DCC–IME–USP

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

MAC 336 - Criptografia e Segurança de Dados

Primeiro Semestre de 2017

Exercício-Programa 1 Data de entrega : veja no paca.ime.usp.br.

Observações

- Este exercício é para ser feito *individualmente*.
- Entregue no sistema Panda 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, compilador e sistema operacional foi usado (modelo, versão, etc..),
    - 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 diminuida de um ponto a cada dia "corrido" de atraso na entrega.

Este exercício-programa consiste em:

1. Elaborar um programa para criptografar e decriptografar arquivos de qualquer comprimento, com o Algoritmo K128, descrito na Seção 4, na página 4.
2. O programa deve também medir a aleatoridade do algoritmo, conforme descrito a partir da página 7.

Implementar o Algoritmo criptográfico K128, com **chave principal** K de 128 bits, e com blocos de entrada e saída de 64 bits. Você deve **deduzir** o algoritmo inverso do K128. A chave principal K de 128 bits é derivada de uma senha, como descrito na Seção 2, página 3.

1 Execução na 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>

2 Senha e chave principal K

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).

3 Algoritmo de geração de subchaves k_j

Seja R o número de iterações (rounds). O algoritmo no início divide 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_{2R+1}.

- Seja $\boxplus$ a operação de soma aritmética sobre operandos de 64 bits, módulo 2^{64}.
- Seja $\beta << \alpha$ rotação 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, número de rounds R.

Saída: $2R + 1$ subchaves de 64 bits $k_1, k_3, ...k_{2R+1}$.

1. $L_0 \leftarrow$"metade esquerda da chave K"; $L_1 \leftarrow$"metade direita da chave K"
2. **para** $j = 2, 3, ...2R + 1$ **faça**: $L_j \leftarrow L_{j-1} \boxplus 0x(9e3779b97f4a7c15)$
3. $k_0 \leftarrow 0x(b7e151628aed2a6b)$;
4. **para** $j = 1, 2, ...2R + 1$ **faça**: $k_j \leftarrow k_{j-1} \boxplus 0x(7f4a7c159e3779b9)$
5. $i \leftarrow 0; j \leftarrow 0; A \leftarrow 0; B \leftarrow 0$
6. **para** $s = 1, 2, 3, ...2R + 1$ **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 $k_1, k_2, ...k_{2R+1}$

4 Definição do Algoritmo K128

Todas as operações, como soma, subtração, logaritmo, são sobre operandos em bytes de 8 bits, resultando um byte, como veremos mais tarde. Por exemplo, sendo x e y valores inteiros positivos armazenados cada um em um byte, $x + y$ será um valor de 8 bits.

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

O comprimento da chave principal chamada K é 128 bits.

Cada iteração (ou round) da criptografia/decriptografia exige 2 subchaves de 64 bits. Para $r = 1, ..., R$ estas duas subchaves são chamadas k_{2r-1}, k_{2r}. A transformação final $T()$ exige uma única subchave de 64 bits, k_{2R+1}. O número de subchaves desejado é $25 = 2R + 1 = 2 * 12 + 1$.

Um bloco B de entrada, de 64 bits, é dividido em 8 sub-blocos $B_1, B_2, B_3, ...B_8$. E então, R *rounds* são aplicados a estes sub-blocos, e depois uma transformação final T é aplicada, obtendo-se a saída final.

4.1 Descrição de uma iteração

Entrada para um *round*: duas subchaves k_{2r-1} e k_{2r}, sendo cada uma de 64 bits e 8 sub-blocos $B_1, B_2, B_3, ...B_8$, sendo cada um de 8 bits

4.1.1 Primeiro passo

Primeiramente cada sub-bloco é submetido a XOR (ou-exclusivo) ou somado a bytes da subchave k_{2r-1} da seguinte forma (sendo $K^1_{2r-1}, K^2_{2r-1}, K^3_{2r-1}, K^4_{2r-1}, K^5_{2r-1}, K^6_{2r-1}, K^7_{2r-1}, K^8_{2r-1}$ os 8 bytes de k_{2r-1},):
$B_1(XOR)K^1_{2r-1} = C_1, B_2 + K^2_{2r-1} = C_2, B_3 + K^3_{2r-1} = C_3, B_4(XOR)K^4_{2r-1} = C_4,$
$B_5(XOR)K^5_{2r-1} = C_5, B_6 + K^6_{2r-1} = C_6, B_7 + K^7_{2r-1} = C_7, B_8(XOR)K^8_{2r-1} = C_8$

4.1.2 Segundo passo

Os 8 bytes C_j são submetidos a dois tipos de operações, conforme a Figura 4.1.3 na página 6:

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

Observe que 257 é primo e 45 é elemento primitivo do corpo $GF(257)$, *i.e.*, $45^x \bmod 257$ para $x = 0, 1, 2, ..., 256$ gera todos os elementos de $GF(257)$. A segunda operação é a inversa da anterior, *i.e.*, $\log_{45}(45^x \bmod 257) = x$.

$$x = \log_{45} y \ (x = 128 \text{ se } y = 0, \text{ para ser consistente com a operação anterior})$$

Recomendamos que estas duas operações 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 $log[exp[i]] = i$.

4.1.3 Terceiro passo

Os blocos de um byte obtidos no Segundo Passo são agora submetidos a operações de soma e XOR com os bytes da chave k_{2r}, conforme a Figura 4.1.3 na página 6.

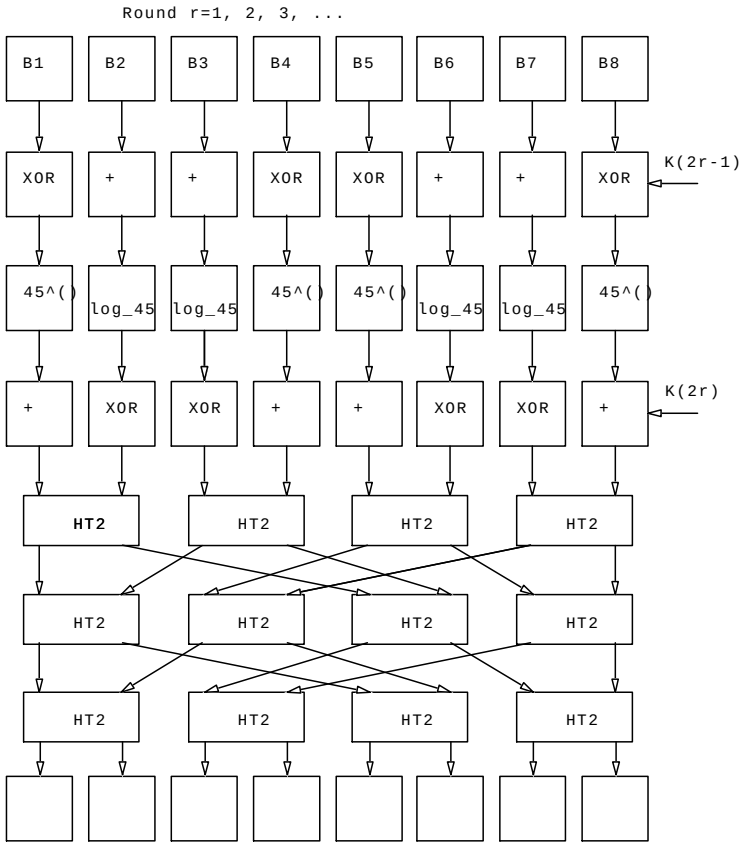

Ilustração de uma iteração (round)

4.1.4 Quarto passo

Finalmente, os 8 bytes são sumetidos à operação HT2 de 2 entradas a_1, a_2 e de duas saídas b_1, b_2 definida por:

$$\begin{aligned} b_1 &= (2a_1 + a_2) \bmod 256 \\ b_2 &= (a_1 + a_2) \bmod 256 \end{aligned}$$

conforme a Figura 4.1.3 na página 6.

4.2 Descrição da transformação final T

Depois de R *rounds*, os 8 blocos são submetidos a uma transformação final T que é exatamente igual ao Primeiro Passo descrito acima, só que a subchave utilizada deve ser a última: k_{2R+1}. E o resultado é a saída final do algoritmo K128.

5 O que o programa deve fazer

O seu programa deve ler do disco o arquivo de entrada Entra, que pode ser QUALQUER sequência de bits (música, figura, etc.), 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 64 bits com o bloco anterior criptografado da maneira vista em aula, e também descrito no livro Segurança de Dados.

O seu programa deve também efetuar os itens 1 e 2 descritos no final deste enunciado, Seção 7, página 7.

6 Modo CBC e testes

Regras:

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, ou MP3, JPG, etc..
3. Um programa-exemplo de leitura e gravação de arquivo em HD estará disponível na página da disciplina.
4. Se o último bloco a ser criptografado não possuir comprimento igual a 64 bits, completá-lo com bits iguais a UM, seguido pelo comprimento do arquivo original Entra. Se for o caso, um *último* bloco **extra** criptografado de Sai deve conter o comprimento do arquivo original Entra.
5. Verifique se o arquivo decriptografado Sai possui o mesmo comprimento que o arquivo original Entra.

7 Medidas de aleatoridade - Efeito Avalanche para blocos de 64 bits

Seja $VetEntra$ um vetor lido de um arquivo de entrada para a memória principal com pelo menos 512 bits (i.e., pelo menos 8 blocos de 64 bits, de modo que

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

cada bloco $Bl()$ de 64 bits e $|VetEntra| \geq 8 * 64 = 512$).
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 8 blocos, $j = 1, 2, 3, ...512$. Por exemplo, no caso de 8 bits em cada bloco, um único bit alterado na posição $j = 2$, $Bl(1) = 01010101$, $BlAlter(1) = 00110101, ...$ e

$$VetEntra = BlAlter(1)||BlAlter(2)||... = 01110101||...$$

$$VetAlter = BlAlter(1)||BlAlter(2)||... = 00110101||...$$

   ou seja diferem só no bit na posição 2, $Hamming(VetEntra, VetAlter) = 1$

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 $H(k) = Ham(BlC(k), BlAlterC(k))$, **separadamente**, entre **cada** bloco $BlC(k)$ de 64 bits de $VetEntraC$ e o correspondente bloco $BlAlterC(k)$ de 64 bits de $VetAlterC$. Veja ilustração nas páginas 9 e 10 para apenas os 4 blocos *iniciais*.

4. seja $SomaH(k)$ a soma acumulada destas medidas de distância de Hamming $H(k)$.Para 8 blocos de 64 bits, tem-se 8 somas acumuladas, sendo que:

   (a) $SomaH(1)$ acumula 64 valores de $H(1)$ correspondentes a $j = 1, 2, 3..., 64$ (para $j > 64$ $H(1) = 0$ pois $BlC(1) = BlAlterC(1)$). Veja ilustração nas páginas 9 e 10 para apenas os 4 blocos *iniciais*.

   (b) $SomaH(2)$ acumula $2 * 64 = 128$ valores de $H(2)$ correspondentes a $j = 1, 2, 3..., 64, 65..., 128$ (para $j > 2 * 64$ $H(2) = 0$ pois $BlC(2) = BlAlterC(2)$ e $H(1) = 0$ pois $BlC(1) = BlAlterC(1)$). Veja ilustração nas páginas 9 e 10 para apenas os 4 blocos *iniciais*.

   (c) $SomaH(3)$ acumula $3 * 64 = 192$ valores de $H(3)$ correspondentes a $j = 1, 2, 3...., 192$

   (d) $SomaH(4)$, acumula $4 * 64 = 256$ valores de $H(4)$ correspondentes a $j = 1, 2, 3..., 256$.

   (e) e assim por diante para $k = 5, 6, ...$

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

$VetEntra =$	$Bl(1)$	$Bl(2)$	$Bl(3)$	$Bl(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetEntraC$ (criptografado)=	$BlC(1) = A$	$BlC(2) = B$	$BlC(3) = C$	$BlC(4) = D$
$VetAlter =$	$BlAlter(1)$	$BlAlter(2)$	$BlAlter(3)$	$BlAlter(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetAlterC$ (criptografado)=	$BlAlterC(1) = A'$	$BlAlterC(2) = B'$	$BlAlterC(3) = C'$	$BlAlterC(4) = D'$
Posição do único bit alterado	$j = 1, 2, ...64$	$j = 65, ...128$	$j = 127, ...192$	$j = 193, ...256$
Distância de Hamming	$H(1) = Ham(A, A')$	$H(2) = Ham(B, B')$	$H(3)Ham(C, C')$	$H(4)Ham(D, D')$
Núm.de valores de $H(k)$	64	128	192	256
Soma acumulada de $H(k)$	$SomaH(1)$	$SomaH(2)$	$SomaH(3)$	$SomaH(4)$
Máximo e Mínimo de $H(k)$	$Max(1), Min(1)$	$Max(2), Min(2)$	$Max(3), Min(3)$	$Max(4), Min(4)$

Tabela geral FINAL para $k = 1, 2, 3, ...$das medidas de aleatoridade do algoritmo K128

$VetEntra =$	$Bl(1)$	$Bl(2)$	$Bl(3)$	$Bl(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetEntraC$ (criptografado)=	$BlC(1) = A$	$BlC(2) = B$	$BlC(3) = C$	$BlC(4) = D$
$VetAlter =$	$BlAlter(1) = Bl(1)$	$BlAlter(2)$	$BlAlter(3)$	$BlAlter(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetAlterC$ (criptografado)=	$BlAlterC(1) = A' = A$	$BlAlterC(2) = B'$	$BlAlterC(3) = C'$	$BlAlterC(4) = D'$
Posição do único bit alterado		$j = 65, ...128$		
Distância de Hamming	$H(1) = Ham(A, A')$**= 0**	$H(2) = Ham(B, B')$	$H(3) = Ham(C, C')$	$H(4) = Ham(D, D')$
Núm.de valores de $H(k)$	64	128	128	128
Soma acumulada de $H(k)$	$SomaH(1)$	$SomaH(2)$	$SomaH(3)$	$SomaH(4)$
Máximo e Mínimo de $H(k)$	$Max(1), Min(1)$	$Max(2), Min(2)$	$Max(3), Min(3)$	$Max(4), Min(4)$

Tabela, para $j = 65, ...128$, de valores para as medidas de aleatoridade do algoritmo K128

$VetEntra =$	$Bl(1)$	$Bl(2)$	$Bl(3)$	$Bl(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetEntraC$ (criptografado)=	$BlC(1) = A$	$BlC(2) = B$	$BlC(3) = C$	$BlC(4) = D$
$VetAlter =$	$BlAlter(1) = Bl(1)$	$BlAlter(2) = Bl(2)$	$BlAlter(3)$	$BlAlter(4)$
Aplica algoritmo em CBC	K128(K)	K128(K)	K128(K)	K128(K)
$VetAlterC$ (criptografado)=	$BlAlterC(1) = A' = A$	$BlAlterC(2) = B' = B$	$BlAlterC(3) = C'$	$BlAlterC(4) = D'$
Posição do único bit alterado			$j = 129, ...192$	
Distância de Hamming	$H(1) = Ham(A, A')$**= 0**	$H(2) = Ham(B, B')$**= 0**	$H(3) = Ham(C, C')$	$H(4) = Ham(D, D')$
Núm.de valores de $H(k)$	64	128	192	192
Soma acumulada de $H(k)$	$SomaH(1)$	$SomaH(2)$	$SomaH(3)$	$SomaH(4)$
Máximo e Mínimo de $H(k)$	$Max(1), Min(1)$	$Max(2), Min(2)$	$Max(3), Min(3)$	$Max(4), Min(4)$

Tabela, para $j = 129, ...192$, de valores para as medidas de aleatoridade do algoritmo K128

No final o programa deve imprimir uma tabela contendo os valores máximos, mínimos e médios das distâncias de Hamming entre **cada** bloco criptografado de 64 bits $BlC(k)$ e $BlAlterC(k)$, conforme o Algoritmo K128, no modo CBC. Para 8 blocos de 64 bits, o programa deve imprimir 8 valores máximos, 8 mínimos, e 8 médios.

O seu programa deve também efetuar os itens seguintes:

Item 1: Medir a aleatoriedade do K128 na forma descrita acima.

Item 2: Efetuar o Item 1 uma outra vez, mas 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 criptografada, devido a dois bytes consecutivos alterados na entrada.