DCC–IME–USP

Routo Terada
Sala 204C Tel.: 3091 5918
e-mail rt@ime.usp.br

MAC 5723 - 336 - Criptografia

Primeiro Semestre de 2012

Exercício-Programa Data de entrega: veja no sistema PACA

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, se for o caso.
  - 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 **diminuída** de um ponto a cada dia "corrido" de atraso na entrega.

1 Programa de criptografia K128

Este exercício consiste em elaborar um programa para criptografar e decriptografar arquivos de qualquer comprimento, com o Algoritmo K128 descrito a seguir. A chave principal K de 128 bits é derivada de uma senha, como descrito abaixo.

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 um arquivo:
  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 (ver Item 1 abaixo):
  programa -1 -i <arquivo de entrada> -p <senha>
- Modo (4) Para calcular aleatoriedade pelo método 2 (ver 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) fica à escolha de cada aluno.

A senha a ser digitada: a senha A no parâmetro -p <senha> deve conter pelo menos **8** caracteres, sendo A com **pelo menos** 2 letras e 2 algarismos decimais;

Geração da chave K de 128 bits a partir da senha: se a senha A digitada possuir menos que 16 caracteres (i.e., 16 bytes), a chave K de 128 bits deve ser derivada de A concatenando-se A com ela própria até completar 16 bytes (128 bits).

Leitura e gravação de arquivo: O seu programa deve ler do disco o arquivo de entrada Entra, e deve gravar o arquivo de saída Sai correspondente a Entra criptografado ou decriptografado com a senha A, no modo CBC (Cipher Block Chaining), que consiste em encadear um bloco de 128 bits com o bloco anterior critografado da maneira vista em aula, e também descrito no livro Segurança de Dados.

1. No modo CBC, utilizar bits iguais a UM como Valor Inicial - VI.
2. Você 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, ou alguma música MP3.
3. Se o último bloco a ser criptografado não possuir comprimento igual a 64 bits, completá-lo com bits iguais a UM.
4. Verifique se o arquivo decriptografado Sai possui o mesmo comprimento que o arquivo original Entra. Algum bloco criptografado de Sai deve conter o comprimento do arquivo original Entra. Mas tome CUIDADO para o local deste bloco não prejudicar o tempo total de execução do seu programa.

O seu programa deve também efetuar os itens seguintes:

Item 1: Medir a aleatoriedade do K128 da seguinte maneira.

Seja $VetEntra$ um vetor lido de um arquivo de entrada para a memória principal com exatamente 1024 bits (i.e., 8 blocos de 128 bits, de modo que

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

sendo cada bloco $Bl()$ de 128 bits e $|VetEntra| = 8 * 128 = 1024$). **Ignorar** os bits restantes do arquivo, considerar apenas os 1024 bits iniciais. Veja ilustração a seguir:

Número do bloco k	1	2	3	4	...
Valores de j, posição de bit	$j = 1, 2, ...128$	$j = 129, ..., 256$	$j = 257, ..., 384$	$j = 385, ..., 512$	...
$VetEntra$	$Bl(1)$	$Bl(2)$	$Bl(3)$	$Bl(4)$	...
$VetEntraC$ (criptografado)	$BlC(1)$	$BlC(2)$	$BlC(3)$	$BlC(4)$	...
$H(VetEntra, VetAlter) = 1$					...
$VetAlter$	$BlAlter(1)$	$BlAlter(2)$	$BlAlter(3)$	$BlAlter(4)$	...
$VetAlterC$ (criptografado)	$BlAlterC(1)$	$BlAlterC(2)$	$BlAlterC(3)$	$BlAlterC(4)$	...
Valores j que afetam $H(k)$	$j = 1, 2, ...128$	$j = 1, 2, ...256$	$j = 1, 2, ...384$	$j = 1, 2, ...512$	...
Dist $H(BlC, BlAlterC)$	$H(1), 64$ vals.	$H(2), 128$ vals.	$H(3), 192$, vals	$H(4), 256$ vals.	...
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. Esse ÚNICO j-ésimo bit pode ser do primeiro bloco $BlAlter(1)$, ou do segundo $BlAlter(2)$, ou de qualquer outro bloco. No caso de apenas 8 blocos, $j = 1, 2, 3, ...1024$. Por exemplo, no caso de $j = 2$, $Bl(1) = (0234567812345678)_{16}$, $Bl(2) = (1234567812345678)_{16}$, ... e

   $$VetAlter = BlAlter(1)||BlAlter(2)||... = 4234567812345678||1234567812345678||...$$

   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 K128-CBC. E seja

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

   o vetor $VetAlter$ criptografado pelo K128-CBC.

3. medir a distância de Hamming, **separadamente**, entre **cada** bloco $BlC(k)$ de 128 bits de $VetEntraC$ e o correspondente bloco $BlAlterC(k)$ de 128 bits de $VetAlterC$. Para 8 blocos de 128 bits, tem-se 8 medidas de distância, sendo cada medida chamada, digamos, $H(k)$ para cada par de blocos $BlC(k), BlAlterC(k)$. Ou seja, para $k = 1, 2, 3, 4, ...8$, $H(k)$ =Hamming$(BlC(k), BlAlterC(k))$.

4. estas medidas de distância de Hamming $H(k)$ devem ser acumuladas em somas chamadas, digamos, $SomaH(k)$. Para 8 blocos de 128 bits, tem-se 8 somas cumulativas, sendo que:

   (a) $SomaH(1)$ acumula 128 valores de $H(1)$ correspondentes a $j = 1, 2, 3..., 128$ (para $j \geq 129$ $H(1) = 0$ pois $BlC(1) = BlAlterC(1)$)

   (b) $SomaH(2)$ acumula $2*128 = 256$ valores de $H(2)$ correspondentes a $j = 1, 2, 3..., 128$, $129, ..., 256$ (para $j \geq 257$ $H(2) = 0$ pois $BlC(2) = BlAlterC(2)$ e $H(1) = 0$ pois $BlC(1) = BlAlterC(1)$)

   (c) $SomaH(3)$ acumula $3*128 = 384$ valores de $H(3)$ correspondentes a $j = 1, 2, 3...., 384$

   (d) $SomaH(4)$, acumula $4*128 = 512$ valores de $H(4)$ correspondentes a $j = 1, 2, 3..., 512$

   (e) E assim por diante, até $SomaH(8)$, com $8 * 128 = 1024$

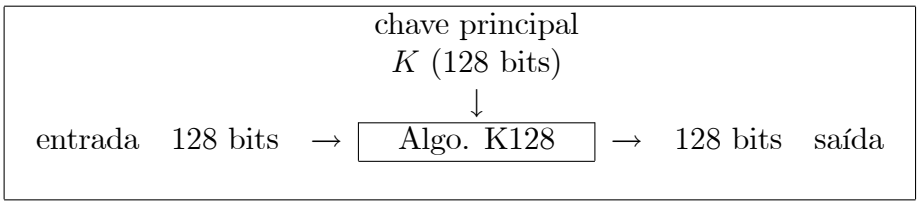
5. de forma análoga às somas $SomaH(k)$, o programa deve calcular os valores mínimo e máximo de $H(1), H(2), ...$

No final o programa deve imprimir uma tabela contendo os valores máximos, mínimos e médios das distâncias de Hamming entre **cada** bloco criptografado de 128 bits $BlC(k)$ e $BlAlterC(k)$, conforme o Algoritmo K128, no modo CBC. Para 8 blocos de 128 bits, o programa deve imprimir 8 valores máximos, 8 mínimos, e 8 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. **Exercício**: por quê essa compensação pode ocorrer no modo CBC?

2 Algoritmo K128

Implementar o Algoritmo criptográfico K128, com **chave principal** K de 128 bits, e com blocos de entrada e saída de 128 bits. Você deve **deduzir** o algoritmo inverso do K128.



O número R de iterações (*rounds*) é variável, mas neste exercício você deve utilizar $R = 12$.

Cada iteração (ou round) da criptografia ou da decriptografia exige 4 subchaves de 64 bits. Para uma iteração r, $r = 1, ..., R$ estas 4 subchaves são chamadas $k_{4r-3}, k_{4r-2}, k_{4r-1}, k_{4r}$. A transformação final $T()$ exige duas subchaves de 64 bits, k_{4R+1}, k_{4R+2}.

O comprimento da chave principal chamada K é 128 bits. O número de subchaves desejado é $4R + 2 = 4 * 12 + 2 = 50$.

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

   (a) 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 $\log_{45} 247 = 31$.

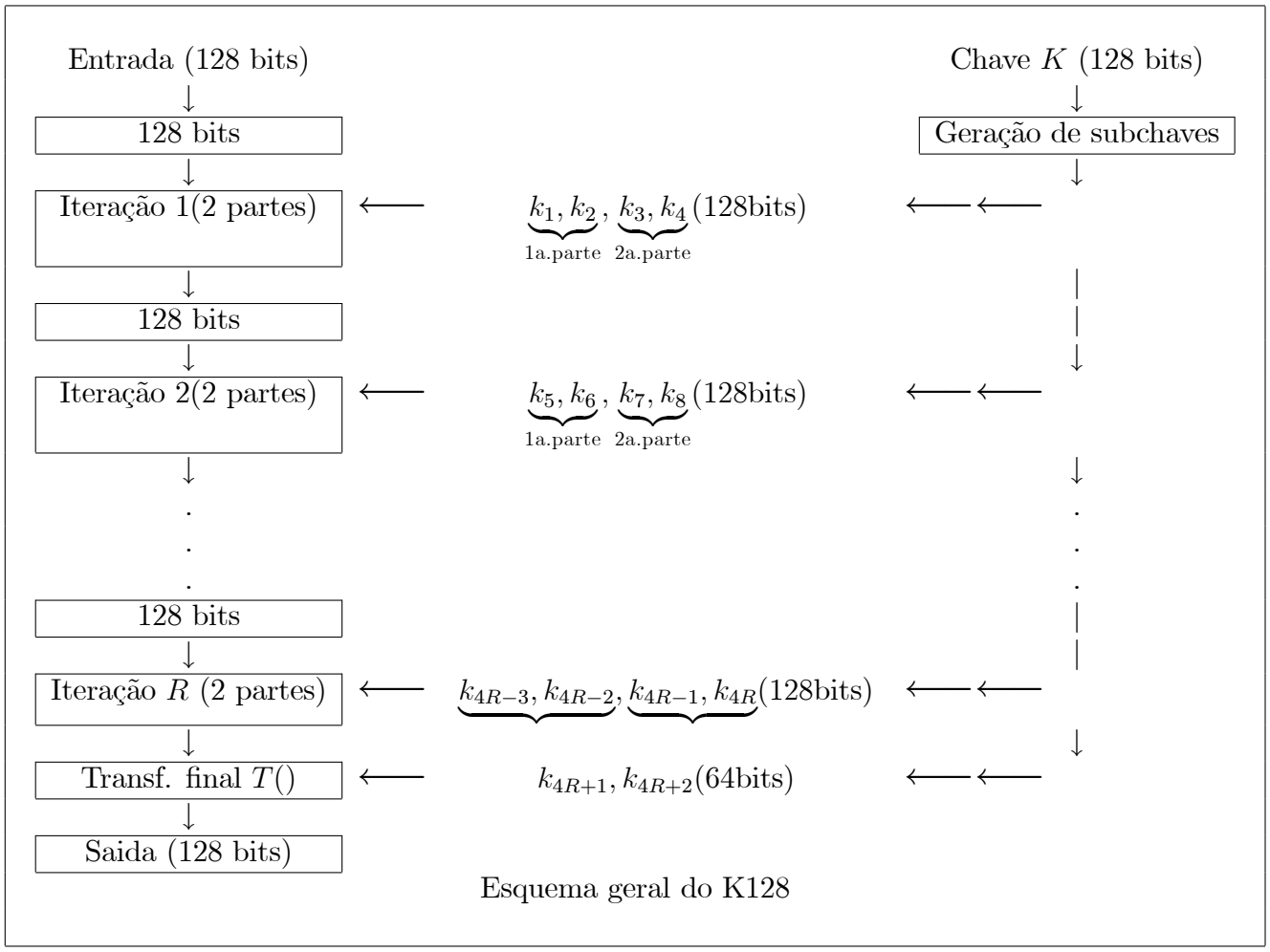
   iii. Recomendamos que estas duas funções sejam previamente calculadas e tabeladas na forma $exp[x] = y$ e $log[y] = x$ onde $exp[]$ e $log[]$ são vetores de 256 posições, para $x, y = 0, 1, 2, ..., 255$. Desta forma, economiza-se tempo, pois consultar estes vetores é mais rápido do que calcular toda 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 Uma iteração (*round*) do K128

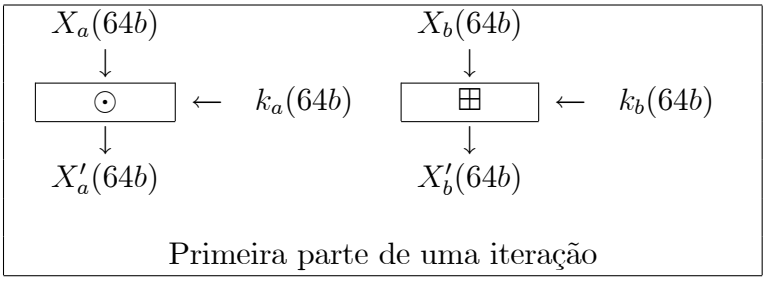
K128 possui $R = 12$ iterações (ou *rounds*) e uma transformação final que chamaremos T. A transformação T utiliza as últimas 2 subchaves: k_{4R+1}, k_{4R+2}, da maneira que descreveremos mais adiante. Cada iteração utiliza 4 subchaves e possui duas partes que descreveremos a seguir.



Esquema geral do K128

2.2.1 Primeira parte de uma iteração

Esta parte utiliza 2 subchaves que chamaremos k_a, k_b. A sua entrada é de 128 bits, tratada como 2 subentradas de 64 bits que chamaremos X_a, X_b. Após certas operações aplicadas sobre esta entrada, a sua saída será constituída de novas versões destes X_a, X_b que chamaremos X'_a, X'_b, total de 128 bits Na primeira iteração, $k_a = k_1, k_b = k_2$, e na segunda iteração $k_a = k_5, k_b = k_6$, 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 $X_b \boxplus k_b$

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

Observe que estas 2 operações são inversíveis. Para se obter X_a a partir de X'_a basta termos calculado previamente a inversa multiplicativa k_a^{-1}pois $X'_a \odot k_a^{-1} = X_a \odot k_a \odot k_a^{-1} = X_a$. E para se obter X_b a partir de X'_b basta termos calculado previamente a inversa aditiva $\overline{k_b}$, pois $X'_b \boxplus \overline{k_b} = X_b \boxplus k_b \boxplus \overline{k_b} = X_b$.

2.2.2 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 128 bits, tratada de novo como 2 subentradas de 64 bits que chamaremos X_e, X_f. Após outras operações aplicadas sobre esta entrada, sua saída será constituída de novas versões destes X_e, X_f que chamaremos X'_e, X'_f. Na primeira iteração, $k_e = k_3, k_f = k_4$, e na segunda iteração $k_e = k_7, k_f = k_8$, e assim por diante.

1. Inicialmente são calculados dois valores intermediários chamados Y_1 e Z_1 da seguinte forma:
   (a) $Y_1 = X_e \oplus X_f$
   (b) $Z_1 = X_e \boxplus X_f$
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'_e, X'_f são calculados da seguinte maneira:
   (a) $X'_e = X_e \oplus Y_2$
   (b) $X'_f = X_f \boxplus Z_2$

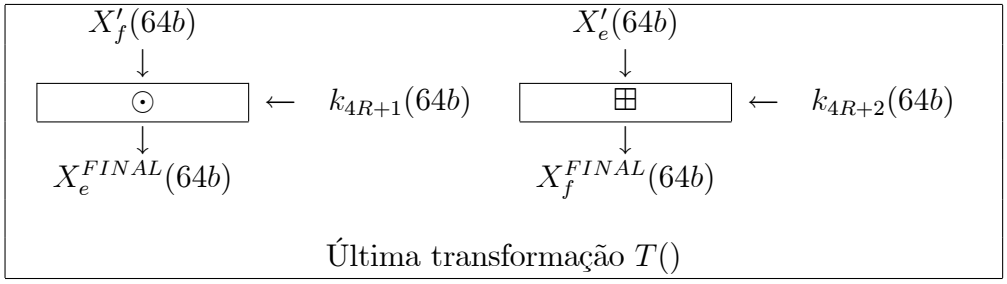
2.2.3 A última transformação T

Após $R = 12$ iterações da primeira e segunda partes como descrito acima, o resultado X'_e, X'_f é fornecido como entrada para a última transformação T.

Como mencionado anteriormente, a transformação T utiliza as últimas 2 subchaves: k_{4R+1}, k_{4R+2}. E esta tranformação é semelhante à primeira parte de uma iteração, exceto que k_{4R+1} é aplicado sobre X'_f e k_{4R+2} é aplicado sobre X'_e:

1. X_e^{FINAL} é $X'_f \odot k_{4R+1}$

2. X_f^{FINAL} é $X'_e \boxplus k_{4R+2}$



3 Algoritmo de geração de subchaves

No início este algoritmo os 128 bits da chave K em duas variáveis de 64 bits, L_0 e L_1. A seguir expande L_0, L_1 para obter L_2, L_3,...L_{4R+2}.

- Seja $\boxplus$ a operação de soma aritmética sobre operandos de 64 bits, módulo 2^{64}.
- Seja $\beta << \alpha$ rotação (deslocamento circular) de α bits para a esquerda dos 64 bits de β.
- $0x(...)$ denota um valor em notação hexadecimal.

Algoritmo de geração de subchaves
Entrada: chave principal K de 128 bits.
Saída: $4R + 2$ subchaves de 64 bits $k_1, k_3, ...k_{4R+2}$.

1. $L_0 \leftarrow$"metade esquerda da chave K, 64 bits"; $L_1 \leftarrow$"metade direita da chave K, 64 bits"
2. **para** $j = 2, 3, ...4R + 2$ **faça**: $L_j \leftarrow L_{j-1} \boxplus 0x(9e3779b97f4a7c15)$; (* 64 bits *)
3. $k_0 \leftarrow 0x(8aed2a6bb7e15162)$; (* 64 bits *)
4. **para** $j = 1, 2, ...4R + 2$ **faça**: $k_j \leftarrow k_{j-1} \boxplus 0x(7c159e3779b97f4a)$;
5. $i \leftarrow 0; j \leftarrow 0; A \leftarrow 0; B \leftarrow 0;$
6. **para** $s = 1, 2, 3, ...4R + 2$ **faça** {
   (a) $k_i \leftarrow (k_i \boxplus A \boxplus B) \lll 3; A \leftarrow k_i; i \leftarrow i + 1$
   (b) $L_j \leftarrow (L_j \boxplus A \boxplus B) \lll (A \boxplus B); B \leftarrow L_j; j \leftarrow j + 1$
   (c) }
7. A saída é $k_1, k_2, ...k_{4R+2}$

FIM