DCC–IME–USP

ROUTO TERADA
SALA 204C TEL.: 3091 5918
E-MAIL rt@ime.usp.br

MAC 5723 - 336 - Criptografia

PRIMEIRO SEMESTRE DE 2008

Exercício-Programa Data de entrega: até **25 de maio de 2008.**

Observações

- Este exercício é para ser feito *individualmente*.
- Entregue no sistema PACA UM ÚNICO arquivo contendo os arquivos seguintes, eventualmente comprimidos:
  - um arquivo chamado LEIA.ME (em formato .txt) com:
    - seu nome completo, e número USP,
    - os nomes dos arquivos inclusos com uma breve descrição de cada arquivo,
    - uma descrição sucinta de *como usar* o programa executável, necessariamente na linha-de-comando, i.e., SEM interface gráfica,
    - qual computador (Intel, SUN, ou outro) e qual compilador C (gcc, TURBO-C, ou outro) e qual sistema operacional (LINUX, UNIX, MS-DOS, ou outro) foi usado,
    - instruções de como compilar o(s) arquivo(s) fonte(s).
  - o arquivo MAKE,
  - os arquivos do programa-fonte necessariamente em *linguagem ANSI-C*,
  - o programa *compilado*, i.e., incluir o código executável (se não incluir, a nota será zero!)
  - se for o caso, alguns arquivos de entrada e saída usados nos testes: arquivos com os dados de *entrada* chamados ENT1, ENT2, etc., e arquivos com os dados de *saída* correspondentes, chamados SAI1, SAI2, etc.
- Coloque comentários em seu programa explicando o que cada etapa do programa significa! Isso será levado em conta na sua nota.
- Faça uma saída clara! Isso será levado em conta na sua nota.
- Não deixe para a última hora. Planeje investir 70 porcento do tempo total de dedicação em escrever o seu programa todo e simular o programa SEM computador (eliminando erros de lógica) ANTES de digitar e compilar no computador. Isso economiza muito tempo e energia.
- A nota será diminuida de um ponto a cada dia "corrido" de atraso na entrega.

1 Função K256

Implementar a função criptográfica K256, com chave de 256 bits, e com entrada e saída de 256 bits. Esta função está descrita na Seção 2. Você deve **deduzir** o algoritmo inverso do K256.

```
Entrada: bloco de 256 bits divididos em 4 subblocos de 64 bits, A,B,C,D, nessa ordem,
         e chave principal 256 bits
Saida: 256 bits criptografados armazenados em A,B,C,D
```

1.1 Linha de comando

O seu programa deve ser executado na linha de comando, com parâmetros relevantes, em um dos seguintes modos: (se houver a opção -a após a senha, o programa deve gravar brancos no lugar do arquivo de entrada e deletá-lo, o *default* é não efetuar o apagamento)

- Modo (1) Para criptografar arquivos:
  programa -c -i <arquivo de entrada> -o <arquivo de saída> -p <senha> -a
- Modo (2) Para decriptografar arquivos:
  programa -d -i <arquivo de entrada> -o <arquivo de saída> -p <senha>
- Modo (3) Para calcular aleatoriedade pelo método 1 (item 1 abaixo):
  programa -1 -i <arquivo de entrada> -p <senha>
- Modo (4) Para calcular aleatoriedade pelo método 2 (item 2 abaixo):
  programa -2 -i <arquivo de entrada> -p <senha>

A sintaxe dos parâmetros de "features" que não fazem parte da especificação (como a criptografia da senha no início do arquivo, se for o caso) ficariam à escolha de cada aluno.

1.2 Senha e chave principal K

A chave principal K de 256 bits deve ser gerada da senha S digitada no parâmetro -p <senha>. S deve conter pelo menos **8** caracteres, com **pelo menos** 2 letras e 2 algarismos decimais; se S possuir menos que 32 caracteres (i.e., 32 bytes), a chave K de 256 bits deve ser derivada de S concatenando-se S com ela própria até completar 32 bytes (256 bits).

1.3 O programa

O seu programa deve ler do disco o arquivo de entrada Entra, e deve gravar o arquivo de saída Sai correspondente a Entra criptografado/decriptografado com a chave K, no modo CBC (Cipher Block Chaining), com blocos de 256 bits.

1.4 Modo CBC e testes

O Modo CBC consiste em encadear um bloco de 256 bits com o código do bloco anterior da maneira vista em aula.

1. No modo CBC, utilizar bits iguais a UM como Valor Inicial.
2. V. deve testar o programa com pelo menos dois arquivos Entra Por exemplo, o seu próprio programa-fonte. Teste não só com arquivos-texto como com arquivos binários; por exemplo, com algum código executável.
3. Se o último bloco a ser criptografado não possuir comprimento igual a 256 bits, completá-lo com bits iguais a UM.
4. Verifique se o arquivo decriptografado Sai possui o mesmo comprimento que o arquivo original Entra. O *último* bloco criptografado de Sai deve conter o comprimento do arquivo original Entra.

1.5 Medidas de aleatoridade

O seu programa deve também efetuar os itens seguintes:

Item 1: Medir a aleatoriedade do K256 da seguinte maneira.

Seja $VetEntra$ um vetor lido de um arquivo de entrada para a memória principal com pelo menos 1024 bits (i.e., pelo menos 4 blocos de 256 bits, de modo que

$$VetEntra = Bl(1)||Bl(2)||Bl(3)||Bl(4)||...,$$

cada bloco $Bl()$ de 256 bits e $|VetEntra| \geq 4 * 256 = 1024$).

$VetEntra$	$Bl(1)$	$Bl(2)$	$Bl(3)$	$Bl(4)$	...
$VetEntraC$ (criptografado)	$BlC(1)$	$BlC(2)$	$BlC(3)$	$BlC(4)$	...
					...
$VetAlter$	$BlAlter(1)$	$BlAlter(2)$	$BlAlter(3)$	$BlAlter(4)$	...
$VetAlterC$ (criptografado)	$BlAlterC(1)$	$BlAlterC(2)$	$BlAlterC(3)$	$BlAlterC(4)$	...
	$j = 1, 2, ...256$	$j = 1, 2, ...512$	$j = 1, 2, ...768$	$j = 1, 2, ...1024$	...
Dist ncia de Hamming	$H(1)$	$H(2)$	$H(3)$	$H(4)$	...
Soma acumulada de $H(k)$	$SomaH(1)$	$SomaH(2)$	$SomaH(3)$	$SomaH(4)$	...

Para $j = 1, 2, ..., |VetEntra|$ fazer o seguinte:

1. alterar apenas na memória só o j-ésimo bit do vetor $VetEntra$ de cada vez, obtendo um **outro vetor** na memória principal chamado $VetAlter$, para $j = 1, 2, 3, ...$ tal que $|VetEntra| = |VetAlter|$; isto é, $VetEntra$ e $VetAlter$ só diferem no j-ésimo bit, mas são de igual comprimento. No caso de apenas 4 blocos, $j = 1, 2, 3, ...1024$. Por exemplo, no caso de $j = 2$, $Bl(1) = 01010101\alpha$, $Bl(2) = 00110101\alpha, ...$ e

   $$VetAlter = BlAlter(1)||BlAlter(2)||... = 00010101||00110101||\alpha...$$

   ou seja diferem só no bit na posição 2.

2. seja $VetEntraC = BlC(1)||BlC(2)||BlC(3)||BlC(4)||...$ o vetor $VetEntra$ criptografado pelo K256-CBC. E seja

$$VetAlterC = BlAlterC(1)||BlAlterC(2)||BlAlterC(3)||BlAlterC(4)||...$$

o vetor $VetAlter$ criptografado pelo K256-CBC.

3. medir a distância de Hamming, **separadamente**, entre **cada** bloco $BlC(k)$ de 256 bits de $VetEntraC$ e o correspondente bloco $BlAlterC(k)$ de 256 bits de $VetAlterC$. Para 4 blocos de 256 bits, tem-se 4 medidas de distância, sendo cada medida chamada, digamos, $H(k)$ para cada par de blocos $BlC(k), BlAlterC(k)$. Ou seja, para $k = 1, 2, 3, 4$, $H(k) =$Hamming$(BlC(k), BlAlterC(k))$.

4. estas medidas de distância de Hamming $H(k)$ devem ser acumuladas em somas chamadas, digamos, $SomaH(k)$. Para 4 blocos de 256 bits, tem-se 4 somas cumulativas, sendo que:

   (a) $SomaH(1)$ acumula 256 valores de $H(1)$ correspondentes a $j = 1, 2, 3..., 256$ (para $j > 256$ $H(1) = 0$ pois $BlC(1) = BlAlterC(1)$)

   (b) $SomaH(2)$ acumula $2*256 = 512$ valores de $H(2)$ correspondentes a $j = 1, 2, 3..., 256, 257..., 512$ (para $j > 2*256$ $H(2) = 0$ pois $BlC(2) = BlAlterC(2)$ e $H(1) = 0$ também pois $BlC(1) = BlAlterC(1)$)

   (c) $SomaH(3)$ acumula $3*256 = 768$ valores de $H(3)$ correspondentes a $j = 1, 2, 3...., 768$

   (d) $SomaH(4)$, acumula $4*256 = 1024$ valores de $H(4)$ correspondentes a $j = 1, 2, 3..., 1024$.

5. de forma análoga às somas $SomaH(k)$, o programa deve calcular os valores mínimo e máximo de $H(1), H(2), ...$

No final o programa deve imprimir uma tabela contendo os valores máximos, mínimos e médios das distâncias de Hamming entre **cada** bloco criptografado de 256 bits $BlC(k)$ e $BlAlterC(k)$, conforme o Algoritmo K256, no modo CBC. Para 4 blocos de 256 bits, o programa deve imprimir 4 valores máximos, 4 mínimos, e 4 médias.

Item 2: Efetuar o Item 1 uma outra vez, trocando a alteração do j-ésimo bit por alteração **simultânea** do j-ésimo e do $(j + 8)$-ésimo bits. Isso detetaria uma provável compensação de bits na saída, devido a dois bytes consecutivos alterados na entrada.

2 Definição de K256

2.1 As três operações básicas

Neste projeto há três operações distintas sobre 2^{64} elementos (*i.e.*, oito bytes). Se A, B, C denotam três elementos de 64 bits, as três operações são:

1. Ou-exclusivo (XOR) sobre 64 bits, que será representada pelo símbolo $\oplus$, *i.e.*, $A = B \oplus C$; note que $B \oplus C \oplus C = B$, ou seja, conhecendo-se A e C pode-se obter B.

2. Soma mod 2^{64}, que é equivalente à soma usual em que o bit mais à esquerda correspondente ao valor 2^{64} deve ser sempre igual a zero após a soma; esta operação será denotada pelo símbolo $\boxplus$, *i.e.*, $A = B \boxplus C$; note que se $\overline{C}$ é o inverso de C mod 2^{64} (*i.e.*, $\overline{C} + C = 2^{64} = 0 \bmod 2^{64}$), então $B \boxplus C \boxplus \overline{C} = B$; ou seja, conhecendo-se A e $\overline{C}$ pode-se obter B.

3. A terceira operação é representada pelo símbolo $\odot$, e é um pouco mais complicada que as anteriores. Seja $y = f(x)$ a função seguinte que mapeia um byte $x \in \{0,1\}^8$ para um byte $y \in \{0,1\}^8$:

$$y = f(x) = 45^x \bmod 257 \ (y = 0 \text{ se } x = 128, \text{ pois } 45^{128} \bmod 257 = 256)$$

   Por exemplo: $45^{31} \bmod 257 = 247$

   i. Observe que 257 é primo e 45 é gerador do corpo $GF(257)$, *i.e.*, $45^x \bmod 257$ para $x = 0, 1, 2, ..., 256$ gera todos os elementos de $GF(257)$.
   ii. A função inversa de $f()$, $x = f^{-1}(y)$, é definida a seguir: $x = f^{-1}(y) = \log_{45} y$ ($x = 128$ se $y = 0$, para ser consistente com a operação anterior) *i.e.*, $\log_{45}(45^x \bmod 257) = x$. Por exemplo $\log_{45} 247 = 31$.
   iii. Recomendamos que estas duas funções sejam previamente calculadas e tabeladas na forma $exp[x] = y$ e $log[y] = x$ onde $exp[]$ e $log[]$ são vetores de 256 posições, para $x, y = 0, 1, 2, ..., 255$. Desta forma, economiza-se tempo, pois consultar estes vetores é mais rápido do que calcular toda vez que se necessitar de um valor. Note que uma vez calculado o valor de $exp[i]$, podemos definir $log[exp[i]] = i$.
   iv. Para A, B, C de 64 bits, $A = B \odot C$ significa:
   - dividir os 64 bits de B em 8 bytes de 8 bits: $B_1||B_2||B_3||B_4||B_5||B_6||B_7||B_8$; dividir da mesma forma C em $C_1||C_2||C_3||C_4||C_5||C_6||C_7||C_8$;
   - Cada byte do resultado $A = A_1||A_2||A_3||A_4||A_5||A_6||A_7||A_8 = B \odot C$ é obtido da seguinte forma: para $j = 1, 2, ...8 : A_j = f(B_j) \oplus f(C_j)$.

2.2 Geração das subchaves

A partir da chave secreta K de 256 bits são geradas 52 subchaves de 64 bits que chamaremos $K_1, K_2, K_3, ..., K_{52}$. As primeiras 4 subchaves são geradas simplesmente considerando os primeiros 64 bits da esquerda para a direita de K como sendo K_1, os 64 bits seguintes de K como sendo K_2, e assim por diante, os últimos 64 bits de K como sendo K_8.

256 bits $\rightarrow$	64	64	64	64
de K	K_1	K_2	K_3	K_4

A seguir as 4 subchaves seguintes são geradas de forma análoga, mas a K_5 começando após 7 bits à direita do extremo esquerdo de K, a K_6 começando após $7 + 64 = 71$ bits à direita do extremo esquerdo de K, e assim por diante, a K_7 começa após $7 + 2 \times 64 = 135$ bits à direita do extremo esquerdo de K. Por enquanto a K_8 contém apenas 57 bits. Os 7 bits restantes da K_{15} começam no primeiro bit de K como se uma cópia de K estivesse concatenado à direita de K como no esquema abaixo. Isto equivale a ter deslocado circularmente para a *esquerda* a chave K, de 7 posições, *antes* de iniciar a geração de cada uma das 4 subchaves, e K_5 começar no *primeiro* bit.

256 bits de K	$\rightarrow$	7	64 K_5	64 K_6	64 K_7	57 parte K_8

7 parte K_8	64	64	64	57	$\leftarrow$ 256 bits de K

As 4 subchaves seguintes são geradas de forma análoga mas a K_9 começando após 14 bits à direita do extremo esquerdo de K, a K_{10} começando após $14 + 64 = 78$ bits à direita do extremo esquerdo de K, e assim por diante, a K_{12} começa após $14 + 3 \times 64 = 206$ bits à direita do extremo esquerdo de K, e termina no 14^o bit de K como se uma cópia de K estivesse concatenado à direita de K. Isto equivale a ter deslocado circularmente para a *esquerda* a chave K, de 7 posições além das 7 anteriores, *antes* de iniciar a geração de cada uma das 4 subchaves, e K_9 começar no *primeiro* bit.

256 bits K	$\rightarrow$	14	64 K_9	64 K_{10}	64 K_{11}	50 parte K_{12}

14 parte K_{12}	64	64	64	50	$\leftarrow$ 256 bits de K

As subchaves K_{13} a K_{16} são geradas como no esquema a seguir:

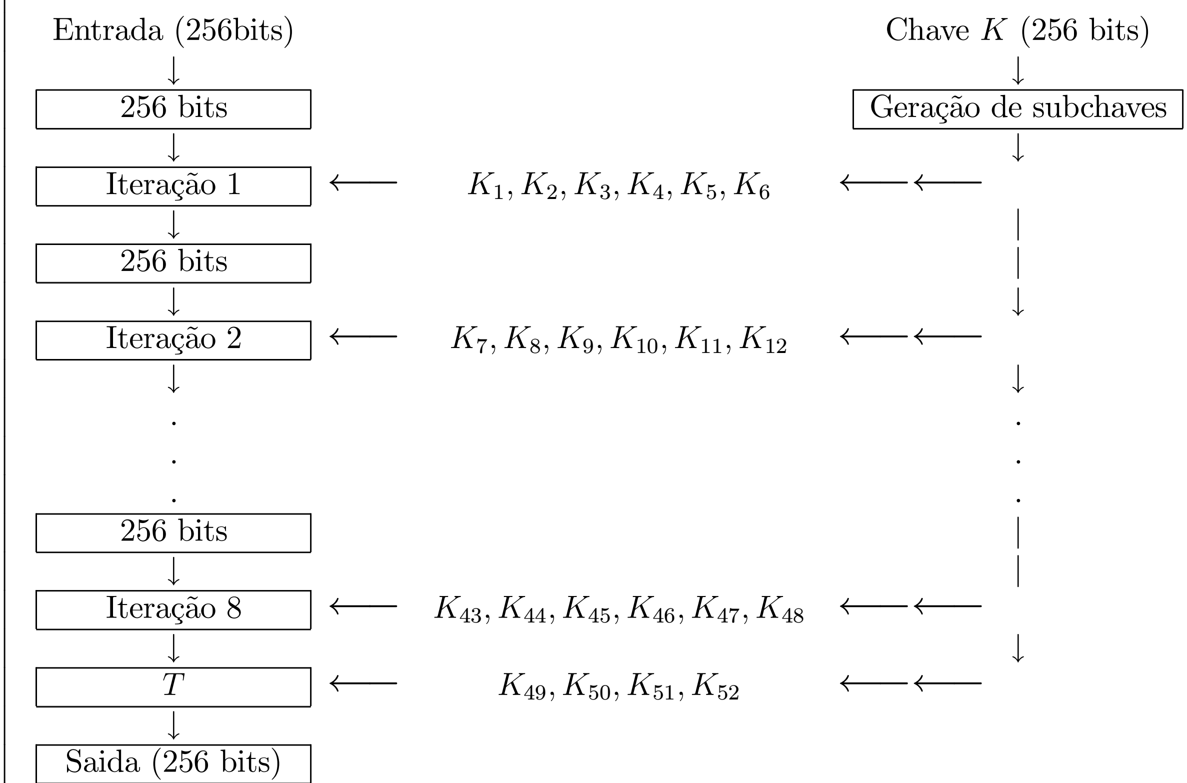
256 bits de K	$\rightarrow$	21	64 K_{13}	64 K_{14}	64 K_{15}	43 parte K_{16}

21 parte K_{16}	64	64	64	43	$\leftarrow$ 256 bits de K

As subchaves K_{17} a K_{52} são geradas analogamente.

2.3 Uma iteração (*round*) do K256

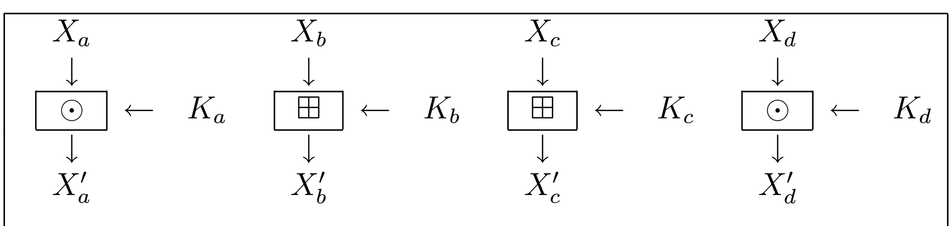
K256 possui 8 iterações (ou *rounds*), de forma semelhante ao DES, e uma transformação final que chamaremos T. A transformação T utiliza as últimas 4 subchaves: $K_{49}, K_{50}, K_{51}, K_{52}$, da maneira que descreveremos mais adiante. Cada iteração utiliza 6 subchaves e possui duas partes que descreveremos a seguir.



Esquema geral do K256

2.3.1 Primeira parte de uma iteração

Esta parte utiliza 4 subchaves que chamaremos K_a, K_b, K_c, K_d. A sua entrada é de 256 bits, tratada como 4 subentradas de 64 bits que chamaremos X_a, X_b, X_c, X_d. Após certas operações aplicadas sobre esta entrada, a sua saída será constituída de novas versões destes X_a, X_b, X_c, X_d que chamaremos X'_a, X'_b, X'_c, X'_d. Na primeira iteração, $K_a = K_1, K_b = K_2, K_c = K_3, K_d = K_4$, e na segunda iteração $K_a = K_7, K_b = K_8, K_c = K_9, K_d = K_{10}$, e assim por diante.



Primeira parte de uma iteração

As operações são as seguintes:

1. X'_a é $X_a \odot K_a$

2. X'_b é $X_b \boxplus K_b$

3. X'_c é $X_c \boxplus K_c$

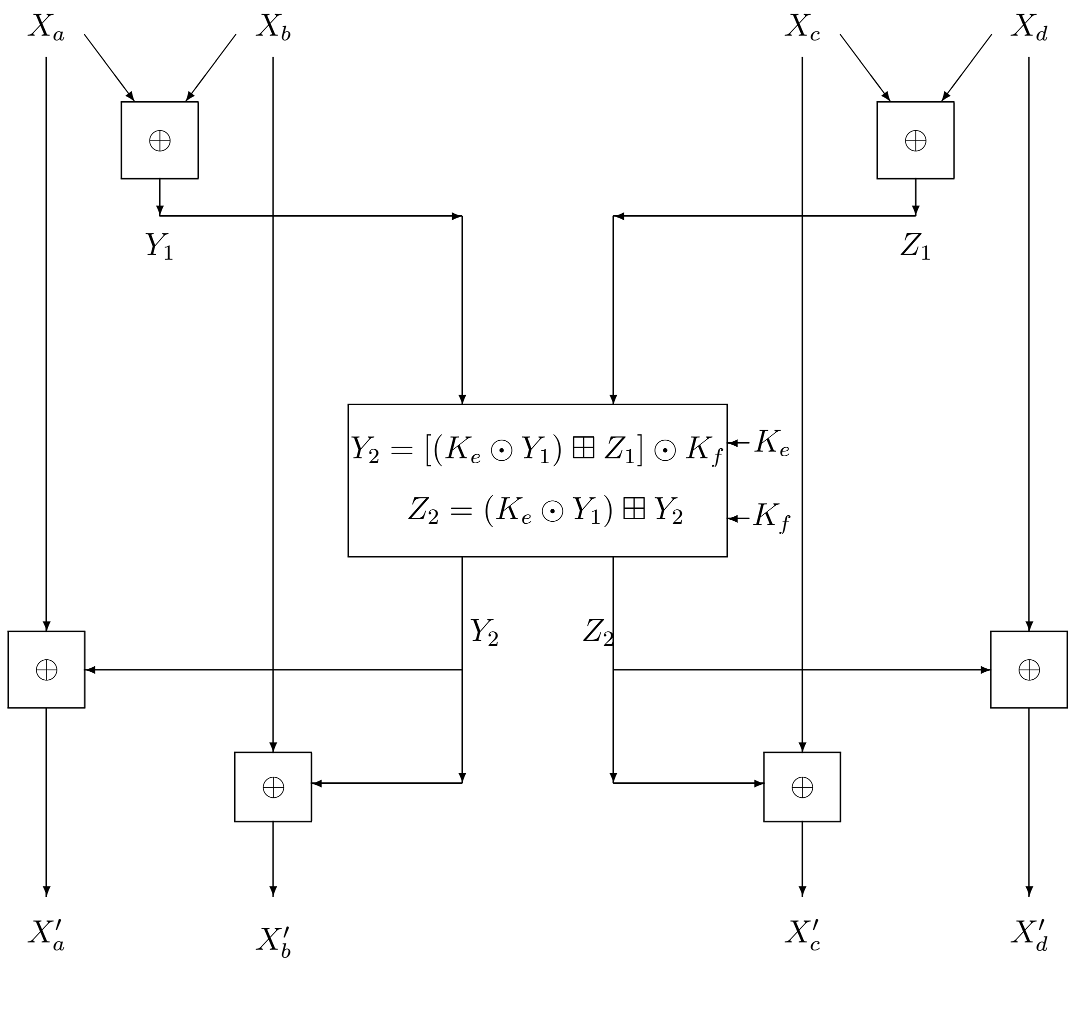
4. X'_d é $X_d \odot K_d$

Note que o resultado desta parte, em ordem, é X'_a, X'_b, X'_c, X'_d.

Observe que estas 4 operações são inversíveis. Para se obter X_a a partir de X'_a basta termos calculado previamente a inversa multiplicativa K_a^{-1}pois $X'_a \odot K_a^{-1} = X_a \odot K_a \odot K_a^{-1} = X_a$. De forma análoga, pode-se obter X_d a partir de X'_d. E para se obter X_c a partir de X'_b basta termos calculado previamente a inversa aditiva $\overline{K_c}$, pois $X'_b \boxplus \overline{K_c} = X_c \boxplus K_c \boxplus \overline{K_c} = X_c$. E de forma análoga, pode se obter X_b a partir de X'_c.

2.3.2 Segunda parte de uma iteração

Essa parte utiliza 2 subchaves que chamaremos K_e, K_f. Sua entrada é a saída da primeira parte, de 256 bits, tratada de novo como 4 subentradas de 64 bits que chamaremos X_a, X_b, X_c, X_d. Após outras operações aplicadas sobre esta entrada, sua saída será constituída de novas versões destes X_a, X_b, X_c, X_d que, de novo, chamaremos X'_a, X'_b, X'_c, X'_d. Na primeira iteração, $K_e = K_5, K_f = K_6$, e na segunda iteração $K_e = K_{11}, K_f = K_{12}$, e assim por diante.



Segunda parte de uma iteração

1. Inicialmente são calculados dois valores intermediários chamados Y_1 e Z_1 da seguinte forma:
   (a) $Y_1 = X_a \oplus X_b$
   (b) $Z_1 = X_c \oplus X_d$
2. A seguir outros dois valores intermediários chamados Y_2 e Z_2 são calculados:
   (a) $Y_2 = [(K_e \odot Y_1) \boxplus Z_1] \odot K_f$
   (b) $Z_2 = (K_e \odot Y_1) \boxplus Y_2$
3. E os valores X'_a, X'_b, X'_c, X'_d são calculados da seguinte maneira:
   (a) $X'_a = X_a \oplus Y_2$
   (b) $X'_b = X_b \oplus Y_2$

(c) $X'_c = X_c \oplus Z_2$

(d) $X'_d = X_d \oplus Z_2$

Uma observação *muito importante* aqui é que a inversa desta segunda parte é a própria! Isto é, se a entrada para a segunda parte for X'_a, X'_b, X'_c, X'_d com subchaves K_e, K_f, obtêm-se X_a, X_b, X_c, X_d de volta!! Para justificarmos isso, observe os seguintes passos:

1. Tendo X'_a, X'_b na entrada à esquerda, o início (Passo (1) anterior) desta segunda parte efetua $X'_a \oplus X'_b = (X_a \oplus Y_2) \oplus (X_b \oplus Y_2) = X_a \oplus X_b = Y_1$. E tendo X'_c, X'_d na entrada à direita, o início desta segunda parte efetua $X'_c \oplus X'_d = (X_c \oplus Z_2) \oplus (X_d \oplus Z_2) = X_c \oplus X_d = Z_1$. Portanto, os valores Y_1 e Z_1 são reconstruídos exatamente como no Passo (1) anterior.

2. O passo para recalcular Y_2 e Z_2 é também exatamente igual ao Passo (2) anterior.

   (a) $Y_2 = [(K_e \odot Y_1) \boxplus Z_1] \odot K_f$

   (b) $Z_2 = (K_e \odot Y_1) \boxplus Y_2$

3. Esse passo também é igual ao Passo (3) anterior, pois:

   (a) $X'_a \oplus Y_2 = (X_a \oplus Y_2) \oplus Y_2 = X_a$

   (b) $X'_b \oplus Y_2 = (X_b \oplus Y_2) \oplus Y_2 = X_b$

   (c) $X'_c \oplus Z_2 = (X_c \oplus Z_2) \oplus Z_2 = X_c$

   (d) $X'_d \oplus Z_2 = (X_d \oplus Z_2) \oplus Z_2 = X_d$

Em resumo, a criptografia nesta segunda parte é exatamente igual à decriptografia, não necessitando qualquer cálculo de chave inversa (que a primeira parte exige).

2.3.3 A última transformação T

Após 8 iterações da primeira e segunda partes como descrito acima, o resultado X'_a, X'_b, X'_c, X'_d é fornecido como entrada para a última transformação T.

Como mencionado anteriormente, a transformação T utiliza as últimas 4 subchaves: $K_{49}, K_{50}, K_{51}, K_{52}$. E esta tranformação é quase igual à primeira parte de uma iteração, exceto que K_{50} é aplicado sobre X'_c e K_{51} é aplicado sobre X'_b:

1. X_a^{FINAL} é $X'_a \odot K_{49}$

2. X_b^{FINAL} é $X'_b \odot K_{50}$

3. X_c^{FINAL} é $X'_c \boxplus K_{51}$

4. X_d^{FINAL} é $X'_d \boxplus K_{52}$

FIM