

MAC 5723 - 336 - Criptografia
PRIMEIRO SEMESTRE DE 2007

Exercício-Programa

Data de entrega: até **22 de maio de 2007**.

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 (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 por cento 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.

Função K128

Implementar a função criptográfica K128, com chave de 128 bits, e com entrada e saída de 128 bits. Você deve **deduzir** o algoritmo inverso do K128. Na realidade a chave K de 128 bits serve de “semente” para derivar uma matriz pseudo-aleatória chamada SBoxes (como explicado abaixo), mas para este exercício teremos uma matriz fixa para padronizar os testes.

Entrada: 128 bits divididos em 4 blocos de 32 bits, E1,E2,D1,D2, nessa ordem,
8 chaves secundarias de 32 bits em ChavesSecund[]

e uma matriz SBoxes

Saida: 128 bits criptografados armazenados em E1,E2,D1,D2

```
E1,E2, D1,D2: int32; // cada variavel com 32 bits
Nrounds: integer; // valor "default" ee 16, mas pode ser 24 ou 32
//matriz SBoxes abaixo e' uma chave pre-gerada a partir de K;
// cada valor entre 0 e 65535 (i.e., 32 bits)
SBoxes: ARRAY [1 .. Nrounds/8] OF ARRAY [0 .. 255] OF int32;
// 8 chaves auxiliares abaixo sao pre-geradas a partir de K tambem;
// cada valor entre 0 e 65535 (32 bits)
ChavesSecund: ARRAY[1 .. 8] OF int32;
// Vetor abaixo e' o numero de posicoes de bits a serem rotacionados
PosicoesRotacao: ARRAY [1 .. 8] := [16,8,16,8,16,24,16,24];
IndLinhaNaSBoxes: integer; //indice de linha na matriz Sboxes

E1 := E1 XOR ChavesSecund[1]; // ou-exclusivo de 32 bits
E2 := E2 XOR ChavesSecund[2];
D1 := D1 XOR ChavesSecund[3];
D2 := D2 XOR ChavesSecund[4];
IndLinhaNaSBoxes := 1;
FOR round := 1 to Nrounds DO // Por definicao 'Nrounds' deve ser multiplo de 8
  Begin
    IndCol:= "8 bits mais 'a direita de E1"; // int entre 0 e 255
    D1 := D1 XOR SBoxes[IndLinhaNaSBoxes][IndCol];
    E1 := RotacioneParaDireita(E1, PosicoesRotacao[1+(round-1) mod 8] );
    IndCol:= "8 bits mais 'a direita de D2"; // corrigido, maio 2007
    E2 := E2 XOR SBoxes[IndLinhaNaSBoxes][IndCol];
    D2 := RotacioneParaDireita(D2, PosicoesRotacao[1+(round-1) mod 8] );
    Trocar(E1,E2);
    Trocar(D1,D2);
    if (round mod 8 = 0) then IndLinhaNaSBoxes := IndLinhaNaSBoxes + 1;
  End;
E1 := E1 XOR ChavesSecund[5];
E2 := E2 XOR ChavesSecund[6];
```

```
D1 := D1 XOR ChavesSecund[7];
D2 := D2 XOR ChavesSecund[8];
```

1. As ChavesSecund[] devem ser derivadas da chave K através de qualquer algoritmo pseudo-aleatório da sua escolha. Devem ser únicas para cada chave K.
2. Cada linha na matriz SBoxes é constituída de 256 elementos de 32 bits; cada elemento é constituído de 4 inteiros de 8 bits. Os 4 inteiros formam 4 colunas; cada uma dessas colunas é uma permutação dos inteiros de 0 a 255 (8 bits) calculada pelo algoritmo no Vol II do livro de Don Knuth, como veremos a seguir.
3. A chave K de 128 bits servirá de “semente ” para derivar as SBoxes, e deve ser única para cada chave K. Uma matriz fixa pre-gravada no **web-site da disciplina**, pode ser usada para testes. A versão final para entregar deverá derivar a SBoxInicial (veja a seguir) através de qualquer algoritmo determinístico da sua escolha.
4. Neste algoritmo no livro de D. Knuth, descrito a seguir, tem-se no início um vetor SBox inicial chamado SBoxInicial, de 256 inteiros de 32 bits; este vetor é interpretado como uma matriz de 4 colunas e 256 linhas de inteiros de 8 bits chamada Sbox (como foi feito antes com uma linha de SBoxes). Cada coluna de SBox contém uma permutação dos inteiros de 0 a 255
5. Os inteiros na permutação em Sbox são trocados entre si de forma pseudo-aleatória resultando uma outra permutação.

```
FOR IndLinhaNaSboxes:= 1 to (Nrounds/8) DO // para cada linha de 256 colunas
// A seguir, uma matriz SBoxInicial supostamente gerada por um algoritmo
// tendo K como "semente". Este algoritmo de geracao devera' ser projetada
// por voce, no caso de aplicacao real
SBox:= SBoxInicial; // Sbox e' matriz de 4 colunas e 256 linhas de ints de 8 bits
// cada coluna e' uma permutacao dos ints de 0 a 255
FOR IndCol := 0 to 3 DO // note 4 colunas de ints, cada coluna com 1 permutacao
BEGIN
FOR i := 0 to 255 DO // 256 linhas com uma permutacao de 8 bits
BEGIN
IndAleatorioLin := AleatorioEntre(i,255); // esta funcao retorna um inteiro
// entre i e 255, inclusive
TroqueInteiro(SBox[i,IndCol], SBox[IndAleatorioLin,IndCol]);
END; // for i
END; // for IndCol
SBoxes[IndLinhaNaSBoxes] := SBox; // define outra linha de SBoxes, 256 colunas
END; // for IndLinhaNaSBoxes
```

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) ficarão à escolha de cada aluno.

Geração da chave K de 128 bits: a senha A no parâmetro -p <senha> deve conter pelo menos **8** caracteres, sendo A com **pelo menos 2** letras e 2 algarismos decimais; se A 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).

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 A , no modo CBC (Cipher Block Chaining), com blocos de 128 bits.

Observação: Modo CBC consiste em encadear um bloco de 128 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 128 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 $Entra$.

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 pelo menos 512 bits (i.e., pelo menos 4 blocos de 128 bits, de modo que

$$VetEntra = BI(1)||BI(2)||BI(3)||BI(4)||\dots,$$

cada bloco $BI(i)$ de 128 bits e $|VetEntra| \geq 4 * 128 = 512$).

<i>VetEntra</i>	<i>Bl(1)</i>	<i>Bl(2)</i>	<i>Bl(3)</i>	<i>Bl(4)</i>	...
<i>VetEntraC</i> (criptografado)	<i>BIC(1)</i>	<i>BIC(2)</i>	<i>BIC(3)</i>	<i>BIC(4)</i>	...
					...
<i>VetAlter</i>	<i>BIAAlter(1)</i>	<i>BIAAlter(2)</i>	<i>BIAAlter(3)</i>	<i>BIAAlter(4)</i>	...
<i>VetAlterC</i> (criptografado)	<i>BIAAlterC(1)</i>	<i>BIAAlterC(2)</i>	<i>BIAAlterC(3)</i>	<i>BIAAlterC(4)</i>	...
	$j = 1, 2, \dots, 128$	$j = 1, 2, \dots, 256$	$j = 1, 2, \dots, 384$	$j = 1, 2, \dots, 512$...
Distância de Hamming	<i>H(1)</i>	<i>H(2)</i>	<i>H(3)</i>	<i>H(4)</i>	...
Soma acumulada de <i>H(k)</i>	<i>SomaH(1)</i>	<i>SomaH(2)</i>	<i>SomaH(3)</i>	<i>SomaH(4)</i>	...

Para $j = 1, 2, \dots, |VetEntra|$ 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 $j = 1, 2, 3, \dots$ 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, $j = 1, 2, 3, \dots, 512$. Por exemplo, no caso de $j = 2$, $Bl(1) = 01010101$, $Bl(2) = 00110101, \dots$ e

$$VetAlter = BIAAlter(1) || BIAAlter(2) || \dots = 00010101 || 00110101 || \dots$$

ou seja diferem só no bit na posição 2.

2. seja $VetEntraC = BIC(1) || BIC(2) || BIC(3) || BIC(4) || \dots$ o vetor *VetEntra* criptografado pelo K128-CBC. E seja

$$VetAlterC = BIAAlterC(1) || BIAAlterC(2) || BIAAlterC(3) || BIAAlterC(4) || \dots$$

o vetor *VetAlter* criptografado pelo K128-CBC.

3. medir a distância de Hamming, **separadamente**, entre **cada** bloco $BIC(k)$ de 128 bits de *VetEntraC* e o correspondente bloco $BIAAlterC(k)$ de 128 bits de *VetAlterC*. Para 4 blocos de 128 bits, tem-se 4 medidas de distância, sendo cada medida chamada, digamos, $H(k)$ para cada par de blocos $BIC(k), BIAAlterC(k)$. Ou seja, para $k = 1, 2, 3, 4$, $H(k) = \text{Hamming}(BIC(k), BIAAlterC(k))$.
4. estas medidas de distância de Hamming $H(k)$ devem ser acumuladas em somas chamadas, digamos, $SomaH(k)$. Para 4 blocos de 128 bits, tem-se 4 somas cumulativas, sendo que:
 - a. $SomaH(1)$ acumula 128 valores de $H(1)$ correspondentes a $j = 1, 2, 3, \dots, 128$ (para $j > 128$ $H(1) = 0$ pois $BIC(1) = BIAAlterC(1)$)
 - b. $SomaH(2)$ acumula $2 * 128 = 256$ valores de $H(2)$ correspondentes a $j = 1, 2, 3, \dots, 128, 129, \dots, 256$ (para $j > 2 * 128$ $H(2) = 0$ pois $BIC(2) = BIAAlterC(2)$ e $H(1) = 0$ pois $BIC(1) = BIAAlterC(1)$)
 - c. $SomaH(3)$ acumula $3 * 128 = 384$ valores de $H(3)$ correspondentes a $j = 1, 2, 3, \dots, 384$
 - d. $SomaH(4)$, acumula $4 * 128 = 512$ valores de $H(4)$ correspondentes a $j = 1, 2, 3, \dots, 512$.

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

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 $BIC(k)$ e $BIAIterC(k)$, conforme o Algoritmo K128, no modo CBC. Para 4 blocos de 128 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.

FIM