Factoring a multi-prime modulus N with random bits

Routo Terada
(rt@ime.usp.br)
Reynaldo C. Villena
(reynaldo@ime.usp.br)

U. of S. Paulo

ISC 2013

Agenda

1 - Basic concepts

The RSA cryptosystem

The RSA cryptosystem consists of 3 algorithms

1.-Algorithm to generate the keys

$$N = \prod_{i=1}^{u} r_i \qquad ed = 1 \mod \phi(N)$$

- Public key $pk\langle N, e\rangle$
- Private key $sk\langle N, d\rangle$

2.- Algorithm to encrypt

$M \in \mathbb{Z}_N,\ pk\langle N, e\rangle$

$$C = M^e \mod N$$

2.- Algorithm to decrypt

$C,\ sk\langle N, d\rangle$

$$M = C^d \mod N$$

RSA versions

- case $u = 2$ known as **Basic RSA cryptosystem**
- case $u \geq 3$ known as **Multi-prime RSA cryptosystem**

PKCS - Public Key Cryptography Standards

- PKCS is a set of standards published by *RSA Labs*
- PKCS contains specifications to speed-up software implementations of public key cryptosystems.

Where

PKCS #1 is a standard with recommendations for RSA implementation.

Representation of the RSA public key according to PKCS #1

- $pk\langle N, e\rangle$ $\rightarrow C = M^e \mod N.$

Representation of the RSA private key according to PKCS #1

- $pk\langle N, d\rangle$ $\rightarrow M = C^d \mod N.$
- $sk\langle r_1, r_2, d_1, d_2, r_2^{-1}, \langle r_3, d_3, t_3\rangle, .., \langle r_u, d_u, t_u\rangle\rangle$ $\rightarrow$ CRT[a].

[a] Chinese Remainder Theorem

PKCS #1 - RSA (Recomendation for RSA implementations)

ANS.1 representation of the RSA keys according to PKCS #1.

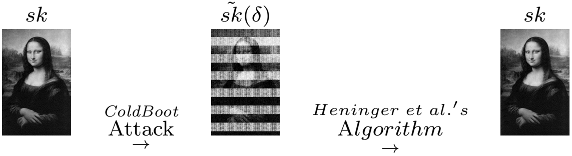
```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER,  -- n
    publicExponent    INTEGER   -- e
}
RSAPrivateKey ::= SEQUENCE {
    version           Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
Version ::= INTEGER { two-prime(0), multi(1) }
    (CONSTRAINED BY {-- version must be multi if otherPrimeInfos present --})
OtherPrimeInfos ::= SEQUENCE SIZE(1..MAX) OF OtherPrimeInfo
OtherPrimeInfo ::= SEQUENCE {
    prime             INTEGER,  -- ri
    exponent          INTEGER,  -- di
    coefficient       INTEGER   -- ti
}
```

- High redundancy in the private key is noticeable.
- $sk\langle N, e, d, r_1, r_2, d_1, d_2, r_2^{-1}, \langle r_3, d_3, t_3\rangle, .., \langle r_u, d_u, t_u\rangle\rangle$.

2 - Paper goals

Previous works

- J. A. Halderman (2008) showed it is possible to recover bits due to the data remanent property of DRAM memory (*Cold Boot* attacks).
- N. Heninger and H. Shacham published an algorithm to reconstruct the private key (only for the Basic RSA) that uses the redundancy of the secret key in the PCKS #1 standard.



- Kogure et al. proved a general theorem to factor a multi-power modulus $N = r_1^m r_2$ with random bits of its prime factors. The particular cases of Takagi's variant of RSA and Paillier Cryptosystem are addressed. The bounds for expected values in our cryptanalysis are derived directly, without applying their theorem.

Paper goals

Our goals

- To factor integer $N = \prod_{i=1}^{u} r_i$ given a fraction δ of random bits of its primes.
- Generalize the Heninger and Shacham's algorithm to recover the RSA key sk given a fraction δ of the $\tilde{sk}$ key bits.

3 - Prime factorization of N

Introduction

$$N = \prod_{i=1}^{u} r_i$$

Idea of the algorithm

$$f(x_1, x_2, ..., x_u) = N - \prod_{i=1}^{u} x_i \quad \overset{\text{solution}}{\Longrightarrow} \quad f(r_1, r_2, ..., r_u) = 0$$

Let us suppose we have

$$f(r'_1, r'_2, ..., r'_u) \pmod{2^j} \quad \Longrightarrow \quad f(x_1, x_2, ..., x_u) \pmod{2^{j+1}}$$

How the algorithm works:

$$f \pmod 2 \Rightarrow f \pmod{2^2} \Rightarrow ... \Rightarrow f \pmod{2^j} \Rightarrow f \pmod{2^{j+1}} \Rightarrow ... \Rightarrow f \pmod{2^{\frac{n}{u}}}$$

- Notice that the primes r_i have the same bit length: $lg(r_i) = \frac{n}{u}$

$$f(r_1, r_2, ..., r_u) \in f \pmod{2^{\frac{n}{u}}}$$

Hensel's Lemma

Multivariate Hensel's Lemma

One root $r = (r_1, r_2, ..., r_u)$ of the polynom $f(x_1, x_2, ..., x_u) \mod \pi^j$ can be used to generate a root $r + b \mod \pi^{j+1}$ if $b = (b_1\pi^j, b_2\pi^j, ..., b_u\pi^j)$, $0 \le b_i \le \pi - 1$, that is a solution for the equation

$$f(r+b) = f(r) + \sum_i b_i \pi^j f_{x_i}(r) \equiv 0 \pmod{\pi^{j+1}}$$

(where, f_{x_j} is a partial derivative of f with respect to x_j)

With $r(r'_1, r'_2, ..., r'_u)$ that is a root of the polynom $f(x_1, x_2, ..., x_u) \pmod{2^j}$, we can obtain the root $r(r'_1 + 2^j b_1, r'_2 + 2^j b_2, ..., r'_u + 2^j b_u)$ that is a root of $f(x_1, x_2, ..., x_u)$ $\pmod{2^{j+1}}$

$$\left(N - \prod_{i=1}^{u} r'_i\right)[j] = \sum_{i=1}^{u} b_i \pmod 2$$

Observe that for a root of $f \pmod{2^j}$ can generate a total of 2^{u-1} roots of f $\pmod{2^{j+1}}$

Algorithm to factor a multiprime N

Define

$$root[j-1] = \langle r'_1, r'_2, ..., r'_u \rangle \in f \pmod{2}^j$$

where $root[0] = \langle 1, 1, ..., 1 \rangle$

$$root[0] \Rightarrow ... \Rightarrow root[j-1] \Rightarrow root[j] \Rightarrow ... \Rightarrow root\left[\frac{n}{u}\right]$$

From the solutions $root[j-1] = \langle r'_1, r'_2, ..., r'_u \rangle$, the solutions $root[j]$ are obtained as follows

$$root[j] = \langle r'_1 + 2^j r_1[j], r'_2 + 2^j r_2[j], ..., r'_u + 2^j r_u[j] \rangle$$

where the following should be satisfied

$$\left(N - \prod_{i=1}^{u} r'_i \right)[j] = \sum_{i=1}^{u} r_i[j] \pmod{2}$$

Algorithm to factor a multiprime N

Algorithm 1: Factoring N

Input: $N, u, \langle \tilde{r_1}, \tilde{r_2}, ..., \tilde{r_u} \rangle$

Output: $root[\frac{n}{u}]$ where $\langle r_1, r_2, ..., r_u \rangle$ is in $root[\frac{n}{u}]$

1 $root[0] = [\langle 1_1, 1_2, ..., 1_u \rangle]$;

2 $j = 1$;

3 **for each** $\langle r'_1, r'_2, ..., r'_u \rangle$ **in** $root[j-1]$ **do**

4   **for all possible** $\langle r_1[j], r_2[j], ..., r_u[j] \rangle$ **do**

5   **if** $(N - \prod_{i=1}^{u} r'_i)[j] \equiv \sum_{i=1}^{u} r_i[j] \pmod 2$ **then**

6         $root[j].add(\langle r'_1 + 2^j r_1[j], r'_2 + 2^j r_2[j], ..., r'_u + 2^j r_u[j] \rangle)$

7 **if** $j < \frac{n}{u}$ **then**

8     $j := j + 1$;

9   go to step 3;

10 **return** $root[\frac{n}{u}]$;

- if $r_i[j]$ is known then there is only one fixed value.
- if $r_i[j]$ is not known then there are two possible values, 0 or 1.

Complexity of Algorithm 1 to factor a multiprime N

Behavior of Algorithm 1

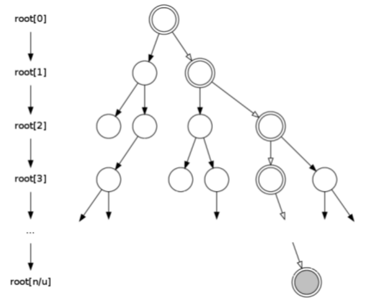


Figure 1: *(Factoring N) Behavior of Algorithm to factor N*

Complexity analysis of Algorithm 1

- G: Number of incorrect roots lifted by a good root.
- B: Number of incorrect roots lifted by a incorrect root.
- X_j: Number of incorrect roots lifted at level j.

Number of roots lifted by a good root

- Have a good root of $root[j-1]$
- Have some known bits of $\langle r_1[j], r_2[j], ..., r_u[j]\rangle$ (have a fraction δ of known bits in $\langle \tilde{r_1}, \tilde{r_2}, ..., \tilde{r_u}\rangle$)

$$\boxed{\left(N - \prod_{i=1}^{u} r_i'\right)[j] = \sum_{i=1}^{u} r_i[j] \pmod 2}$$

Number of roots lifted by a good root

Let h be the number of unknown bits in $\langle r_1[j], r_2[j], ..., r_u[j]\rangle$

Cases	Number of roots lifted
$1 \le h \le u$	2^{h-1}
$h = 0$	1

Notice that a good root of $root[j-1]$ always produces a good root of $root[j]$ (that is unique at any level).

Number of incorrect roots lifted by a good root (B)

Number of incorrect roots lifted by a good root (B)

Cases	Number of incorrect solutions lifted
$1 \leq h \leq u$	$2^{h-1} - 1$
$h = 0$	0

Expected Value of G ($\mathbb{E}[G]$)

$$\mathbb{E}[G] = \sum_{h=1}^{u} (2^{h-1} - 1) P(b_u = h)$$
$$= \sum_{h=1}^{u} (2^{h-1} - 1) \binom{u}{h} (1 - \delta)^h (\delta)^{u-h}$$

with $P(b_u = h) = P(bits_{unknown} = h)$

Number of incorrect roots lifted by an incorrect root

Define

$$c_1 = \left(N - \prod_{i=1}^{u} r'_i\right)[j]$$

that is computed by a good root in $root[j-1]$.

Types of incorrect roots in $root[j-1]$

There are two types of incorrect roots

$$c_1 \equiv \left(N - \prod_{i=1}^{u} r'_i\right)[j] = \sum_{i=1}^{u} r_i[j] \pmod 2$$
$$\overline{c_1} \equiv \left(N - \prod_{i=1}^{u} r'_i\right)[j] = \sum_{i=1}^{u} r_i[j] \pmod 2$$

Number of incorrect roots lifted by an incorrect root

Number of incorrect roots lifted by an incorrect root

Number of known bits	$c_1 \equiv \left(N - \prod_{i=1}^{u} r'_i\right)[j]$	$\overline{c_1} \equiv \left(N - \prod_{i=1}^{u} r'_i\right)[j]$
$1 \leq h \leq u$	2^{h-1}	2^{h-1}
$h = 0$	1	0

Expected Value of B ($\mathbb{E}[B]$)

$$\mathbb{E}[B] = \sum_{h=1}^{u} 2^{h-1} P(b_u = h) P(c_1) + \sum_{h=1}^{u} 2^{h-1} P(b_u = h) P(\overline{c_1}) + P(b_u = 0) P(c_1)$$
$$= \frac{(2-\delta)^u}{2}$$

where $P(c_1) \approx P(\overline{c_1}) \approx P\left(\left(N - \prod_{i=1}^{u} r'_i\right)[j] = 1\right) \approx P\left(\left(N - \prod_{i=1}^{u} r'_i\right)[j] = 0\right) \approx \frac{1}{2}$.

Number of Incorrect Solutions Generated at level j

Recurrence function: $X_j = X_{j-1}B + G$

Expected Value of X_j

$$\mathbb{E}[X_j] = \mathbb{E}[G]\frac{1-\mathbb{E}[B]^j}{1-\mathbb{E}[B]}$$

$$\begin{aligned}\mathbb{V}ar[X_j] =& \mathbb{E}[B]^{2(j-1)}\left[-\frac{\mathbb{E}[G][\mathbb{E}[B^2]-\mathbb{E}[B]+\mathbb{E}[B]\mathbb{E}[G]]\mathbb{E}[B]}{(1-\mathbb{E}[B])(1-\mathbb{E}[B]^2)}\right] + \mathbb{E}[G]\frac{1-\mathbb{E}[B]^j}{1-\mathbb{E}[B]} \\ & - \mathbb{E}[B]^{j-1}\left[\frac{\mathbb{E}[G][\mathbb{E}[B^2]-\mathbb{E}[B]+2\mathbb{E}[B]\mathbb{E}[G]]}{(1-\mathbb{E}[B])^2}\right] - \left[\mathbb{E}[G]\frac{1-\mathbb{E}[B]^j}{1-\mathbb{E}[B]}\right]^2 \\ & \frac{1}{1-\mathbb{E}[B]^2}\left[\frac{\mathbb{E}[G][\mathbb{E}[B^2]-\mathbb{E}[B]+\mathbb{E}[B]\mathbb{E}[G]]}{1-\mathbb{E}[B]}\right]\end{aligned}$$

The definition of $\mathbb{E}[X_j]$ and $\mathbb{V}ar[X_j]$ are functions of j and δ.

Number of incorrect roots analyzed by Algorithm 1

$$\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] = \sum_{j=1}^{\frac{n}{u}} \mathbb{E}\left[X_j\right] = \sum_{j=1}^{\frac{n}{u}} \mathbb{E}[G]\frac{1-\mathbb{E}[B]^j}{1-\mathbb{E}[B]}$$

$$= \frac{n}{u}\frac{\mathbb{E}[G]}{1-\mathbb{E}[B]} + \frac{\mathbb{E}[G]\mathbb{E}[B](\mathbb{E}[B]^{\frac{n}{u}}-1)}{(\mathbb{E}[B]-1)^2}$$

$$\mathbb{V}ar\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] = \sum_{l=1}^{\frac{n}{u}}\sum_{j=1}^{\frac{n}{u}} Cov(X_l, X_j) \leq \sum_{l=1}^{\frac{n}{u}}\sum_{j=1}^{\frac{n}{u}} \sqrt{\mathbb{V}ar[X_l]\mathbb{V}ar[X_j]}$$

$$\leq \sum_{l=1}^{\frac{n}{u}}\sum_{j=1}^{\frac{n}{u}} \sqrt{\max(\mathbb{V}ar[X_1], .., \mathbb{V}ar[X_{\frac{n}{u}}])^2}$$

$$\leq \left(\frac{n}{u}\right)^2 \max(\mathbb{V}ar[X_1], .., \mathbb{V}ar[X_{\frac{n}{u}}])$$

Number of incorrect roots analyzed by Algorithm 1

Where the behavior of $\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right]$ and $\mathbb{V}ar\left[\sum_{j=1}^{\frac{n}{u}} X_j\right]$ can be:

- Exponential ($\mathbb{E}[B] > 1$ because $\lim\limits_{n\to\infty} \mathbb{E}[B]^{\frac{n}{u}} = +\infty$)
- Polynomial ($\mathbb{E}[B] < 1$ because $\lim\limits_{n\to\infty} \mathbb{E}[B]^{\frac{n}{u}} = 0 < 1$)

With $\mathbb{E}[B] < 1$ we get

$$\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] = \frac{n}{u}\frac{\mathbb{E}[G]}{1-\mathbb{E}[B]} + \frac{\mathbb{E}[G]\mathbb{E}[B](\mathbb{E}[B]^{\frac{n}{u}}-1)}{(\mathbb{E}[B]-1)^2} < \frac{n}{u}\frac{\mathbb{E}[G]}{1-\mathbb{E}[B]}$$

$$\mathbb{V}ar\left[\sum_{j=1}^{\frac{n}{u}} X_j\right] \le \left(\frac{n}{u}\right)^2 \max(\mathbb{V}ar[X_1], .., \mathbb{V}ar[X_{\frac{n}{u}}]),$$

where the values for $\mathbb{E}\left[\sum_{j=1}^{\frac{n}{u}} X_j\right]$ and $\mathbb{V}ar\left[\sum_{j=1}^{\frac{n}{u}} X_j\right]$ are bounded by polynomial functions.

Analysis of expected behavior of Algorithm 1

Chebyshev's Theorem

The Chebyshev's inequality provides a probability of how many standard deviations of a random variable is far from the expected value.

$$P(\mathbb{E}[X] - c\sigma < X < \mathbb{E}[X] + c\sigma) \geq 1 - \frac{1}{c^2}$$

The probability that any random variable is c standard deviations far from the expected value is at least $1 - \frac{1}{c^2}$.

Applying Chebyshev's inequality, we have that the probability of Algorithm 1 to analyze more than

$$\mathbb{E}[\sum_{j=1}^{\frac{n}{u}} X_j] + n\sqrt{\mathbb{V}ar[\sum_{j=1}^{\frac{n}{u}} X_j]} \leq \frac{n}{u}\frac{\mathbb{E}[G]}{1-\mathbb{E}[B]} + \left(\frac{n}{u}\right)^2 \max(\mathbb{V}ar[X_1], .., \mathbb{V}ar[X_{\frac{n}{u}}])$$

incorrect roots is less than $\frac{1}{n^2}$.

Algorithm to factor a multiprime N

Result of the complexity analysis of Algorithm 1

To factor a multiprime $N = \prod_{i=1}^{u} r_i$ in polynomial time, $O(n^2)$, with probability greater than $1 - \frac{1}{n^2}$ the ratio δ of known random bits of $\langle \tilde{r_1}, \tilde{r_2}, ..., \tilde{r_u} \rangle$ is greater than $2 - 2^{\frac{1}{u}}$ ($\delta > 2 - 2^{\frac{1}{u}}$).
Summary:

$$\mathbb{E}[B] = \frac{(2-\delta)^u}{2} < 1 \qquad \Rightarrow \qquad \delta > 2 - 2^{\frac{1}{u}}.$$

Some examples

- To factor $N = \prod_{i=1}^{2} r_i$ should have $\delta > 2 - 2^{\frac{1}{2}} = 0.5857$ ($\delta \geq 0.59$)
- To factor $N = \prod_{i=1}^{3} r_i$ should have $\delta > 2 - 2^{\frac{1}{3}} = 0.7401$ ($\delta \geq 0.75$)
- To factor $N = \prod_{i=1}^{4} r_i$ should have $\delta > 2 - 2^{\frac{1}{4}} = 0.8108$ ($\delta \geq 0.82$)

Implementation of Algorithm 1

Besides the analysis, we also did an implementation of Algorithm 1 to validate it.

- Algorithm 1 was implemented in C language with the *Relic-toolkit* library on a Intel Core I3 2.4 Ghz with 3 Mb of cache and 4 Gb of DDR3 memory.
- The experiments were done with N 2048 bits long and specific δ values.
- For each δ, 100 integers N were lifted.
- For each integer N, 100 inputs with δ fraction of correct bits were lifted.
- The experiments were done for integers $N = \prod_{i=1}^{u} r_i$ with $2 \leq u \leq 4$.

Experiments

- For $N = \prod_{i=1}^{2} r_i$ 2048 bits $\delta = 0.59$ less than $15n + 15n^2$ roots were analyzed.

	Number of analyzed roots			# Exp.	Time (sec)
δ	Min	Max	Average	(> 1M)	Average
0.62	1861	347138	3709	0	0.047510
0.61	1983	945728	4949	0	0.115277
0.60	2233	789608	6344	0	0.119484
0.59	2411	928829	8953	2	0.187600
0.58	2631	987577	14736	7	0.250224
0.57	3436	994640	24281	29	0.531079
0.56	4012	998414	42231	134	0.722388

Experiments

- For $N = \prod_{i=1}^{3} r_i$ 2048 bits $\delta = 0.75$ less than $3n + 4n^2$ roots were analyzed.

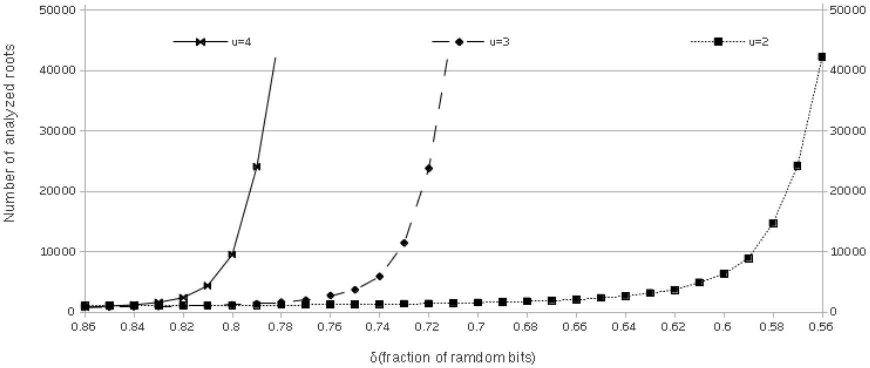
	Number of analyzed roots			# Exp.	Time (sec)
δ	Min	Max	Average	(> 1M)	Average
0.78	985	35509	1676	0	0.032866
0.77	1128	171142	2022	0	0.033884
0.76	1205	323228	2777	0	0.049238
0.75	1380	177293	3723	1	0.099373
0.74	1607	571189	5941	1	0.197553
0.73	1681	999766	11470	11	0.281414
0.72	2087	983404	23826	50	0.995017

Experiments

- For $N = \prod_{i=1}^{4} r_i$ 2048 bits, $\delta = 0.82$ less than $2n + 2n^2$ roots were analyzed.

	Number of analyzed roots			# Exp.	Time (sec)
δ	Min	Max	Average	(> 1M)	Average
0.85	692	32620	1026	0	0.019939
0.84	716	31447	1245	0	0.024748
0.83	823	67456	1649	0	0.040714
0.82	931	217391	2424	0	0.063754
0.81	1044	558521	4408	1	0.111688
0.80	1249	994386	9571	14	0.236320
0.79	1632	972196	24085	58	0.609435

Experiments - Algorithm 1



Thanks for yor attention!!!