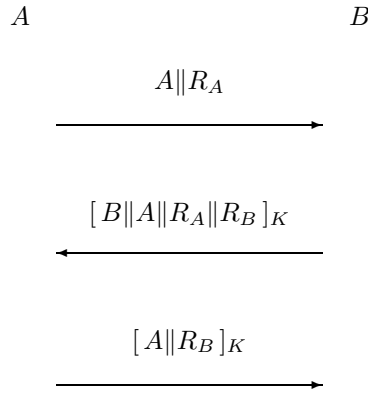
$$conv_B = (\tau_1, A\|R_A, [B\|A\|R_A\|R_B]_K), (\tau_3, [A\|R_B]_K, \text{“no message output”}).$$

Clearly, $conv_A$ and $conv_B$ are two matching conversations.

To prove that *MAP1* is secure when Alice and Bob is interfaced with any kind of polynomially bounded attacker (i.e., Malice), Bellare and Rogaway consider two experiments. In the first experiment, prf_K in *MAP1* is a truly random function, that is, $\text{MAP1}_{A,B}^s$ and $\text{MAP1}_{B,A}^t$ somehow share the function prf_K ; when they apply it to a given input x , the result $\text{prf}_K(x)$ is a bit-string uniformly distributed in $\{0, 1\}^k$. Of course, we must admit that there exists no real-world method to

Protocol 15.6: Protocol *MAP1*

PREMISE: Alice (*A*) and Bob (*B*) share a secret symmetric key K of size k ;
 R_A is Alice's nonce, R_B is Bob's nonce, both are of size k ;
 $[x]_K$ denotes pair $(x, \text{prf}_K(x))$ where $x \in \{0, 1\}^*$ and
 $\text{prf}_K : \{0, 1\}^* \mapsto \{0, 1\}^k$ is a pseudo-random function keyed by K .

**Figure 15.6.**

implement such a shared function. In the second experiment, *MAP1* is realized by a pseudo-random function family just as in the case what will happen in the real world.

Now that in the first experiment, since $\text{prf}_K(x)$ is a k -bit uniform string, when $\text{MAP1}_{A,B}^s$ sees the conversation conv_A , it sees that the uniformly random string $[B||A||R_A||R_B]_K$ is computed using R_A invented by itself; it can therefore conclude that the probability for this bit-string not having been computed by its intended peer (in other words, having been computed by Malice) is at the level of 2^{-k} . This level of the probability value should hold regardless how Malice acts, benign or malicious. Consequently, it can conclude that its intended peer has a conversation which is prefixed by $(\tau_1, A||R_A, [B||A||R_A||R_B]_K)$. This essentially shows to $\text{MAP1}_{A,B}^s$ that there exists a conversation matching conv_A and this conversation has been computed by the intended peer in an overwhelming probability.

Likewise, when $\text{MAP1}_{B,A}^t$ sees conv_B , it can conclude that a conversation matching conv_B must have been produced by its intended peer except for a probability at the level of 2^{-k} .

So if *MAP1* is realized by a truly random and shared function, then Malice

cannot win except for a negligible probability (review Definition 15.2 for the meaning for “Malice wins”).

The remaining part of the proof is an argument by contradiction.

Suppose that in the second experiment, Malice wins in a non-negligible probability. We can construct a polynomial time test T which distinguishes random functions from pseudo-random functions. T receives a function $g : \{0, 1\}^* \mapsto \{0, 1\}^k$ which is chosen according to the following experiment: flip a coin C ; if $C = \text{“Heads”}$ let g be a random function, else K at random and let $g = \text{prf}_K$. T ’s job is to predict C with some advantage non-negligible in k . T ’s strategy is to run MAP1 which is realized by g .

If Malice wins (note that T can tell whether or not Malice wins since T has in its possession of both oracles $\text{MAP1}_{A,B}^s$ and $\text{MAP1}_{B,A}^t$, and hence can see their conversations), then T predicts $C = \text{“Heads”}$ (i.e., g is a pseudo-random function), else T predicts $C = \text{“Tails”}$ (g is truly random). Thus we see that T ’s advantage to make distinction between a k -bit-output random function and a k -bit-output pseudo-random function is similar to Malice’s advantage to win, i.e., non-negligible in k . This contradicts to the common belief that there exists no polynomial-time (in k) distinguisher to make such a distinction (see Assumption 4.2).

In practice, the pseudo-random function prf_K can be practically realized by a message authentication code in cipher-block-chaining mode of operation (CBC-MAC, see §9.2.3) or by a keyed cryptographic hash function (HMAC, see §9.2.2). These are practically efficient realizations.

15.3.4 Further Work in Computational Model for Protocols Correctness

In their seminal paper [BR94], Bellare and Rogaway also consider the correctness for authenticated session key establishment (session key transport) protocols (also for the two-party case). For those protocols, “Malice wins” means either a win under Definition 15.2, or a successful guess of the session key. Since the transported key is encrypted under the shared long-lived key, successful guessing of the key is similarly hard as making distinction between a pseudo-random function and a truly random function.

Later, Bellare and Rogaway extend their initial work to the three-party case: also simple protocols using a trusted third party as an authentication server [BR95b].

Several other authors develop and explore the same approach further: e.g., [BWM98] on key transport protocols, [BWJM97], [BWM99] on key agreement protocols, [BPR00], [BMP00] on password-based protocols, [BCK98], [CK01] on key exchange protocols and design to be correct protocols, and [CK02] on the Internet Key Exchange (IKE) protocols.

15.3.5 Discussion

With the rigorous formulation of matching conversations, Bellare and Rogaway provide a useful formalization for protocol security. Immediately we can see that in order for a principal to reach a meaningful conversation, some silly protocols flaws (design features) which cause reflection attacks (see §10.7.4) and type flaws (see §10.7.6) can be easily eliminated from protocol design. Also, the mathematical analysis can impose, to certain extent, a protocol designer or analyzer to consider using correct or more precise cryptographic services, and so protocol flaws due to misuse of cryptographic services (see §10.7.8) will become less frequent.

It is clear that this proof technique should be working together with protocol design. It guides protocol design, and it only works on correctly designed protocols.

We should remark a limitation in this approach. Definition 15.1, and hence Definition 15.2, do not consider the case of acceptance by one oracle when two oracles do not have matching conversations. In this case, these definitions do not judge whether or not a protocol is secure since we only have one acceptance. However, because the acceptance oracle actually reaches a wrong decision, the protocol should actually be deemed insecure.

Perhaps, the difficulty for covering this case is due to the following problem: for any secure protocol, Malice can always cut the final message and thereby prevents the oracle which should otherwise receive the final message from reaching an acceptance decision. Clearly, such a non-acceptance by one oracle should not turn the protocol to be insecure. While we agree that this is indeed a problem yet to solve, we should also remind that there exists non-trivial authentication failures (i.e., not a result of the non-interesting “dropping the final message attack”) which cannot be handled by Definitions 15.1 and 15.2. The authentication-failure flaw in the IKE Protocol (Protocol 11.1) we demonstrated in Example 11.1 is an example.

Due to the communication nature, provable security for authentication protocols is considerably harder a problem than provable security for a cryptographic algorithm. The approach of Bellare and Rogaway sets out a correct direction. Further work extending their initial promising result is currently an active research topic.

15.4 A Symbolic Manipulation View of Correct Protocols

A symbolic manipulation view of correct authentication protocols is based on formal methods research results conducted by theoretic computer scientists. Under this view, security properties are expressed as a set of abstract symbols which can be manipulated, sometimes by a formal logic system, sometimes by a mechanical tool called a theorem prover, toward a Yes/No result.

15.4.1 Theorem Proving

A theorem proving approach can be described as follows:

- A set of algebraic or logical formulae are defined for use in system behavior description or in statements construction, where statements can be premises (known formulae) or consequences (formulae to be derived);
- A set of axioms are postulated for allowing algebraic or logical ways to derive new formulae from known ones;
- Desired behavior or properties of a system being analyzed are specified as a set of theorems that need to be proved;
- A proof of a theorem is conducted by using premises and applying axioms or proved theorems to reach desired consequences.

Sometimes, the proof process in a theorem proving approach can be mechanized if there are certain rules for applying axioms or theorems. The proof tool is then called a (mechanical) theorem prover. Applying *term rewriting rules* to rewrite a formula to a *normal form* is a standard example for mechanizing a proof. For instance, it is well-known that any boolean expression can be mechanically rewritten to a “conjunctive normal form” (CNF). However, in most cases, a mechanical theorem prover produces long and tedious proofs. It is also a well-known phenomenon that the length of a mechanical proof can be a non-polynomially bounded quantity in the size of the formula being reasoned about (review §4.9 for the meaning of a non-polynomial bounded quantity).

Although a mechanical theorem prover tends to produce impractically large proofs, a theorem proving approach is, nevertheless, capable of dealing with a system whose behavioral description cannot be represented by a finite structure (e.g., a system has an infinite state space). An induction proof of an integer-based mathematical statement is such an example. However, a short-cut proof usually requires the involvement of human ingenuity.

A necessary property in an algebraic-based theorem proving approach is that the axiom system must preserve a so-called **congruence** property. This is a generalization of the congruence relation over integers (defined in §4.5.5) to an arbitrary algebraic structure. A binary relation R over an algebraic structure is said to be a congruence if for every dyadic operation \circ of the structure, whenever

$$R(x, u) \text{ and } R(y, v)$$

then

$$R(x \circ y, u \circ v).$$

The congruence property is also referred to as the substitution property or substitutability. With substitutability, a system component can be substituted by

another one of a related behavior while the desired system behavior can be consistently maintained according to the relation between the substituted components. A theorem proving system will not be regarded to be a sound system if it does not preserve substitutability. Therefore, substitutability is also referred to as the *soundness* property of a theorem proving system. An unsound “theorem proving” should be useless because it is capable of producing an inconsistent statement, for example, a nonsense statement “ $1 = 2$ ”.

The *completeness* property of a theorem proving system refers to a sufficiency status of its axiom system with respect to a notion of “semantic validity”. Basically, if a theorem proving system is complete, then any semantically valid statement must be provable, that is, there must exist a sequence of axioms applications which demonstrates *syntactically* the validity of the statement. The completeness property is desirable but is generally missing from a mechanical theorem prover.

A theorem proving approach aims to demonstrate some desired properties of a system, rather than to find errors in a system. This is because usually an undesirable property cannot be formulated into a theorem. Nevertheless, failure for demonstrating a desired property by a theorem proving system may often result in some insightful ideas leading to a revelation of a hidden error.

Authentication protocols are extremely error-prone systems. In general, it is difficult to come up with a secure protocol in the first place and then to demonstrate its security using a theorem proving approach. Therefore, a theorem proving approach is less useful than one which is capable of finding flaws directly.

15.4.2 A Logic of Authentication

One of the first attempts at formalizing the notion of protocol correctness is the “Logic of Authentication” proposed by Burrows, Abadi and Needham, named the BAN logic [BAN89]. The BAN logic can be viewed as to take the theorem proving approach. It provides a set of logical formulae to model the basic actions of protocol participants and the meanings of the basic protocol components in an intuitive manner:

- a principal **sees** a message;
- a principal **utters** a message;
- a principal **believes** or **provides jurisdiction** over the truth of a logical statement;
- two (or more) principals **share** a secret (message);
- message encryption;
- message freshness;
- conjunction of logical statements;

- a good shared key: it has never been discovered by Malice and it is fresh.

Its axiom system is also postulated on the basis of intuition. For example, the following rule (named “nonce verification”) captures the freshness requirement in message authentication precisely:

$$\frac{P \text{ believes fresh}(X) \wedge P \text{ believes } (Q \text{ said } X)}{P \text{ believes } (Q \text{ believes } X)} \quad (15.4.1)$$

Here Q believe X in the consequence should be interpreted to mean that principal Q uttered message X *recently*. This axiom should be interpreted as saying: if principal P believes that message X is fresh and that principal Q said X , then he should believe that Q has said X *recently*.

A protocol analysis in the BAN logic starts by formulating a set of premises which are protocol assumptions. Next, the protocol messages are “idealized”. This is a process to transform protocol messages into logical formulae. Then, axioms are applied on the logical formulae with an aim to establish a desired property such as a good-key statement.

As a theorem proving approach, the BAN logic does not have a mechanism for directly finding a flaw in a protocol. However, we should notice that one can conduct a reasoning process in a backward manner: starting from the desired goal and applying the axioms to workout a necessary set of premises. Therefore, the BAN logic has been very successful in uncovering implicit assumptions that were missing from protocol specifications but are actually necessary in order for the statement of the desired goal to hold true. A missing assumption can often lead to a discovery of a flaw. A number of flaws in a number of authentication protocols have been uncovered in the seminal work [BAN89]. However, flaw finding in this way demands highly the (human) analyzer’s experience, insight or even luck.

The procedure for protocol idealization can be an error-prone process. Protocols in the literature are typically specified as a sequence of principals sending/receiving messages. Use of the BAN logic requires that the analyzer translates a protocol into formulae about the message transmitted between principals, so that axioms can be applied. For example, if Trent sends a message containing the key K_{AB} , then the message sending step might need to be converted into

$$T \text{ said } A \xleftrightarrow{K_{AB}} B$$

and this means that the key K_{AB} is a good key for communication between Alice and Bob. This idealization step seems to be a quite straightforward. However, protocol idealization is actually a subtle job. Mao observed that the BAN logic provided a context-free procedure for protocol idealization [Mao95]. For example, the idealization step we have illustrated above is done without considering the context of the protocol. This is in fact a dangerous simplification. Mao argued that the protocol idealization in the BAN logic must be a context-sensitive process.

Another drawback in the original proposal of the BAN logic is its lack of a formal definition for an underlying semantics upon which the soundness of the axiom systems is based. As a result, the logic as a theorem proving system may be considered unsound. As another result, some axioms even lack a meaningful type; for example, in the nonce-verification rule listed in (15.4.1), Q believes X has a type error for most cases of X (a nonce): X is normally not a logical formula even after idealization. (We have made a necessary correction in our interpretation of the meaning of the nonce-verification rule.)

Attempts for providing the underlying semantics and arguing the soundness for the BAN logic have been made [AM91], [SvO94].

Notable extensions to the BAN logic include the GNY logic of Gong, Needham and Yahalom [GNY90], an extension of van Oorschot [vO93] and Kailar's accountability logic [Kai96].

The GNY logic extension includes the notion of recognizability which is based on specifying type information of protocol messages so to prevent type flaws, and the notion of a message being possessed by a principal as a result of inventing or recognizing the messages.

The extension of van Oorschot is to facilitate examination of public-key based authenticated key establishment protocols. The extended logic is used to analyze three Diffie-Hellman based key agreement protocols which includes the STS Protocol (Protocol 10.6 which we have examined in §10.6.1).

In Kailar's extension, Kailar convincingly argued that in e-commerce applications, it is accountability and not belief that is important, and provides a syntax which allows such properties to be expressed and a set of proof rules for verifying them. Similar to the BAN logic, these three extensions lack a formal semantic model for the soundness of these logics.

Nevertheless, the BAN logic is undoubtedly an important seminal work. It has inspired the starting of the formal approaches to the analysis of authentication protocols.

15.5 Formal Analysis Techniques: Finite-State System Approach

Another popular approach to formal analysis of complex systems' behavior models a complex system into a (finite) state system. Properties of a state system can be expressed by some state satisfaction relations. The analysis of the behavior of a system usually involves a state space exploration to check whether or not certain properties will be or will not be satisfied. The methodology is usually called **model checking**.

In general, a model checking can be one for guarding against certain undesirable properties so that they never occur, or one for making sure that certain desirable

properties do eventually occur. The former kind of checking is usually considered for *safety* of a system; while the latter kind of checking, for *liveness* of a system.

It seems that checking in the safety direction is more relevant for model checking technique to be applied to the analysis of authentication protocols.

15.5.1 Model Checking

A model checking approach can be described as follows (we will use some concrete examples in our description):

- The operational behavior of a *finite state* system is modelled by a finite state transition system which can make state transitions by interacting with its environment on a set of events; such a system is called a “labelled transition system” (LTS);
 - for example, our single-tape Turing machine DIV3 given in Example 4.1 is an LTS which can make state transitions by scanning a bit-string on its input tape.
- Each state of an LTS is interpreted mechanically into (or assigned with) a logical formula;
 - for example, each state of the machine DIV3 can be interpreted into a propositional logic statement in $\{0, 1, 2\}$, each stating: “the bit-string scanned so far represents an integer congruent to 0, 1, or 2 modulo 3”, respectively.
- A system property which is the target of an analysis is also explicitly interpreted into a logical formula;
 - for example, a target statement for the machine DIV3 can be “an accepted bit-string is an integer divisible by 3”.
- An LTS is symbolically executed; a symbolic execution produces a “trace”

$$\pi = f_0 e_1 f_1 e_2 \dots e_n f_n$$

where f_0, \dots, f_n are logical formulae and e_1, \dots, e_n are events;

- for example, all bit-strings in all traces accepted by the machine DIV3 forms the language Div3.
- A mechanical procedure can check whether or not a target formula is satisfiable by any formula in any trace; here satisfiability means that the target formula is a logical consequence of a formula in a trace;

- for example, for the machine DIV3, it is mechanically checkable that any terminating trace satisfies the target formula “an accepted bit-string represents an integer divisible by 3”.

We should notice that, unlike in the case of theorem proving where a theorem must be an assertion of a desired goal of a system, in model checking, a target formula can model an *undesirable* property of the system as well as a desirable one. For example,

“Malice knows the newly distributed session key K ”

is a formula modelling an undesirable property for a key distribution protocol specified to run by other principals. In the case of a target formula modelling an undesirable property, the result of a satisfiable checking produces a trace which provides an explicit description of a system error. Therefore, a model checking approach can work in the mode for finding an error in a system.

We should emphasize that checking for flaws will be the main working mode for a model checking technique applied to the analysis of authentication protocols.

15.5.1.1 System composition in model checking

When we design a complex system it is usually easier to build it up from simpler components.

For example, if we want to design a Turing machine which accepts bit-strings divisible by 6 (let's denote the machine by DIV6), a simple method for us to do the job can be to design a machine which accepts even numbers (let's denote such a machine by DIV2) and DIV3 (which has been given in Example 4.1) and then to compose the latter two machines by a “conjunctive composition”. It is quite possible that designing DIV2 and DIV3 is easier than designing DIV6 from scratch.

In this conjunctive composition, DIV2 and DIV3 scan their respective input tapes with the identical content, and the two machines are concurrently synchronized, namely, they make every move at the same time. Let the composed machine accept an input if and only if the both component machines accept the input. Now it is obvious that this composition method does arrive at a correct DIV6 (a concrete realization of such a “conjunctive composition” will be exemplified in §15.5.3).

For a more convincing example, a “disjunctive composition” of DIV2 and DIV3 will produce a machine to accept strings divisible by 2 *or* by 3, i.e., it accepts any number in the following sequence:

$$0, 2, 3, 4, 6, 8, 9, 10, 12, 14, 15, \dots$$

Here, “disjunctive composition” means that the composed machine should terminate with acceptance whenever either DIV2 or DIV3 does. Clearly, designing a

machine with this rather awkward behavior, if not using our composition method, can be rather an awkward job too.

The task for finding flaws in authentication protocols via a model checking approach can also be simplified by a system composition method.

In fact, the problem for finding flaws from an authentication protocol is a process of examining a system which is always larger than the system which represents the specified part of the protocol. A protocol specification, at best, only describes how legitimate protocol participants should act. A successful attack, however, will always describe the behavior of a larger system in which Malice lives “in harmony” with (some of) the legitimate participants (i.e., Malice successfully cheats a principal or uncovers a secret without being detected).

Therefore, in a model checking approach to authentication protocols analysis, not only will the legitimate participants’ roles specified in a protocol be modelled, but also will some typical behavior of Malice be modelled (we will see later how to model the typical behavior of Malice). Each of these modelled component parts is an LTS. They will be composed into a larger LTS which is then checked. A composition operation in a model checking tool often models asynchronous communications between system components. Here, “asynchronous” means that the composed system may make a move as a result of one of the component subsystem making a move. This models the situation that Malice’s moves may be independent from those of the honest protocol participants.

In the very beginning of our introduction to the model checking approach we have emphasized that such an approach is suitable to deal with a finite state systems. Indeed, a model checking technique can only deal with systems which can be modelled into a finite state LTS. In authentication protocols analysis, this limitation requires that Malice is a computationally bounded principal: actions which relates to unbounded computational power will not be considered.

Model checking methods frequently face a “state explosion” problem: a system maps to a large LTS of too many states so that the computing resources cannot cope with. This is problem can be particularly acute when the system being analyzed is a large software or hardware system. Such a system tends to be modelled by a huge state space. Fortunately, under the reasonable assumption that Malice is computationally bounded, most authentication protocols can be modelled by rather small LTSs. Therefore, model checking techniques are particularly suitable for the analysis of authentication protocols.

Now let us provide a brief introduction to two model checking techniques for authentication protocols analysis.

15.5.2 The NRL Protocol Analyzer

Meadows developed a PROLOG based protocol checking tool named the NRL Protocol Analyzer, where “NRL” stands for Naval Research Laboratory of the United States of America [Mea96b].

Like any other methodologies for authentication protocols analysis, the NRL Protocol Analyzer is also based on the Dolev and Yao threat model of communications [DY81] (see §2.2). So Malice is able to observe all message traffic over the network, intercept, read, modify or destroy messages, perform any transformation operations on the intercepted messages (such as encryption or decryption, as long as he has in his possession of the correct keys), and send his messages to other principals by masquerading as some principal. However, Malice’s computational capability is polynomially bounded. therefore there is set of “words” that Malice does not know for granted at the beginning of a protocol run, and these words should remain unknown to him after an execution of the protocol if the protocol is secure. This set of words can be secret messages or cryptographic keys for which a protocol is meant to protect. Let us call this set of words “forbidden words”.

Although as a model checking method, the NRL also has the flavor of a term-rewriting system. Its version of Dolev-Yao thread model is also modified by the phrase term-rewriting as “term-rewriting Dolev-Yao model”. We can think of Malice as manipulating a term-rewriting system. If the goal of Malice is to find out a forbidden word, then the problem of proving a protocol secure becomes a word problem in a term-rewriting system: forbidden words should remain forbidden. Equivalently, the problem of showing a protocol insecure becomes to establish a term-rewriting sequence which demonstrates that some “forbidden words” can become available to Malice.

In the NRL Protocol Analyzer, a protocol is modelled by a “global” finite-state system. The global state system is composed of a few “local” state systems together with some state information for Malice. Each local state system describes an honest principal who participates in the protocol. This way of building up the system behavior follows the standard methodology for constructing complex system from easier-comprehensible components (see §15.5.1.1).

Malice’s involvement in the global state system models how he can generate his knowledge as a result of a protocol execution. Malice’s goal is to generate a “forbidden word”, may be via some way of his involvement in the global state system which models his attempt to cause honest principals to reach certain states that are incompatible with the aimed function of the protocol. Such a state is called an “insecure state”. If a protocol contains a flaw, then an insecure state should be reachable. Under the term-rewriting model, reaching of an insecure state is equivalent to establishing a term-rewriting sequence which demonstrates that some words which should not be available to Malice (i.e., “forbidden words”) can become available to him.

In the NRL Protocol Analyzer a set of state transition rules are defined. A transition rule can be “fired” when some conditions hold and after “firing” of a rule some consequence will occur:

Before a rule can be fired:

- Malice must be assigned with some words;
- the related local states must be associated with some values;

After a rule is fired:

- some words will be output by an honest principal (and hence learnt by Malice);
- the related local states will be associated with some new values.

The words involved in these rules obey a set of term-rewriting rules. Typically, there are three rules used to capture the notion of equality and the fact that encryption and decryption are inverse functions. These rules are:

$$\text{encrypt}(X, \text{decrypt}(X, Y)) \rightarrow Y$$

$$\text{decrypt}(X, \text{encrypt}(X, Y)) \rightarrow Y$$

$$\text{id_check}(X, X) \rightarrow \text{Yes}$$

To perform an analysis, the user of the NRL Protocol Analyzer queries it by presenting it with a description of a state in terms of words known by Malice (i.e., a description of an insecure state). The NRL Protocol Analyzer then searches backward in an attempt to find the initial state of the global state system. This is accomplished naturally in PROLOG by attempting to unify the current state against the right hand side of a term-rewriting rule and thus reducing from the left hand side what the state description for the previous state must be. If the initial state is found, then the system is indeed in secure; otherwise an attempt is made to prove that the insecure state is unreachable by showing that any state that leads to this particular state is also unreachable. This kind of search often leads to an infinite trace where in order for Malice to learn a word A, he must learn word B, and in order to learn word B, he must learn word C, and so on. The Analyzer includes certain features that allow the user to prove lemmas about the unreachable of classes of states. Eventually, the goal is to reduce the state space to one small enough to be examined by exhaustive search to determine whether or not an attack on the protocol is possible.

We should notice that the main algorithm used in the NRL Protocol Analyzer answers a state reachability problem. It is well known that such algorithms are not guaranteed to terminate. Therefore a limit is placed on the number of recursive

calls allowed for some of the checking routines. Using the tool seems to require quite a high level of user expertise in accurately coding the transition rules for a protocol and in specifying insecure state. The tool also has an inherent limitation on being particularly applicable to protocols for key establishment.

The NRL Protocol Analyzer has been used to analyze a number of authentication protocols and has successfully found or demonstrated known flaws in some of them. These protocols include the Needham-Schroeder Public-key Authentication Protocol (Protocol 2.5) [Mea96a] (for the analysis of this protocol Meadows provided a comparison between the analysis using the NRL Protocol Analyzer and Lowe's analysis using the model checker FDR in [Low96]), a "selective broadcast protocol" of Simmons [Mea92], [Sim85], the Tatebayashi-Matsuzaki-Newman protocol [KMM94], the Internet Key Exchange protocol (IKE, see §11.2.3, a reflection attack is found in the signature based "Phase 2" exchange protocol) [HC98], [Mea99] and the Secure Electronic Transaction protocols (SET) [SET97], [MS98] (see §??).

15.5.3 The CSP Approach

CSP stands for *Communicating Sequential Processes* and is mainly the work of Hoare [Hoa78]. The name was later changed to TCSP (Theoretical CSP) [BHR84] after a renovation on its semantics. Finally in [Hoa85] TCSP was renamed back to CSP.

CSP belongs to a family of systems named **process algebra**. It follows an algebraic approach to the construction of an abstract computational structure (see Chapter 5). An algebra, CSP is a language of *terms* upon which a few *operations* are defined. These operations obey an "Closure Axiom" which means that the CSP terms forms a closure upon these operations (review Definitions 5.1, 5.12 and 5.13). Each operation is associated with an operational semantics to provide the meaning for the term constructed. We will see the basic CSP operations and terms in a moment.

15.5.3.1 Actions and events

In CSP, a system is modelled in terms of the actions that it can perform. An action is a finite sequence of events occurring sequentially, which includes a sequence of zero length which models "doing nothing". The set of all possible events (fixed at the beginning of the analysis) is called the alphabet of a process and is denoted Σ . Thus, for any action a , we have $a \in \Sigma^*$. An example of alphabet for several processes to be given in a moment is $\Sigma = \{0, 1\}$, and an example of an action which can be performed by these processes is a bit-string of certain property (to be clear in a moment).

In the case of CSP modelling protocols or communication systems, an action can be an atomic message, or a sequence of messages. If M and N are sequence

- $STOP$ (“inaction”: do nothing);
- $a \rightarrow P$ (“prefix”: perform action a and then behave like P);
- $P \sqcap Q$ (“deterministic choice”: behave like P or Q reactively according to external action occurred in the environment);
- $P \sqcup Q$ (“nondeterministic choice”: behave like P or Q with no clear reason, perhaps due to the weather);
- $/a$ (“concealment”: pay no attention to action a);
- $\mu X.P(x)$ (“recursion”: iterate the behavior of P with X being a variable, meaning: $\mu X.P(x) \stackrel{\text{def}}{=} P(\mu X.P(x))$);
- $P \parallel Q$ (“concurrency”: P and Q communicate evolve together when *both* can perform the *same* action);
- $P \amalg Q$ (“interleaving”: P and Q are composed without communication, meaning: they do not have to perform the same action).

Figure 15.7. The CSP Language

of messages, then $M.N$ is also a sequence of message. Sometimes we can omit the symbol “.” from a sequence of messages without causing trouble in understanding.

15.5.3.2 Processes

Processes are the components of systems. They are the entities that are described using CSP, and they are described in terms of the possible actions that they may engage in. Figure 15.7 lists the most basic CSP processes and their associated operational semantics.

The basic operations in Figure 15.7 are the basic building blocks for constructing a CSP process to model and describe the behavior of a finite state system. With these building blocks and the associated operational semantics, the CSP language

is powerful enough to describe a complex system finite-state system.

For example, our Turing machine DIV3 given in Example 4.1, which is a finite-state system, can be re-specified concisely by a CSP process which is given in Example 15.1. This CSP specification only uses “prefix”, “deterministic choice” and “recursion” operations.

Example 15.1: CSP Specification of DIV3

$$\begin{aligned} \text{DIV3} &\stackrel{\text{def}}{=} S_0; \\ S_0 &\stackrel{\text{def}}{=} (0 \rightarrow S_0) \sqcap (1 \rightarrow S_1) \sqcap (\{\} \rightarrow \text{STOP}); \\ S_1 &\stackrel{\text{def}}{=} (0 \rightarrow S_2) \sqcap (1 \rightarrow S_0); \\ S_2 &\stackrel{\text{def}}{=} (0 \rightarrow S_1) \sqcap (1 \rightarrow S_2) \end{aligned}$$

□

The process DIV3 is defined from a number of sub-processes in a recursive manner. All the sub-processes, except *STOP*, react on events in $\Sigma = \{0, 1\}$;^a *STOP* reacts on nothing to mean termination. It is easy to verify that DIV3 has the following behavior:

$$\text{DIV3} = a \rightarrow \text{STOP} \quad \text{for all } a \in \text{Div3} \cup \{\}$$

where for the meaning of the language Div3, see Example 4.1.

15.5.3.3 Traces

The semantics of a process P is defined to be the set of sequences of events, denoted $\text{traces}(P)$, that it may possibly perform. Examples of traces include $\{\}$ (the empty trace) and 1001 (a possible trace of DIV3).

An operation “.” is defined on two sets of traces T, T' as follows:

$$T.T' = \{tr.tr' \mid tr \in T \wedge tr' \in T'\}$$

where the concatenated sequence $tr.tr'$ has been defined in §15.5.3.1.

15.5.3.4 Analysing processes

A process P satisfies a language (e.g., a specification) L if all of its traces are part of L :

$$P \text{ sat } L \Leftrightarrow \text{traces}(P) \subseteq L.$$

^aIn fact, recall our stipulation in §15.5.3.1 on eliding an atomic action $\{e\}$ into to e ; these subprocesses, except *STOP*, react on atomic action $\{0\}$ and $\{1\}$ in Σ^* ; S_0 also reacts on the empty action $\{\}$ which leads DIV to termination.

A process P refines a process Q if $traces(P) \subseteq traces(Q)$. This means that if Q satisfies L (i.e., $Q \text{ sat } L$) then P will also satisfies it.

For example, $DIV3 \text{ sat } Div3 \cup \{\}$ since $traces(DIV3) = Div3 \cup \{\}$. Moreover, $DIV3$ refines a process which performs all bit-strings. Also, we naturally have that $STOP$ refines $DIV3$ ($STOP$ refines any process). In a moment, we shall see a non-trivial process which refines $DIV3$.

Model-checking techniques allow the refinement relation to be checked mechanically for finite-state processes using the tool named Failures Divergences Refinement [Ros94] (FDR, a product of Formal Systems (Europe) Ltd; for details, visit their web site <http://www.fsel.com/index.html>).

15.5.3.5 System composition in CSP

In §15.5.1.1 we have argued that system composition plays a crucially important role in a model checking technique. In CSP, system composition is achieved in a mechanical manner by applying “concurrency” and “interleaving” operations (these two operations together form CSP’s system composition operation).

We provide here an example to show the power of CSP’s composition operation. In §15.5.1.1 we have suggested a method to realize $DIV6$, a machine accepting bit-strings which are integers divisible by 6. Our method suggested there is to construct it from $DIV2$ and $DIV3$. That construction, although not complex, is only an abstract idea and does not provide a concrete method for achieving $DIV6$.

Now, if we have constructed $DIV2$ and $DIV3$ in CSP, then $DIV6$ can be built *mechanically* by composing the CSP specifications of $DIV2$ and $DIV3$, and the result is a concrete CSP specification for $DIV6$. First, $DIV2$ is simple enough to construct directly, and is given in Example 15.2.

Example 15.2: CSP Specification of $DIV2$

$$\begin{aligned} DIV2 &\stackrel{\text{def}}{=} R_0; \\ R_0 &\stackrel{\text{def}}{=} (0 \rightarrow R_0) \sqcap (1 \rightarrow R_1) \sqcap (\{\} \rightarrow STOP); \\ R_1 &\stackrel{\text{def}}{=} (0 \rightarrow R_0) \sqcap (1 \rightarrow R_1). \end{aligned}$$

□

Now, the mechanical composition of $DIV6$ is achieved by applying CSP’s “concurrency” operation, and is given in Example 15.3.

Example 15.3: CSP Composition Construction of $DIV6$

$$DIV6 \stackrel{\text{def}}{=} DIV2 \parallel DIV3 = R_0 \parallel S_0;$$

$$\begin{aligned}
R_0 \parallel S_0 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_0) \sqcap (1 \rightarrow R_1 \parallel S_1) \sqcap (\{\} \rightarrow STOP \parallel STOP); \\
R_1 \parallel S_1 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_2) \sqcap (1 \rightarrow R_1 \parallel S_0); \\
R_0 \parallel S_2 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_1) \sqcap (1 \rightarrow R_1 \parallel S_2); \\
R_1 \parallel S_0 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_0) \sqcap (1 \rightarrow R_1 \parallel S_1); \\
R_0 \parallel S_1 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_2) \sqcap (1 \rightarrow R_1 \parallel S_0); \\
R_1 \parallel S_2 &\stackrel{\text{def}}{=} (0 \rightarrow R_0 \parallel S_1) \sqcap (1 \rightarrow R_1 \parallel S_2); \\
STOP \parallel STOP &= STOP.
\end{aligned}$$

□

In Example 15.3,

$$STOP \parallel STOP = STOP$$

is a CSP axiom named “absorption axiom” (absorption for \parallel), which is obviously true since both sides of the equation can perform nothing.

The mechanically composed version of DIV6 does realize the machine correctly (the reader may test it with several numerical examples, though such tests do not form a proof for the correctness). The mechanical model checker FDR will be able to confirm that DIV6 refines DIV3, and it refines DIV2. These two confirmations should constitute a proof that DIV6 is indeed a correct realization of the aimed machine.

We can also mechanically construct a machine from DIV2 and DIV3, which accepts integer strings divisible by 2 or 3. This mechanical composition can be achieved by (i) applying “interleaving” operation where “interleaving” is switched to “concurrency” whenever two terms being composed can perform the same action, and (ii) by applying the following “deadlock axiom” in CSP:

$$STOP \parallel P = STOP \parallel P = P \parallel STOP = P \parallel STOP = STOP.$$

The resultant machine will be rather big (will have many states) and therefore we shall not explicitly provide its specification.

15.5.3.6 Analysis of security protocols

The usefulness of the composition operation in CSP makes the CSP particularly suitable for modelling and describing the behavior of concurrency and communication systems. It is this feature of CSP that has inspired some researchers to argue for its suitability for formal analysis of authentication protocols [Ros95], [Sch96], [RS01]. In addition, there exists a model checker tool FDR [Ros94] which is tailored to check CSP processes for refinement relations. Lowe applied the FDR model checker and successfully uncovered a previously unknown error in the Needham-Schroeder Public-key Authentication Protocol [Low96].

Let I be an initial set of available information. Then

- If $m \in I$ then $I \vdash m$
- If $I \vdash m$ and $I \subseteq I'$ then $I' \vdash m$
- If $I \vdash m_1$ and $I \vdash m_2$ then $I \vdash (m_1, m_2)$ (paring)
- If $I \vdash (m_1, m_2)$ then $I \vdash m_1$ and $I \vdash m_2$ (projection).
- If $I \vdash m$ and $I \vdash K \in \mathcal{K}$ then $I \vdash \{m\}_K$ (encryption).
- If $I \vdash \{m\}_K$ and $I \vdash K^{-1} \in \mathcal{K}$ then $I \vdash m$ (decryption).

Figure 15.8. The Entailment Axioms

When analysing the property of confidentiality of messages, an “entailment” relation $I \vdash m$ captures how a piece of information can be derived from available information. Figure 15.8 specifies the entailment axioms for information derivation.

For example, we have

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash K_3$$

and

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash \{K_1\}_{K_2}$$

but not

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash K_1 \quad (\text{false}).$$

The entailment axioms intuitively model how Malice derives information. When Malice is trying to defeat the (confidentiality) goal of a protocol, he can use an initial set of information he has in his possession and some protocol messages which have been sent to the network. By the way of information derivation, I is in principle an infinite set. However, in reality, given a protocol, we can always limit I to a finite set of “interesting” information. Moreover, researchers have taken advantage of the fact that there is not need to actually construct I . It suffices to check $m \in I$ for some finite number of messages.

In summary, in the CSP approach, the use of a mechanical tool is an important element for analysis of system behavior. The mechanical tool applies a set of intuitively defined rules. For example, in system construction, a composition tool can build a larger system from composing smaller components by applying the semantic rules given in Figure 15.7; in process refinement checking, a tool can check trace

relations by applying definition for refinement (see §15.5.3.4), and in information derivation, a tool can apply the entailment axioms in Figure 15.8).

The syntax of CSP, in particular that involves communications among system components, is rather unsuitable for being comfortably comprehended by humans, although it can cause no trouble when being dealt with by a mechanical checker. (We should notice that our mechanical construction of DIV6 in Example 15.3 using CSP's composition operation is *succinct* because it does not involve any communications *between* the DIV2 and DIV3; these two components concurrently communicate with the environment which need not be specified.)

Since authentication protocols involve communications among several principals, it is far from straightforward to present the CSP model for authentication protocols to most readers of this book who are not specialized in the formal methods areas. Interested reader is referred to a textbook by Ryan and Schneider [RS01].

15.6 Reconciling Two Views of Formal Reasoning about Security

Since Chapter 13, we have seen two distinct views of formal reasoning about security.

One view, which we have introduced in Chapter 13 and revisited in §15.3 in this chapter, is based on a detailed computational model that considers issues of complexity and probability, and reasons about security in a “proof to contradiction” style.

The other view, which we have introduced in §15.4 and §15.5 in this chapter, relies on simple but effective formal language approaches, either based on theorem proving techniques or on state system exploration techniques, and reasons about security via symbolic manipulations.

These two views come from two mostly separate communities. An uncomfortable gap between them and, indeed, between the two communities, has long been noticed. A need for bridging the gap has been considered by Abadi and Rogaway [AR02]. They consider that connections between the symbolic view and the computational view should ultimately benefit one another:

- These connections should strengthen the foundations for formal cryptology, and help in elucidating implicit assumptions and gaps in formal methods. They should confirm or improve the relevance of formal proofs about a protocol to concrete instantiations of the protocol, making explicit requirements on the implementations of cryptographic operations.
- Methods for high-level reasoning seem necessary for computational cryptology as it treats increasingly complex systems. Symbolic approaches suggest such high-level reasoning principles, and even permit automated proofs. In addition, some symbolic approaches capture naive but powerful intuitions about

cryptography; a link with those intuitions should increase the appeal and accessibility of computational cryptology.

The initial proposal of Abadi and Rogaway is about providing a computational justification for a formal treatment of encryption. Nevertheless, it provides insight for conducting further gap-bridging work for formal treatments of security for other cryptographic primitives such as signatures, hash functions, authentication or authenticated key distribution protocols.

15.7 Chapter Summary

In this chapter we return to the practically important subject of authentication protocols: formalism techniques for their correctness.

We begin with arguing the need for a refined method for protocol specification, proposing a refined specification method and presenting a few refined protocols to show the effectiveness of the refined method proposed.

We then introduce formal protocol analysis methodologies under a computational view of proving protocol correctness and under a symbolic manipulation view of model checking for protocol errors.

As both views have their advantages and limitations, we provide a discussion on a recent move for finding relations and reconciling conflict between the two views.

Formal analysis of authentication protocols is still a topic in its initial stage of research and investigation. The material covered in this chapter inevitably has this feature too.

BIBLIOGRAPHY

- [ABB⁺02a] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. Keromytis, and O. Reingold. Efficient, DoS-resistant, secure key exchange for Internet protocols. In B. Christianson et al., editor, *Proceedings of Security Protocols, Lecture Notes in Computer Science 2467*, pages 27–39. Springer-Verlag, 2002.
- [ABB⁺02b] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A.D. Keromytis, and O. Reingold. Efficient, DoS-resistant, secure key exchange for Internet protocols. In *To appear in Proceedings of ACM Conference on Computer and Communications Security (ACM-CCS'02)*. ACM Press, November 2002.
- [ABK] R. Anderson, E. Biham, and L. Knudsen. Serpent: A proposal for the advanced encryption standard. available at <http://www.cl.cam.ac.uk/~rja14/serpent.html>.
- [ABR98] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: an encryption scheme based on the Diffie-Hellman problem. Submission to IEEE P1363, 1998, IEEE P1363: Asymmetric Encryption, available at <http://grouper.ieee.org/groups/1363/P1363a/Encryption.html>, 1998.
- [ACGS88] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. *SIAM Journal of Computing*, 17(2):194–209, April 1988.
- [AD00] C. Abrams and A. Drobik. E-business opportunity index — the EU surges ahead. Research Note, Strategic Planning, SPA-10-7786, GartnerGroup RAS Services, 21, July 2000.
- [AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.
- [AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. <http://www.cse.iitk.ac.in/users/manindra/primalty.ps>, August 2002.

- [Alc00] Alctel. Understanding the IPSec protocol suite. White Papers Archive, http://www.ind.alctel.com/library/whitepapers/wp_IPSec.pdf, March 2000.
- [AM91] M. Abadi and Tuttle M.R. A semantics for a logic of authentication (extended abstract). In *Proceedings of Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.
- [AN95] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. Technical Report DEC SRC Technical Report 125, Digital Equipment Corporation, November 1995.
- [And94] R. Anderson. Liability and computer security: Nine principles. In D. Gollmann, editor, *Proceedings of the Third European Symposium on Research in Computer Security (ESORICS 94), Lecture Notes in Computer Science 875*, pages 231–245. Springer-Verlag, 1994.
- [And01] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., 2001.
- [AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, Spring 2002.
- [Bab79] L. Babai. Talk presented at the 21st Annual Symposium on Foundation of Computer Science. San Juan, Puerto Rico, October 1979.
- [BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report SRC Technical Report 39, Digital Equipment Corporation, February 1989.
- [BB89] C. Bennett and G. Brassard. The dawn of a new era for quantum cryptography: the experimental prototype is working! *SIGACT News*, 20:78–82, Fall 1989.
- [BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal of Computing*, 15(2):364–383, May 1986.
- [BCD⁺] C. Burwick, D. Coppersmith, E. D’Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas Jr., L. O’Connor, M. Peyravian, D. Safford, and N. Zunic. MARS - a candidate cipher for AES. available at: <http://www.research.ibm.com/security/mars.html>.
- [BCK98] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. In *Proceedings of the 30th Annual Symposium on the Theory of Computing (STOC’98)*, pages 419–428. ACM Press, 1998.

- [BD99] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0.292}$. In J. Stern, editor, *Advances in Cryptology — Proceedings of EURO-CRYPT'99, Lecture Notes in Computer Science 1592*, pages 1–11. Springer-Verlag, 1999.
- [BDP97] A. Bosselaers, H. Dobbertin, and B. Preneel. The new cryptographic hash function RIPEMD-160. *Dr. Dobbs*, 22(1):24–28, January 1997.
- [BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology — Proceedings of CRYPTO'98, Lecture Notes in Computer Science 1462*, pages 26–45. Springer-Verlag, 1998.
- [Bel96] S.M. Bellare. Problem areas for the IP security protocols. In *Proceedings of the Sixth Usenix UNIX Security Symposium*, pages 1–16, July 1996.
- [BF01] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In J. Killian, editor, *Advances in Cryptology — Proceedings of CRYPTO'01, Lecture Notes in Computer Science 2139*, pages 213–229. Springer-Verlag, 2001.
- [BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Distributed trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.
- [BG85] M. Blum and S. Goldwasser. An *efficient* probabilistic public-key encryption scheme which hides all partial information. In G.R. Blakley and D. Chaum eds, editors, *Advances in Cryptology — Proceedings of CRYPTO'84, Lecture Notes in Computer Science 196*, pages 289–299. Springer-Verlag, 1985.
- [BGH⁺92] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In J. Feigenbaum, editor, *Advances in Cryptology — Proceedings of CRYPTO'91, Lecture Notes in Computer Science 576*, Springer-Verlag, pages 44–61, 1992.
- [BHR84] S.C. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the Association of Computing Machinery*, 31(7):560–599, 1984.
- [BJN00] D. Boneh, A. Joux, and P.Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure (extended abstract). In T. Okamoto, editor, *Advances in Cryptology — Proceedings of ASIACRYPT'00, Lecture Notes in Computer Science 1976*, pages 30–43. Springer-Verlag, 2000.
- [Bla02] M. Blaze. Efficient, DoS-resistant, secure key exchange for Internet protocols (Transcript of Discussion). In B. Christianson et al., editor, *Proceedings of Security Protocols, Lecture Notes in Computer Science 2467*, pages 40–48. Springer-Verlag, 2002.

- [Blu81] M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. In *Proceedings of the 24th IEEE Computer Conference*, pages 133–137, May 1981.
- [BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. In *Proceedings of 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 112–117, 1982.
- [BM90a] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In G. Brassard, editor, *Advances in Cryptology — Proceedings of CRYPTO’89, Lecture Notes in Computer Science 435*, pages 547–557. Springer-Verlag, 1990.
- [BM90b] S.M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *ACM Computer Communication Review*, 20(5):119–132, 1990.
- [BM92] S.M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, 1992.
- [BM93] C. Boyd and W. Mao. On a limitations of BAN logic. In T. Helleseht, editor, *Advances in Cryptology — Proceedings of EUROCRYPT’93, Lecture Notes in Computer Science 765*, pages 240–247. Springer-Verlag, 1993.
- [BMP00] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *Advances in Cryptology — Proceedings of EUROCRYPT’00, Lecture Notes in Computer Science 1807*, pages 156–171. Springer-Verlag, 2000.
- [Bon97] D. Boneh. The decision diffie-hellman problem. In *Proceedings of 3rd Algorithmic Number Theory Symposium, Lecture Notes in Computer Science 1423*, pages 48–63. Springer-Verlag, 1997.
- [Bon99] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, February 1999.
- [Bon01] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In J. Killian, editor, *Advances in Cryptology — Proceedings of CRYPTO’01, Lecture Notes in Computer Science 2139*, pages 275–291. Springer-Verlag, 2001.
- [Boy90] C. Boyd. Hidden assumptions in cryptographic protocols. *IEE Proceedings, Part E*, 137(6):433–436, November 1990.
- [BPR00] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology — Proceedings of EUROCRYPT’00, Lecture Notes in Computer Science 1807*, pages 139–155. Springer-Verlag, 2000.

- [BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, New York, 1993. ACM Press.
- [BR94] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology — Proceedings of CRYPTO'93, Lecture Notes in Computer Science 773*, pages 232–249. Springer-Verlag, 1994.
- [BR95a] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. de Santis, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'94, Lecture Notes in Computer Science 950*, pages 92–111. Springer-Verlag, 1995.
- [BR95b] M. Bellare and P. Rogaway. Provably secure session key distribution — the three party case. In *Proceedings of 27th ACM Symposium on the Theory of Computing*, pages 57–66. ACM Press, 1995.
- [Bra93] S. Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI Technical Report, 1993.
- [BS91] E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4:3–72, 1991.
- [BWJM97] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Proceedings of the sixth IMA International Conference on Cryptography and Coding, Lecture Notes in Computer Science, 1355*, pages 30–45. Springer Verlag, 1997.
- [BWM98] S. Blake-Wilson and A. Menezes. Security proofs for entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proceedings of 1997 Security Protocols Workshop, Lecture Notes in Computer Science 1361*, pages 137–158. Springer Verlag, 1998.
- [BWM99] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman key agreement protocols. In *Proceedings of Selected Areas in Cryptography (SAC'99), Lecture Notes in Computer Science 1556*, pages 339–361. Springer Verlag, 1999.
- [Car94] U. Carlsen. Cryptographic protocol flaws: know your enemy. In *Proceedings of The Computer Security Foundations Workshop VII*, pages 192–200. IEEE Computer Society Press, 1994.
- [CDL⁺00] S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of a 512-bit RSA modulus. In B. Preneel, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'00, Lecture Notes in Computer Science 1807*, pages 1–18. Springer-Verlag, 2000.

- [CG85] B. Chor and O. Goldreich. RSA/Rabin least significant bits are $\frac{1}{2} + \frac{1}{\text{poly}(\log N)}$ secure. In G.T. Blakley and D. Chaum, editors, *Advances in Cryptology — Proceedings of CRYPTO'84, Lecture Notes in Computer Science 196*, pages 303–313. Springer-Verlag, 1985.
- [CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th Annual Symposium on the Theory of Computing (STOC'98)*, pages 209–218. ACM Press, 1998.
- [CGH01] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. A new version of [CGH98], February 2001.
- [Cho85] B. Chor. *Two Issues in Public Key Cryptography, RSA Bit Security and a New Knapsack Type System*. MIT Press, 1985. An ACM Distinguished Dissertation.
- [CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature: version 1.0. Available at <http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz>, November 1997.
- [CJNP02] J.S. Coron, M. Joye, D. Naccache, and P. Paillier. Universal padding schemes for RSA. In M. Yung, editor, *Advances in Cryptology — Proceedings of CRYPTO'02, Lecture Notes in Computer Science 2442*, pages 226–241. Springer-Verlag, 2002.
- [CK01] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'01, Lecture Notes in Computer Science 2045*, pages 453–474. Springer-Verlag, 2001.
- [CK02] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In M. Yung, editor, *Advances in Cryptology — Proceedings of CRYPTO'02, Lecture Notes in Computer Science 2442*, pages 143–161. Springer-Verlag, 2002. Available at <http://eprint.iacr.org>.
- [Coc01] C. Cocks. An identity-based public-key cryptosystem. Seminar in University of Bristol, April 2001.
- [Coh96] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1996. Graduate Texts in Mathematics 138.
- [Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
- [Cop94] D. Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38:243–250, 1994.

- [Cop96] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In U. Maurer, editor, *Advances in Cryptology — Proceedings of EUROCRYPT 96, Lecture Notes in Computer Science 1070*, pages 178–189. Springer-Verlag, 1996.
- [CP93] D. Chaum and T.P. Pedersen. Wallet databases with observers. In E.F. Brickell, editor, *Advances in Cryptology — Proceedings of CRYPTO'92, Lecture Notes in Computer Science 740*, pages 89–105. Springer-Verlag, 1993.
- [CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptology — Proceedings of CRYPTO'98, Lecture Notes in Computer Science 1462*, pages 13–25. Springer-Verlag, 1998.
- [CS99] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *Proceedings of 6th ACM Conference on Computer and Communication Security*. ACM Press, November 1999.
- [DA99] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. Request for Comments: 2246, January 1999.
- [Dam92] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In J. Feigenbaum, editor, *Advances in Cryptology — Proceedings of CRYPTO'91, Lecture Notes in Computer Science 576*, pages 445–456. Springer-Verlag, 1992.
- [DDD⁺83] R.A. DeMillo, G.L. Davida, D.P. Dobkin, M.A. Harrison, and R.J. Lipton. *Applied Cryptology, Cryptographic Protocols, and Computer Security Models*, volume 29. Providence: American Mathematical Society, 1983. Proceedings of Symposia in Applied Mathematics.
- [DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of 23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991. Journal version in *SIAM Journal on Computing*, vol 30, no. 2, 391–437, 2000.
- [Den82] D. Denning. *Cryptography and Data Security*. Addison-Wesley Publishing Company, Inc., 1982.
- [DH76a] W. Diffie and M. Hellman. Multiuser cryptographic techniques. In *Proceedings of AFIPS 1976 NCC*, pages 109–112. AFIPS Press, Montvale, N.J., 1976.
- [DH76b] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, IT-22(6):644–654, 1976.
- [Dif92] W. Diffie. The first ten years of public key cryptography. In G.J. Simmons, editor, *Contemporary Cryptology, the Science of Information Integrity*, pages 135–175. IEEE Press, 1992.

- [DM83] R.A. DeMillo and M.J. Merritt. Protocols for data security. *Computer*, 16(2):39–50, February 1983.
- [DP89] D.W. Davies and W.L. Price. *Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer (second edition)*. John Wiley & Sons, 1989.
- [DR98] J. Daemen and V. Rijmen. AES Proposal: Rijndael. National Institute of Standards and Technology (NIST), available at <http://csrc.nist.gov/encryption/aes/>, October 6 1998.
- [DR02] J. Daemen and V. Rijmen. The Design of Rijndael: AES — the Advanced Encryption Standard, 2002.
- [DS81] D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
- [DS89] D. Davis and R. Swick. Workstation services and Kerberos authentication at Project Athena. XXXX, October 6 1989.
- [DvOW92] W. Diffie, P.C. van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.
- [DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Proceedings of IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.
- [EFL⁺99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Spki certificate theory. Internet Network Working Group Request for Comments: RFC2693, September 1999.
- [EJKW74] A. Evans Jr., W. Kantrowitz, and E. Weiss. A user authentication scheme not requiring secrecy in the computer. *Communications of the ACM*, 17(8):437–442, 1974.
- [ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.
- [Fei74] H. Feistel. Cryptography and computer privacy. *Sci. Am.*, 228(5):15–23, May 1974.
- [FKK96] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL Protocol, Version 3.0. INTERNET-DRAFT, draft-freier-ssl-version3-02.txt, November 1996.
- [FO99a] E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In H. Imai and Y. Zheng, editors, *Public Key Cryptography — Proceedings of PKC’99, Lecture Notes in Computer Science 1560*, pages 53–68. Springer-Verlag, 1999.

- [FO99b] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. Wiener, editor, *Advances in Cryptology — Proceedings of CRYPTO'99, Lecture Notes in Computer Science 1666*, pages 537–554. Springer-Verlag, 1999.
- [FOPS01] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP Is secure under the RSA assumption. In J. Killian, editor, *Advances in Cryptology — Proceedings of CRYPTO'01, Lecture Notes in Computer Science 2139*, pages 260–274. Springer-Verlag, 2001.
- [Fou98] Electronic Frontier Foundation. Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design, 1998. ISBN 1-56592-520-3.
- [FS00] N. Ferguson and B. Schneier. A cryptographic evaluation of IPsec. Counterpane Labs <http://www.counterpane.com/ipsec.pdf>, 2000.
- [Gal01] S. Galbraith. Supersingular curves in cryptography. In C. Boyd, editor, *Advances in Cryptology — Proceedings of ASIACRYPT'01, Lecture Notes in Computer Science 2248*, pages 495–513. Springer-Verlag, 2001.
- [Gau01] C.F. Gauss. *Disquisitiones Arithmeticae*. Translated by A. Arthur and S.J. Clark, 1996, Yale University Press, New Haven, 1801.
- [GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [GM84a] S. Goldwasser and S. Micali. Probabilistic encryption,. *Journal of Computer and System Sciences*, 28:270–299, 1984.
- [GM84b] F.T. Grampp and R.H. Morris. Unix operating system security. *AT&T Bell Laboratories Technical Journal*, 63(8):1649–1672, October 1984.
- [GMT82] S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *Proceedings of 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 134–144, 1982.
- [GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.
- [Gol99] D. Gollmann. *Computer Security*. John Wiley & Sons, Inc., 1999. ISBN: 0-471-97884-2.
- [Gol01] D. Gollmann. Authentication — myths and misconceptions. *Progress in Computer Science and Applied Logic*, 20:203–225, 2001. Birkhäuser Verlag Basel/Switzerland.

- [GS91] K. Gaarder and E. Snekenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3(2):81–98, 1991.
- [Gün90] C.G. Günther. An identity-based key-exchange protocol. In J.-J. Quisquater and J. Vanderwalle, editors, *Advances in Cryptology — Proceedings of EURO-CRYPT’89, Lecture Notes in Computer Science 434*, pages 29–37. Springer-Verlag, 1990.
- [GW96] I. Goldberg and D. Wagner. Randomness and the Netscape Browser, how secure is the World Wide Web? *Dr. Dobbs’s Journal*, pages 66–70, January 1996.
- [Hal94] N.M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994.
- [Hås86] J. Håstad. On using RSA with low exponent in a public key network. In H.C. Williams, editor, *Advances in Cryptology — Proceedings of CRYPTO’85, Lecture Notes in Computer Science 218*, pages 403–408. Springer-Verlag, 1986.
- [HC98] D. Harkins and D. Carrel. The Internet key exchange protocol (IKE). The Internet Engineering Task Force Request For Comments (IETF RFC) 2409, available at <http://www.ietf.org/rfc/rfc2409.txt>, November 1998.
- [Hic95] K.E.B. Hickman. The SSL Protocol. Available at http://www.netscape.com/eng/security/SSL_2.html, February 1995.
- [HJW00] D. Hühnlein, M. Jacobson, and D. Weber. Towards practical non-interactive public key cryptosystems using non-maximal imaginary quadratic orders. In *Proceedings of Selected Areas of Cryptography — SAC 2000, Lecture Notes in Computer Science 2012*, pages 275–287. Springer-Verlag, 2000.
- [Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985. Series in Computer Science.
- [Hof02] P. Hoffman. Features of proposed successors to IKE. INTERNET-DRAFT, draft-ietf-ipsec-soi-features-01.txt, May 2002.
- [ISO91a] ISO/IEC. Information Processing — Modes of operation for an n -bit block cipher algorithm. International Organization for Standardization and International Electro-technical Commission, 1991. 10116.
- [ISO91b] ISO/IEC. Information Technology — Security Techniques — summary of voting on letter ballot No.6, document SC27 N277, CD 9798-3.3 “Entity Authentication Mechanisms” — Part 3: Entity authentication mechanisms using a public key algorithm. International Organization for Standardization and International Electro-technical Commission, October 1991. ISO/IEC JTC 1/SC27 N313.

- [ISO92] ISO/IEC. Information Technology — Security Techniques — Entity Authentication Mechanisms — Part 2: Entity authentication using symmetric techniques. International Organization for Standardization and International Electro-technical Commission, 1992. ISO/IEC JTC 1/SC 27 N489 CD 9798-2, 1992-06-09.
- [ISO93] ISO/IEC. Information Technology — Security Techniques — Entity Authentication Mechanisms — Part 2: Entity authentication using symmetric techniques. International Organization for Standardization and International Electro-technical Commission, 1993. ISO/IEC JTC 1/SC 27 N739 DIS 9798-2, 1993-08-13.
- [ISO96] ISO/IEC. Information Technology — Security Techniques — Entity Authentication — Part 1: General. International Organization for Standardization and International Electro-technical Commission, 1996. ISO/IEC JTC 1/SC 27 DIS 9798-1: 1996 (E).
- [ISO98a] ISO/IEC. Information Technology — Security Techniques — Entity Authentication — Part 2: Mechanisms using symmetric encipherment algorithms. International Organization for Standardization and International Electro-technical Commission, December 1998. ISO/IEC JTC 1/SC 27 N2145 FDIS 9798-2.
- [ISO98b] ISO/IEC. Information Technology — Security Techniques — Entity Authentication — Part 3: Mechanisms using digital signature techniques. International Organization for Standardization and International Electro-technical Commission, October 1998. BS ISO/IEC 9798-3.
- [ISO99] ISO/IEC. Information Technology — Security Techniques — Entity Authentication — Part 4: Mechanisms using a cryptographic check function. International Organization for Standardization and International Electro-technical Commission, April 1999. ISO/IEC JTC 1/SC 27 N2289 FDIS 9798-4.
- [ISO00] ISO/IEC. Information technology — Security techniques — Digital signature schemes giving message recovery — Part 3: Discrete logarithm based mechanisms. International Organization for Standardization and International Electro-technical Commission, April 2000. ISO/IEC JTC 1/SC 27 9796-3.
- [ISO01] ISO/IEC. Information Technology — Security Techniques — Hash Functions — Part 3: Dedicated hash-functions. International Organization for Standardization and International Electro-technical Commission, November 2001. ISO/IEC JTC1, SC27, WG2, Document 1st CD 10118-3.
- [IT93] ITU-T. Rec. x.509 (revised) the directory — authentication framework, 1993. International Telecommunication Union, Geneva, Switzerland (equivalent to ISO/IEC 9594-8:1995.).
- [JC93] Kohl J. and Neuman C. The Kerberos network authentication service (v5). Internet Archive RFC 1510, September 1993.

- [JN01] A. Joux and K. Nguyen. Separating decision diffie-hellman from diffie-hellman in cryptographic groups. Cryptology ePrint Archive, 2001/003, <http://eprint.iacr.org/>, 2001.
- [Kai96] R. Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5):313–328, May 1996.
- [Kau02a] C. Kaufman. Comparison of IKEv2, JFK, and SOI requirements. <http://www.ietf.org/proceedings/02mar/slides/ipsec-1/>, April 2002.
- [Kau02b] C. Kaufman. Internet key exchange (IKEv2) protocol. INTERNET-DRAFT, draft-ietf-ipsec-ikev2-03.txt, <http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-03.txt>, October 2002.
- [KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.
- [Kob87] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(5):203–209, 1987.
- [Koh78] L.M. Kohnfelder. *Towards a Practical Public-key Cryptosystem*. MIT B.S. Thesis, MIT Department of Electrical Engineering, May 1978.
- [KPS02] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World, Second Edition*. Prentice-Hall PTR, 2002.
- [Kra] H. Krawczyk. SIGMA: the ‘SIGn-and-MAC’ approach to authenticated Diffie-Hellman protocols. Available at <http://www.ee.technion.ac.il/~hugo/sigma.html>.
- [Kra86] E. Kranakis. *Primality and Cryptography*. John Wiley & Sons, 1986. Wiley-Teubner Series in Computer Science.
- [Kra96] H. Krawczyk. SKEME: a versatile secure key exchange mechanism for Internet. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, pages 114–127. IEEE Computer Society Press, February 1996.
- [Lam81] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.
- [LeV77] W.J. LeVeque. *Fundamentals of Number Theory*. Dover Publications, Inc., 1977.
- [Lip79] R.J. Lipton. How to cheat at mental poker. Technical Report, Comp. Sci., Dept. Univ. of Calif., Berkeley, Calif., August 1979. (This is an internal technical report; a simple description of the attack is available in page 174 of [DDD⁺83]).

- [LN97] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 1997. Encyclopedia of Mathematics and its Applications 20.
- [Low94] G. Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society Press, June 1994.
- [Low95] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
- [Low96] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using csp and fdr. In *Proceedings of TACAS, Lecture Notes in Computer Science 1055*, pages 147–166. Springer-Verlag, 1996.
- [LV00] A. Lenstra and E. Verheul. The XTR public key system. In M. Bellare, editor, *Advances in Cryptology — Proceedings of CRYPTO’00, Lecture Notes in Computer Science 1880*, pages 1–19. Springer-Verlag, 2000.
- [Mao95] W. Mao. An augmentation of BAN-like logics. In *Proceedings of Computer Security Foundations Workshop VIII*, pages 44–56. IEEE Computer Society Press, June 1995.
- [MB93] W. Mao and C. Boyd. On the use of encryption in cryptographic protocols. In P.G. Farrell, editor, *Codes and Cyphers — Proceedings of 4th IMA Conference on Cryptography and Coding*, pages 251–262, December 1993. The Institute of Mathematics and Its Applications, 1995.
- [MB94] W. Mao and C. Boyd. On the use of encryption in cryptographic protocols, February 1994. Distributed by International Organization for Standardization (ISO) and International Electro-technical Commission (IEC) JTC1, SC27, WG2, Document N262: “Papers on authentication and key management protocols based on symmetric techniques”, This ISO document distributes the paper published in [MB93].
- [MB95] W. Mao and C. Boyd. Methodical use of cryptographic transformations in authentication protocols. *IEE Proceedings, Comput. Digit. Tech.*, 142(4):272–278, July 1995.
- [Mea92] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–53, 1992.
- [Mea96a] C. Meadows. Analyzing the Needham-Schroeder public key protocol: a comparison of two approaches. In E. Bertino et al, editor, *Proceedings of Computer Security, ESORICS’96, Lecture Notes in Computer Science 1146*, pages 351–364. Springer-Verlag, February 1996.
- [Mea96b] C. Meadows. The NRL protocol analyzer: an overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.

- [Mea99] C. Meadows. Analysis of the internet key exchange protocol using the NRL Protocol Analyzer. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 216–231. IEEE Computer Society Press, May 1999.
- [Mer78] R.C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, 1978.
- [MH78] R.C. Merkle and M.E. Hellman. Hiding information and signatures in trap-door knapsacks. *IEEE Trans. on Info. Theory*, 24:525–530, 1978.
- [MNSS87] S.P. Miller, C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, 1987.
- [Moo88] J.H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–601, 1988.
- [Moo92] J.H. Moore. Protocol failures in cryptosystems. In G.J. Simmons, editor, *Contemporary Cryptology, the Science of Information Integrity*, pages 541–558. IEEE Press, 1992.
- [MOV83] A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to a finite field. *IEEE Trans. Info. Theory*, 39:1636–1646, 1983.
- [MS98] C. Meadows and P. Syverson. A formal specification of requirements for payment transactions in the SET protocol. In R. Hirschfeld, editor, *Proceedings of Financial Cryptography (FC'98), Lecture Notes in Computer Science 1465*, pages 122–140. Springer-Verlag, February 1998.
- [MSST98] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP), version 10. draft-ietf-ipsec-isakmp-10.txt, also: Internet Engineering Task Force - Requests for Comments (IETF-RFC) 2048, November 1998.
- [MT79] R. Morris and K. Thompson. Password security: a case history. *Communications of the ACM*, 22(5):594–597, 1979.
- [MvOV97] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [MW99] U. Maurer and S. Wolf. The relationship between breaking the diffie-hellman protocol and computing discrete logarithms. *SIAM Journal of Computing*, 28(5):1689–1721, 1999.
- [MY91] U. Maurer and Y. Yacobi. Non-interactive public-key cryptography. In D.W. Davies, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'91, Lecture Notes in Computer Science 574*, pages 498–507. Springer-Verlag, 1991.

- [NBS77] NBS. Data Encryption Standard. U.S. Department of Commerce, FIPS Publication 46, Washington, D.C., January 1977. National Bureau of Standards.
- [NIS91] NIST. A proposed Federal Information Processing Standard for Digital Signature Standard (DSS). Federal Register Announcement August 30, 1991. National Institute of Standards and Technology.
- [NIS94] NIST. Digital Signature Standard. Federal Information Processing Standards Publication 186, 1994. U.S. Department of Commerce/N.I.S.T.
- [NIS95] NIST. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, April 1995. U.S. Department of Commerce/N.I.S.T.
- [NIS01a] NIST. Recommendation for block cipher modes of operation. NIST Special Publication 800-38A 2001 Edition, December 2001. U.S. Department of Commerce/N.I.S.T.
- [NIS01b] NIST. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication (FIPS PUB) 197, November 2001. U.S. Department of Commerce/N.I.S.T.
- [NR93] K. Nyberg and R. Rueppel. A new signature scheme based on the DSA giving message recovery. In *1st ACM Conference on Computer and Communications Security*, pages 58–61. ACM Press, 1993.
- [NR97] M. Naor and O. Reingold. Number theoretic constructions of efficient pseudo random functions. In *Proceedings of FOCS'97*, pages 458–467, 1997.
- [NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
- [NS87] R. Needham and M. Schroeder. Authentication revisited. *Operating Systems Review*, 21:7, 1987.
- [NS93] B.C. Neuman and S.G. Stubblebine. A note on the use of timestamps as nonces. *ACM Operating Systems Review*, 27(2):10–14, April 1993.
- [NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of 22nd ACM Symposium of Theory of Computing*, pages 427–437, 1990.
- [Odl99] A.M. Odlyzko. Discrete logairthms: the past and the future. XXXX, July 1999.
- [OP01] T. Okamoto and D. Pointcheval. REACT: rapid enhanced-security asymmetric cryptosystem transform. In D. Naccache, editor, *Topics in Cryptography, Cryptographers' Track, RSA Conference 2001 — Proceedings of CT-RSA'00, Lecture Notes in Computer Science 2020*, pages 159–175. Springer-Verlag, 2001.

- [OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.
- [Orm96] H. Orman. The Oakley key determination protocol, version 2. draft-ietf-ipsec-oakley-02.txt, 1996.
- [OU98] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In K. Nyberg, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'98, Lecture Notes in Computer Science 1403*, pages 308–318. Springer-Verlag, 1998.
- [Oxf91] Oxford. *Oxford Reference, Dictionary of Computing, Third Edition*. Oxford University Press, 1991.
- [Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'99, Lecture Notes in Computer Science 1592*, pages 223–238. Springer-Verlag, 1999.
- [PG97] J. Patarin and L. Goubin. Trapdoor one-way permutations and multivariate polynomials. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security — Proceedings of ICICS'97, Lecture Notes in Computer Science 1334*, pages 356–368. Springer-Verlag, 1997.
- [PH78] S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
- [PKC01] PKCS. Public Key Cryptography Standards, PKCS#1 v2.1. RSA Cryptography Standard, Draft 2, available at <http://www.rsasecurity.com/rsalabs/pkcs/>, 2001.
- [Poi99a] D. Pointcheval. HD-RSA: hybrid dependent RSA, a new public-key encryption scheme. Submission to IEEE P1363: Asymmetric Encryption, available at <http://grouper.ieee.org/groups/1363/P1363a/Encryption.html>, 1999.
- [Poi99b] D. Pointcheval. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'99, Lecture Notes in Computer Science 1592*, pages 239–254. Springer-Verlag, 1999.
- [Poi00] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In H. Imai and Y. Zheng, editors, *Public Key Cryptography — Proceedings of PKC'00, Lecture Notes in Computer Science 1751*, pages 129–146. Springer-Verlag, 2000.
- [Pol74] J.M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974.

- [PTVF88] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1988.
- [Rab79] M.O. Rabin. Digitized signatures and public-key functions as intractible as actorization. Technical Report LCS/TR-212, MIT Laboratory for Computer Science, 1979.
- [Riv92] R.L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments 1321, April 1992.
- [RL96] R.L. Rivest and B. Lampson. SDSI - A simple distributed security infrastructure. Invited Speech at CRYPTO'96, available at <http://theory.lcs.mit.edu/~cis/sdsi.html>, August 1996.
- [Ros94] A.W. Roscoe. Model checking CSP. In A.W. Roscoe, editor, *A Classical Mind: Essays in honour of C.A.R. Hoare*. Prentice-Hall, 1994.
- [Ros95] A.W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proceedings of Computer Security Foundations Workshop VIII*, pages 98–107. IEEE Computer Society Press, June 1995.
- [RRY98] R. Sidney R.L. Rivest, M.J.B. Robshaw and Y.L. Yin. The RC6 Block Cipher, v1.1. AES proposal, available at <http://www.rsa.com/rsalabs/aes/>, 1998.
- [RS92] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Advances in Cryptology — Proceedings of CRYPTO'91, Lecture Notes in Computer Science 576*, pages 433–444. Springer-Verlag, 1992.
- [RS96] R. Rivest and A. Shamir. PayWord and MicroMint: two simple micropayment schemes. *CryptoBytes, RSA Laboratories*, 2(1):7–11, Spring 1996.
- [RS01] P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.
- [RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [SA98] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Comm. Math. Univ. Sancti. Pauli*, 47:81–92, Spring 1998.
- [SB92] M.E. Smid and D.K. Branstad. The data encryption standard, past and future. In G.J. Simmons, editor, *Contemporary Cryptology, the Science of Information Integrity*, pages 43–46. IEEE Press, 1992.

- [Sch90] C.P. Schnorr. Efficient identification and signature for smart cards. In G. Brassard, editor, *Advances in Cryptology — Proceedings of CRYPTO'89, Lecture Notes in Computer Science 435*, pages 239–252. Springer-Verlag, 1990.
- [Sch91] C.P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
- [Sch96] S. Schneider. Security properties and CSP. In *Proceedings of the 1996 IEEE Symposium in Security and Privacy*, pages 174–187. IEEE Computer Society Press, 1996.
- [Sch01] B. Schneier. *Secrets and Lies*. John Wiley & Sons, 2001.
- [Sem98] L.A. Semaev. Evaluation of discrete logarithms in a group of p -torsion points of an elliptic curve in characteristic p . *Math. Comp.*, 67(221):353–356, 1998.
- [SET97] SET. Secure Electronic Transaction Specification, Version 1.0. Available at <http://www.setco.org/>, May 1997.
- [SGR85] S. Micali, S. Goldwasser and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of 17th Ann. ACM Symp. on Theory of Computing*, pages 291–304, 1985. A journal version under the same title appears in: *SIAM Journal of Computing* vol. 18, pp. 186–208, 1989.
- [Sha48a] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(3):379–423, July 1948.
- [Sha48b] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:623–656, October 1948. Continued from July 1948 issue (i.e., [Sha48a]).
- [Sha49] C.E. Shannon. Communications theory of secrecy systems. *Bell Systems Technical Journal*, 28:656–715, October 1949.
- [Sha51] C.E. Shannon. Predilection and entropy of printed english. *Bell Systems Technical Journal*, 30:50–64, January 1951.
- [Sha85] A. Shamir. Identity-based cryptosystems and signature schemes. In G.T. Blakley and D. Chaum, editors, *Advances in Cryptology — Proceedings of CRYPTO'84, Lecture Notes in Computer Science 196*, pages 48–53. Springer-Verlag, 1985.
- [Sho00] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In B. Preneel, editor, *Advances in Cryptology — Proceedings of EUROCRYPT'00, Lecture Notes in Computer Science 1807*, pages 275–288. Springer-Verlag, 2000.
- [Sho01a] V. Shoup. OAEP Reconsidered. In J. Killian, editor, *Advances in Cryptology — Proceedings of CRYPTO'01, Lecture Notes in Computer Science 2139*, pages 239–259. Springer-Verlag, 2001.

- [Sho01b] V. Shoup. A proposal for an ISO standard for public key encryption (version 2.1). Distributed by International Organization for Standardization (ISO) and International Electro-technical Commission (IEC) JTC1, SC27, WG2, December 2001. An earlier version appeared in ISO/IEC JTC 1/SC 27 N2765 “Editor’s contribution on public key encryption” (February 2001).
- [Sil97] R.D. Silverman. Fast generation of random, strong RSA primes. *CryptoBytes*, 3(1):9–13, 1997.
- [Sim85] G.J. Simmons. How to (selectively) broadcast a secret. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 108–113. IEEE Computer Society Press, 1985.
- [Sim92] G.J. Simmons. A survey of information authentication. In G.J. Simmons, editor, *Contemporary Cryptology, the Science of Information Integrity*, pages 379–419. IEEE Press, 1992.
- [Sin99] S. Singh. *The Code Book*. Fourth Estate, 1999.
- [SKW⁺98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: a 128-bit block cipher, AES proposal. Available at: <http://www.counterpane.com/twofish.html>, 1998.
- [Sma99] N.P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12:193–196, 1999.
- [SOK00] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Proceedings of the 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, January 2000*.
- [Sol01] D. Soldera. SEG - a provably secure variant of El-Gamal. Technical Report Technical Report, HPL-2001-149, Hewlett-Packard Laboratories, Bristol, June 2001.
- [SRA80] A. Shamir, R. Rivest, and L. Adleman. Mental poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43, Boston, Mass, 1980. Prindle, Weber & Schmidt.
- [Sol77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal of Computing*, 6(1):84–85, March 1977.
- [SSQ02] D. Soldera, J. Seberry, and C. Qu. The analysis of Zheng-Seberry scheme. In L. M. Batten and J. Seberry, editors, *7th Australian Conference in Information Security and Privacy — Proceedings of ACISP’02, Lecture Notes in Computer Science 2384*, pages 159–168. Springer-Verlag, 2002.
- [Sta96] M. Stadler. Publicly verifiable secret sharing. In U. Maurer, editor, *Advances in Cryptology — Proceedings of EUROCRYPT’96, Lecture Notes in Computer Science 1070*, pages 190–199. Springer-Verlag, 1996.

- [Sti95] D.R. Stinson. *Cryptography: Theory and Practice*. CRC Press, Inc., 1995.
- [SvO94] P. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of 1994 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1994.
- [Sv93] P. Syverson. On key distribution protocols for repeated authentication. *ACM Operating Systems Review*, 27(4):24–30, October 1993.
- [Tan88] H. Tanaka. A realization scheme for the identity-based cryptosystem. In C. Pomerance, editor, *Advances in Cryptology — Proceedings of CRYPTO'87, Lecture Notes in Computer Science 293*, pages 340–349. Springer-Verlag, 1988.
- [TI89] S. Tsuji and T. Itoh. An ID-based cryptosystem based on the discrete logarithm problem. *IEEE Journal on Selected Areas in Communication*, 7(4):467–473, 1989.
- [Tru92] G. Trudik. Message authentication with one-way functions. *Computer Communication Review*, 22:29–38, 1992.
- [Tuc79] W. Tuchman. Hellman presents no shortcut solutions to the DES. *IEEE Spectrum*, 16(7):40–41, 1979.
- [VAB91] V. Varadharajan, P. Allen, and S. Black. An analysis of the proxy problem in distributed systems. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 255–275, 1991.
- [vO93] P.C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols (extended abstract). In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 232–243, 1993.
- [VV85] U. Vazirani and V. Vazirani. Efficient and secure pseudo-random number generation (extended abstract). In G.T. Blakley and D. Chaum, editors, *Advances in Cryptology — Proceedings of CRYPTO'84, Lecture Notes in Computer Science 196*, pages 193–202. Springer-Verlag, 1985.
- [WC00] C.P. Williams and S.H. Clearwater. *Ultimate Zero and One*. Copernicus, Springer-Verlag New York, Inc., 2000.
- [Wie90] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36(3):553–558, 1990.
- [Wie94] M. Wiener. Efficient des key search. Technical report, Technical Report TR-244, School of Computer Science, Carleton University, Ottawa, May 1994.
- [WL92] T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.

- [WL94] T.Y.C. Woo and S.S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, July 1994.
- [Yao82] A.C. Yao. Theory and applications of trapdoor functions (extended abstract). In *Proceedings of 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.
- [Ylo95] T. Ylonen. The SSH (secure shell) remote login protocol. INTERNET-DRAFT, draft-ylonen-ssh-protocol-00.txt, September 1995.
- [Ylo02a] T. Ylonen. SSH authentication protocol. INTERNET-DRAFT, draft-ietf-userauth-16.txt, September 2002.
- [Ylo02b] T. Ylonen. SSH connection protocol. INTERNET-DRAFT, draft-ietf-connect-16.txt, September 2002.
- [Ylo02c] T. Ylonen. SSH protocol architecture. INTERNET-DRAFT, draft-ietf-architecture-13.txt, September 2002.
- [Ylo02d] T. Ylonen. SSH transport layer protocol. INTERNET-DRAFT, draft-ietf-transport-15.txt, September 2002.
- [Zhe97] Y. Zheng. Digital signcryption or how to achieve $\text{cost}(\text{signature} \& \text{encryption}) < \text{cost}(\text{signature}) + \text{cost}(\text{encryption})$. In B. Kaliski Jr., editor, *Advances in Cryptology — Proceedings of CRYPTO'97, Lecture Notes in Computer Science 1294*, pages 165–179. Springer-Verlag, 1997.
- [Zim95] P.R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, Massachusetts, 1995. Second printing.
- [ZS93a] Y. Zheng and J. Seberry. Immunizing public key cryptosystems against chosen ciphertext attacks. *Special Issue on Secure Communications, IEEE Journal on Selected Areas on Communications*, 11(5):715–724, June 1993.
- [ZS93b] Y. Zheng and J. Seberry. Practical approaches to attaining security against adaptively chosen ciphertext attacks (extended abstract). In E.F. Brickell, editor, *Advances in Cryptology — Proceedings of CRYPTO'92, Lecture Notes in Computer Science 740*, pages 291–304. Springer-Verlag, 1993.