Prática 2 - Criptografia e Confidencialidade

Parte I - Data Encryption Standard (DES)

1. Cifre algo com `DES` e observe a saída; mude 1 bit apenas na mensagem de entrada e cifre novamente usando a mesma chave.

- Quantos bytes foram alterados na saída em relação à anterior?
- Pode-se tirar alguma conclusão sobre o que foi observado?

Como podemos observar nas saídas abaixo, a mudança de 1 bit na mensagem fez com que todos os bytes de saída fossem alterados.

Esse efeito é desejado. Caso ele não esteja presente a cifra fica vulnerável a ataques que alteram um pouco a entrada e observam uma pequena mudança na saída. No `DES` isso ocorre devido a cada uma das 16 rodadas utilizarem resultados de rodadas anteriores, assim o efeito se acumula. Esse efeito também é chamado de efeito avalanche.

Saída:

```
Message: 0123456789abcdef
Output: aa39b9777efc3c14

Message: 1123456789abcdef
Output: 5581526b65c70bf6
```

2. Observe a evolução do algoritmo passo-a-passo no quadro "Details".

- Quantas execuções da Função de Feistel (f) são efetuadas?
- Quantas execuções de f ocorrem quando se escolhe `3DES`?

A função de Feistel é executada 16 vezes com `DES` e 48 vezes com `3DES`.

3. Qual é o tamanho da chave em bits, nessa implementação do `3DES`? Observe que a chave não tem o mesmo tamanho da especificação vista em sala de aula; qual é mais segura e por quê?

No `3DES` a mensagem cifrada é dada por:

$$\text{cifrada} = C(D(C(m, k_1), k_2), k_3)$$

Onde $C(m, k)$ e $D(m, k)$ são os processo de cifrar e decifrar, respectivamente, a mensagem m com a chave k.

A implementação do `3DES` usa como chaves $k_1 = k_3 = 3b3898371520f75e$ e $k_2 = 922fb510c71f436e$. Como temos duas chaves iguais, temos efetivamente uma chave de 112 bits.

A especificação vista em sala do `3DES` utiliza $k_1 \neq k_2 \neq k_3$. Assim temos uma chave de 168 bits, logo essa versão é mais segura.

4. Veja que nesse mesmo link, logo abaixo do quadro "Details", há um resumo da descrição do `DES` (How DES Works). Em linhas gerais, em que partes do algoritmo são aplicados os conceitos de Confusão e de Difusão, para aumento da entropia do texto cifrado?

A confusão aumenta a entropia a partir da falta de relação entre a `mensagem`, a `chave` e a `mensagem cifrada` através de *substuições polialfabéticas*. No `DES` podemos ver esse conceito, por exemplo, na Função de Feistel e nas iterações $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$

A difusão aumenta a entropia a partir da dispersão de informações da `mensagem` e da `chave` pela `mensagem cifrada` através de *permutações* e *transposições*. No `DES` podemos ver, por exemplo, na permutação inicial e final, na Função de Feistel, na geração de subchaves e nas trocas de $Li = R_{i-1}$.

5. Esse simulador do DES foi escrito em Javascript; exiba o código no próprio navegador. Em que vetor (variável do tipo array) são guardadas as diferentes chaves usadas em cada rodada da Função de Feistel (f)? Quantas chaves derivadas são geradas para a execução do DES simples?

As chaves usadas em cada rodada da Função de Feistel estão no vetor `KS[i]` da função `des_encrypt`. São utilizadas 16 chaves para a execução do `DES` simples.

6. Para observar o comportamento do `DES` em uma única iteração, insira uma sequência de 8 zeros (em ASCII "0") como mensagem de entrada. Cifre.

- Em "Details", imediatamente antes de iniciar o *Round 1*, `L[0]` e `R[0]` guardam os 8 zeros da entrada, com uma permutação inicial.Percentualmente, quantos bits em `L[0]` e `R[0]` são iguais a 0 e 1?
- Imediatamente antes de iniciar o *Round 2*, `L[1]` e `R[1]` guardam a mensagem original processada apenas uma vez com Feistel. Qual é o percentual de bits 0 e 1 nesse instante?
- Uma cifra forte, após cada iteração, gera um estado intermediário em que cerca de 50% dos bits são iguais a zero, independentemente da entrada e da chave. Isso ocorre com o `DES`? Que conclusão você tira a respeito disso?

Antes de iniciar o *Round 1* temos:

```
L[0]:  00000000 11111111 00000000 00000000
R[0]:  00000000 11111111 00000000 00000000
```

Logo, juntando `L[0]` e `R[0]`, temos que 75% dos bits são iguais a 0 e 25% são iguais a 1.

Antes de iniciar o *Round 2* temos:

```
L[i]: 00000000 11111111 00000000 00000000
R[i]: 11010110 01011110 01000110 00101110
```

Logo, juntando `L[1]` e `R[1]`, temos 61% dos bits são 0 e 39% são 1.

Como o estado intermediário não está próximo da distribuição de bits que esperamos de uma cifra forte, podemos concluir que o `DES` não é uma cifra forte.

Parte 2 - Advanced Encryption Standard (AES)

1. Faça alguns testes para cifrar e decifrar com o `AES`. Veja o pseudo-código nesta mesma página (textos *Encryption Algorithm* e *AES Decryption*). Observe que para cifrar há um laço de repetição com 10 passos, enquanto para decifrar, o laço é de 9 passos. Em sua opinião, estão corretos esses pseudo-códigos? Justifique.

Na última iteração da cifra não há `MixColumns`, logo não devemos inverter essa operação ao decifrar. Então o último conjunto de operações difere do que está dentro do laço de decifrar.

2. Qual é a função do vetor `w` nessa implementação?

O vetor `w` é utilizado para guardar as informações da chave expandida.

Parte 3 - Modos de Operação

1. Insira duas diferentes mensagens de 16 caracteres (128 bits) em *Message Part 1* e *Message Part 2*. Observe os textos cifrados nos modos *Electronic Codebook* (`ECB`) e *Cipher Block Chaining* (`CBC`). Os dois modos de operação produziram alguma semelhança na saída? Explique o motivo.

Utilizando

- `Message Part 1 = aca2c616b84bf1b141c3b0c5d1b8e51b`
- `Message Part 2 = d9cdf1490ec7cffe8d1c593151e6b518`

Temos como resultados:

Output `Eletronic Codebook`:

```
Output Part 1: 7a 26 00 9b 1a 17 1f cf 97 d5 6c ce 6b 1f ae 22
Output Part 2: 81 0e f4 92 5d 23 b5 42 52 61 67 c5 69 d4 70 cb
```

Output `Cipher Block Chaining`:

```
Output Part 1: f8 8f 6b 91 65 ba da b9 2d ac 77 9e 2e 7b 89 14
Output Part 2: 4b bd b4 c9 3f 94 26 bf ca 07 6b 69 d4 15 93 79
```

Os dois modos produziram saídas totalmente diferentes. Isso ocorre pois o `CBC` realiza `XOR`, de um vetor de inicialização (no primeiro bloco) ou a cifra do bloco anterior, com o bloco de mensagem. Já o `ECB` não faz esse procedimento, cifrando de maneira independentemente os blocos.

2. Repita o teste anterior para mensagens iguais (*Message Part 1 = Message Part 2*), para os mesmos dois modos de operação. O que é observado e o que se pode concluir?

Utilizando

- `Message Part 1 = aca2c616b84bf1b141c3b0c5d1b8e51b`
- `Message Part 2 = aca2c616b84bf1b141c3b0c5d1b8e51b`

Temos como resultados:

Output `Eletronic Codebook`:

```
Output Part 1: 7a 26 00 9b 1a 17 1f cf 97 d5 6c ce 6b 1f ae 22
Output Part 2: 7a 26 00 9b 1a 17 1f cf 97 d5 6c ce 6b 1f ae 22
```

Output `Cipher Block Chaining`:

```
Output Part 1: f8 8f 6b 91 65 ba da b9 2d ac 77 9e 2e 7b 89 14
Output Part 2: e9 f5 95 9c 5d ac 32 9a 5f bf 7f da f0 18 d7 7a
```

As saídas do `ECB` são iguais. Isso ocorre pois ele cifra os blocos de maneira independente, dessa maneira blocos de mensagem iguais levam a mesma cifra.

Já as saídas do `CBC` são diferentes devido ao explicado na questão 1.

3. A implementação dos modos `ECB` e `CBC` nesse Javascript não faz *Padding*, o que é necessário se fazer numa aplicação real. O que é *Padding* e como se pode perceber que esse Javascript não o faz?

Padding é a técnica de aumentar a mensagem para que seu comprimento seja um múltiplo do tamanho do bloco.

Podemos perceber que essa implementação não possuí *padding* ao tentar colocar, por exemplo, uma mensagem de 33 dígitos hex.

4. Teste o modo de operação *Output Feedback* para mensagens de tamanhos menores que 16 caracteres.

- Por que funciona e o que se pode concluir? (veja a descrição dos modos de operação nessa mesma página).
- O `AES` com o modo *Output Feedback* poderia substituir uma cifra de fluxo?
- Para que tipo de aplicação seria mais vantajoso usar cifra de fluxo e porquê?
- Para que tipo de aplicação seria mais vantajoso usar `AES` com *Output Feedback* e porquê?

O `Output Feedback` (`OP`) funciona com uma mensagem menor pois ele utiliza um vetor de inicialização (`IV`) para iniciar a cifra. Então, a cifra da mensagem é resultado de um `XOR` da mensagem com a cifra do `IV`, logo não é necessário que a mensagem possua tamanho exato. Esse resultado pode ser cifrado e `XOR` com outra mensagem, e assim podemos transformar a cifra de bloco em uma cifra de fluxo.

Seria mais interessante utilizar cifras de fluxo em aplicações em que o tamanho da saída deve ser o menor possível. Pois o resultado de uma cifra de fluxo tem o mesmo tamanho que a entrada, enquanto o resultado do `OP` sempre tem o tamanho do bloco.

O resultado do `OP` também depende de execuções anteriores, logo ele pode propagar erros.

5. Localize nessa página observações a respeito do vetor de inicialização IV. Quais são as considerações de segurança feitas e suas justificativas?

O `IV` deve ser único para cada mensagem. Pois duas mensagens cifradas com o mesmo `IV` podem ser atacadas utilizando as mensagens e suas cifras.

No `OP`, por exemplo, se utilizarmos o mesmo `IV` para duas mensagens o resultado da cifra é apenas o `XOR` da cifra do `IV` com as mensagens. Basta notar que a cifra do `IV` é igual nos dois casos. Assim podemos apenas manipular as cifras, ignorando a função de cifra, para determinar as mensagens.