



**Universidade Federal do Rio de Janeiro**  
Instituto de Matemática  
Departamento de Ciência da Computação  
Grupo de Resposta a Incidentes de Segurança

---

Rio de Janeiro, RJ – Brasil

## **Fundamentos da Criptologia**

### **Parte III – Criptografia Simétrica (Continuação)**

GRIS-2005-A-005

Breno Guimarães de Oliveira

A versão mais recente deste documento pode ser obtida na página oficial do GRIS

GRIS – Grupo de Resposta a Incidentes de Segurança  
CCMN Bloco I 1º andar  
Sala: I1021  
Av. Brigadeiro Trompowski, s/nº  
Cidade Universitária - Rio de Janeiro/RJ  
CEP: 21949-900  
Telefone: +55 (21) 2598-3309

Este documento é Copyright© 2005 GRIS. Ele pode ser livremente copiado desde que sejam respeitadas as seguintes condições:

É permitido fazer e distribuir cópias inalteradas deste documento, completo ou em partes, contanto que esta nota de copyright e distribuição seja mantida em todas as cópias, e que a distribuição não tenha fins comerciais. Se este documento for distribuído apenas em partes, instruções de como obtê-lo por completo devem ser incluídas. É vedada a distribuição de versões modificadas deste documento, bem como a comercialização de cópias, sem a permissão expressa do GRIS.

Embora todos os cuidados tenham sido tomados na preparação deste documento, o GRIS não garante a correção absoluta das informações nele contidas, nem se responsabiliza por eventuais consequências que possam advir do seu uso.

## **Sumário:**

1. Cifras Computacionais.....	3
2. XOR.....	3
3. DES.....	3
4. Referências Bibliográficas.....	10
Apêndice: Funções de Seleção do DES.....	11

## 1. Cifras Computacionais

A possibilidade de automatização de cálculos complexos através de computadores permitiu enormes avanços na criptologia, que não eram possíveis quando codificação e decodificação precisavam ser feitos a mão. Agora, chaves e funções muito mais complexas podem ser utilizadas com pouco ou nenhum esforço de ambos remetente e destinatário, já que a máquina faz todo o “trabalho sujo”. Antes de começarmos, vale a pena discutir brevemente a terminologia utilizada. Quando falamos em *bits*, estamos nos referindo a dígitos binários, isto é, um conjunto de zeros e uns. Em computação, os dados são todos armazenados em *bits*, e separados em blocos chamados *bytes*. Geralmente, cada byte possui 8 bits, e é esse valor que será considerado aqui. Além disso, como os valores são representados na base 2 (onde cada casa possui apenas os valores 0 ou 1) e não na base 10 a que estamos acostumados (onde cada casa possui valores de 0 a 9), vale lembrar que a progressão dos números não é mais 0, 1, 2, 3, 4, 5, 6, ... e sim 0, 1, 10, 11, 100, 101, 110, ...

## 2. XOR

<i>p</i>	<i>q</i>	<i>p XOR q</i>
0	0	0
0	1	1
1	0	1
1	1	0

**Definição:** A codificação por XOR – o “ou exclusivo” – é bem mais complexa que a ROT-X, e exige uma abstração um pouco maior. Aqui o trabalho é todo feito com valores binários da mensagem e da chave escolhida. Para os que não sabem ou não se lembram, o “ou exclusivo” é uma operação lógica entre duas variáveis, cada qual podendo assumir o valor de *verdadeiro* ou *falso* (representados na computação como 1 e 0, respectivamente). O XOR retorna verdadeiro somente quando os valores das variáveis são diferentes entre si, isto é, quando ou uma é verdadeira ou a

outra, mas nunca as duas ao mesmo tempo (daí seu nome). Acontece que, num computador, letras e números também são convertidos em valores binários (conjuntos de zeros e uns) equivalentes, de acordo com tabelas arbitrárias de conversão. A mais popular delas é a tabela ASCII – *American Standard Code for Information Interchange* – padrão na maioria dos computadores pessoais. Ela utiliza uma sequência de oito bits para representar caracteres e símbolos especiais, e os valores para as letras de nosso alfabeto são os seguintes:

A	B	C	D	E	.....	Z
01000001	01000010	01000011	01000100	01000101	.....	01011010
a	b	c	d	e	.....	z
01100001	01100010	01100011	01100100	01100101	.....	01111010

Caracteres acentuados e pontuações também possuem códigos equivalentes, como 00100001 para o símbolo de exclamação (!) e 10000010 para o “e” com acento agudo (é). Para a codificação XOR, o primeiro passo é converter nossa mensagem para código binário. Precisamos agora de uma chave, que pode ser qualquer sequência de caracteres, de preferência grande e com poucas repetições. O que fazemos é emparelhar os bits da mensagem com os da chave escolhida, e fazer uma operação de XOR entre os dois. Como a mensagem provavelmente será maior que a chave, esta é simplesmente repetida até atingir seu comprimento ideal. Para o exemplo a seguir, usamos a palavra “criptograma” como chave:

Chave:

"criptograma"

Chave transformada em código ASCII binário:

"0110001101110010011010010111000001110100011011110110011101110010011000010110110101100001"

Mensagem original:

"O pacote 22 foi entregue. Aguardo instruções."

Mensagem transformada em código ASCII binário:

"01001111001000000111000001100001011000110110111101110100011001010010000000110010001100100010000001100110  
011011110110100100100000011001010110111001110100011100100110010101100111011101010110010100101110001000000  
100000101100111011101010110000101110010011001000110111100100000011010010110111001110011011101000111001001  
1101011000011111100100011001010111001100101110"

Chave estendida até o comprimento da mensagem (mesma chave, repetida um pouco mais de quatro vezes):

"01100011011100100110100101110000011101000110111101100111011100100110000101101101011000010110001101110010  
011010010111000001110100011011110110011101110010011000010110110101100001011000110111001001101001011100000  
11101000110111101100111011100100110000101101101011000010110001101110010011010010111000001110100011011101  
1001110111001001100001011011010110000101100011"

Operação de XOR entre a mensagem e chave:

	0100111100100000011100000110000101100011...	(mensagem original)
XOR	<u>0110001101110010011010010111000001110100...</u>	(chave)
	00101100010100100001110010001000100010111...	(criptograma)

Criptograma final:

"001011000101001000011001000100010001011100000000001001100010111010000010101111010100110100001100010100  
000001100001100101010100000010100000100100000110000100110000100000000110000101100001011101000111010100000  
011010100001000000100100001001100010011000010010000111001000011000110110000011100000001100000000001110100  
0100101111010110000101000010000001001001001101"

O criptograma resultante nada mais é do que uma marcação de todos os bits diferentes entre a mensagem original e a chave escolhida (repetida tantas vezes quanto o necessário para a mensagem em questão). Analogamente, para decodificar, basta verificar os bits diferentes entre a chave e o criptograma, fazendo uma a operação de XOR entre eles:

Operação de XOR entre criptograma e chave:

	0010110001010010000110010001000100010111...	(criptograma)
XOR	<u>0110001101110010011010010111000001110100...</u>	(chave)
	0100111100100000011100000110000101100011...	(mensagem original)

**Observação:** Embora a mensagem original possa ser facilmente convertida da representação binária para a textual, o mesmo não acontece com o criptograma gerado. Isso porque o valor gerado para ele, quando quebrado em blocos de oito bits, não necessariamente gera um caractere pronunciável na tabela ASCII. Embora isso não apresente empecilho algum numa transmissão de dados, que é feita em modo binário de qualquer forma, às vezes é interessante exibir o criptograma visualmente. Como pode ser facilmente observado pelo exemplo acima, colocar o código binário de um criptograma em forma de texto toma muito espaço – mais precisamente oito dígitos para cada caractere – então, para contornar esse problema, podemos usar a notação hexadecimal<sup>1</sup> quando desejarmos representar o mesmo. Assim, o criptograma final do exemplo acima assumiria a seguinte forma:

0x2C52191117001317415F5343140619540A0906130806161747503508121313090E431B0703001D12F58508124D

**Código-fonte:** O código a seguir realiza uma operação de XOR entre duas seqüências de caracteres intercaladas, e retorna o código binário da mesma. Entrando com mensagem e chave, você obtém o criptograma. Entrando com criptograma e chave, você obtém a mensagem original. Seguindo o exemplo anterior, se quiséssemos codificar a frase “*O pacote 22 foi entregue. Aguardo instruções.*” usando a chave “criptograma” (no caso, “criptogramacriptogramacriptogramacriptogramac”), a entrada que usaríamos para esse programa seria “cOr ippatcoogtree m2a2c rfiopit oegnrarmeagcurei.p tAogguraarmdaoc riinpsttorgurçaõmeasc.”, que são as duas frases intercaladas entre si.

```
----- xor.c (início) -----
#include <stdio.h>

void main(void)
{
    int entrada, chave=0, mensagem=0, saida, i;

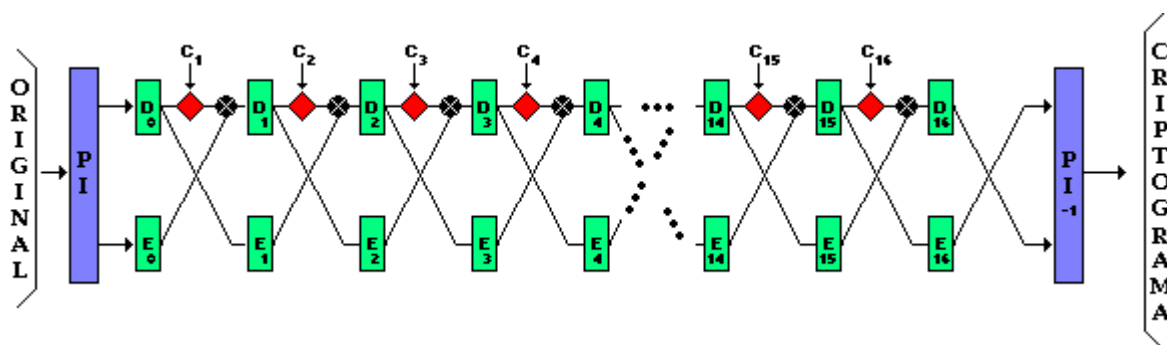
    printf("digite a mensagem e a chave intercaladas, caractere por caractere:\n");
    while(entrada=getchar())
    {
        if(entrada==10)
            printf("\n");
        else
        {
            if(chave==0)
                chave = entrada;
            else
            {
                mensagem = entrada;
                saida = chave ^ mensagem;
                for(i=0; i<8; i++)
                    printf("%d", (saida & (int)(1 << (7-i))) >> (7-i));
                mensagem = chave = 0;
            }
        }
    }
}
----- xor.c (fim) -----
```

---

<sup>1</sup> A notação hexadecimal utiliza a base 16, onde cada casa assume um valor entre 0 e 16. Esses valores são representados na casa pelos símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F, nessa ordem. Quando expressamos um valor em notação hexadecimal, costuma-se colocar um “0x” na frente do número.

### 3. DES

**Definição:** O Data Encryption Standard (DES) foi a cifra padrão oficial dos Estados Unidos entre 1976 e 2001, quando foi substituído pelo AES. Ao contrário das técnicas vistas até agora, o DES utiliza um método conhecido como *cifra de bloco*, em que a mensagem original é separada em pedaços de tamanho fixo – chamados “blocos” – antes de ser codificada, e o criptograma gerado tem exatamente o mesmo tamanho do bloco (no caso do DES, 64 bits). Uma chave de 64 bits é utilizada para a codificação do DES, dos quais 56 bits são utilizados no algoritmo e os 8 restantes servem apenas para verificação de erros por paridade<sup>2</sup>. A mensagem original em formato binário é separada em blocos de 64 bits, e cada bloco passa pelo método de codificação mostrado na figura abaixo:



Primeiramente, os bits do bloco passam por uma permutação inicial **PI**. Considerando a ordenação dos bits da mensagem como sendo da esquerda para a direita, isto é, o bit mais a esquerda possuindo índice 1, e o mais a direita possuindo índice 64, PI coloca os bits da mensagem na seguinte ordem:

```
58-50-42-34-26-18-10-02-60-52-44-36-28-20-12-04-62-54-46-38-30-
22-14-06-64-56-48-40-32-24-16-08-57-49-41-33-25-17-09-01-59-51-
43-35-27-19-11-03-61-53-45-37-29-21-13-05-63-55-47-39-31-23-15-07
```

Ou seja, após esse processo, o primeiro bit da mensagem permutada é o bit 58 da mensagem original, o segundo é o de índice 50 e assim por diante, até o sétimo bit da mensagem original, que é o último da mensagem permutada.

**Código-fonte:** A função `permuta_PI()` abaixo realiza a permutação PI dos elementos de um bloco qualquer e a armazena em um bloco de destino. Foi adicionada a função principal `main()` que chama a anterior com um bloco de exemplo e exibe a permutação na tela.

<sup>2</sup> A verificação de erros por paridade é uma prática comum em circuitos computacionais para detectar erros na transmissão de bits. No caso do DES, cada bit dos 8 restantes são definidos como 0 ou 1 para manter a paridade de cada byte da chave ímpar, ou seja, com um número ímpar de “1”s em cada byte. Assim, ao receber uma nova chave, o computador pode verificar se cada byte desta possui um número ímpar de “1”s e, caso contrário, houve algum tipo de erro na transmissão e o processo é abortado.

```

----- DES.c (início) -----
#include <stdio.h>

int PI[64] = {58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30,
             22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51,
             43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7};

void permuta_PI(int bloco_origem[64], int bloco_destino[64])
{
    int i;
    for(i = 0; i < 64; i++)
        bloco_destino[i] = bloco_origem[PI[i] - 1];
}

void main(void)
{
    int i;
    /** bloco exemplo de entrada (codigo binario do bloco) **/
    int bloco1[64] = {1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                     0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
                     0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1};

    int bloco2[64]; /** bloco para receber a permutacao do bloco anterior **/

    permuta_PI(bloco1, bloco2); /* chama a funcao de permutacao */

    /* exibe o resultado */
    printf("\no bloco\n");
    for(i = 0; i < 64; i++)
        printf("%d-", bloco1[i]);
    printf("\n\npermutado fica:\n");
    for(i = 0; i < 64; i++)
        printf("%d-", bloco2[i]);
}
----- DES.c (fim) -----

```

Após a permutação inicial, a mensagem passa por um processo conhecido como *rede de Feistel*, idealizada pelo criptólogo Horst Feistel, em que a mensagem é dividida em duas partes processadas alternadamente por uma operação de XOR e uma função arbitrária. No desenho acima, o XOR e a função estão representados pelo círculo cortado em X e o losango vermelho, respectivamente. No DES, a mensagem permutada é dividida em duas partes, **E** e **D**, que contém os 32 bits da esquerda e os 32 bits da direita da mensagem, respectivamente. Assim, para um par  $E_n D_n$  em um momento  $n$  qualquer do processamento, temos que:

$$\begin{aligned}
 E_{n+1} &= D_n \\
 D_{n+1} &= E_n \otimes f(D_n, C_{n+1})
 \end{aligned}$$

Isso significa apenas que, após a iteração, o novo  $E$  conterá o antigo valor de  $D$ , e o novo  $D$  conterá o resultado da operação XOR entre o antigo valor de  $E$  e o resultado da função de codificação. No DES, esse passo é repetido 16 vezes, conforme o esquema mostrado anteriormente. Após a última iteração, o par  $E D$  volta a compor um único bloco de 64 bits, só que invertidos (os 32 bits de  $D$  são colocados ANTES dos 32 bits de  $E$ , como mostrado na figura anterior). Finalmente, esse bloco passa pela permutação final  $\mathbf{PI}^{-1}$ , que é a permutação inversa de  $\mathbf{PI}$ , colocando os bits na seguinte ordem:

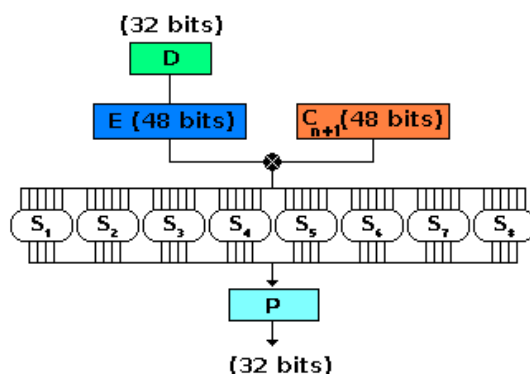
40-08-48-16-56-24-64-32-39-07-47-15-55-23-63-31-38-06-46-14-54-  
22-62-30-37-05-45-13-53-21-61-29-36-04-44-12-52-20-60-28-35-03-  
43-11-51-19-59-27-34-02-42-10-50-18-58-26-33-01-41-09-49-17-57-25

Após essa última permutação, o resultado é a mensagem codificada.

Vejamos agora como funciona a função  $f$  de codificação, ilustrada na figura abaixo. Primeiro ela transforma, através de uma permutação de expansão, os 32 bits de  $D$  em 48 bits. Isso é feito duplicando alguns bits de  $D$  e colocando-os na seguinte disposição:

32-01-02-03-04-05-04-05-06-07-08-09-08-09-10-11-  
12-13-12-13-14-15-16-17-16-17-18-19-20-21-20-21-  
22-23-24-25-24-25-26-27-28-29-28-29-30-31-32-01

Ou seja, o primeiro bit é o de índice 32 de  $D$ , seguido pelo primeiro e segundo bits de  $D$ , e terminando com os bits de índices 32 e 1 novamente. Fazemos então uma operação de XOR entre esse resultado e 48 bits arbitrários da chave (o modo como os bits da chave são escolhidos será descrito mais tarde). Para que os 48 bits resultantes dessa operação voltem a forma de 32 bits, eles são divididos em 8 pedaços sequenciais de 6 bits cada, que passarão por 8 funções de seleção  $S$  – uma para cada pedaço. Cada função de seleção receberá os 6 bits designados e os transformará em 4, através das tabelas descritas no apêndice.



Para exemplificar essa transformação, vejamos como a função de seleção  $S_3$  transforma os bits 110110 em 4 bits.

Tabela da função de seleção  $S_3$ :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

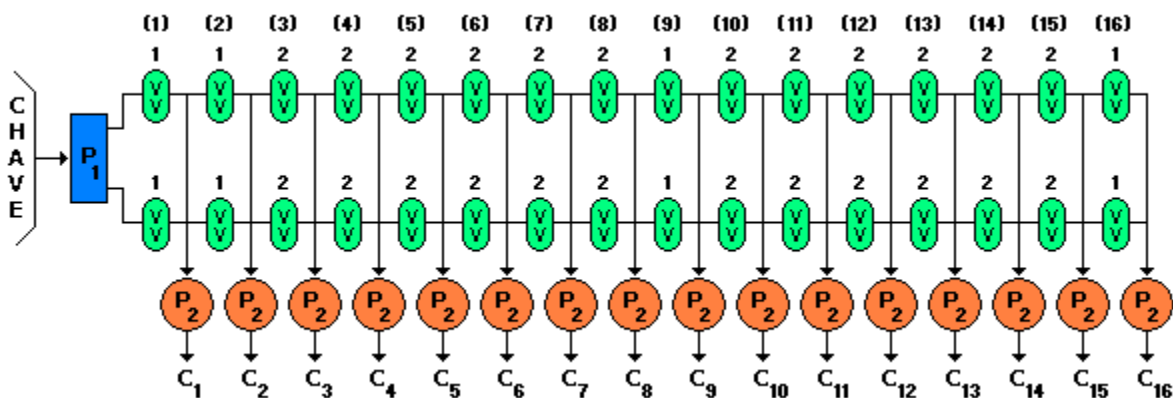
Os primeiro e último dígitos do trecho formam um número entre 0 e 3 em código binário (00 = 0, 01=1, 10=2, 11=3), que indicará o índice da linha. Em nosso exemplo, **110110** → 10 → 2. Os 4 dígitos que sobraram formam um número entre 0 e 15 em código binário, que indicará o índice da coluna. Em nosso exemplo, **110110** → 1011 → 11 (onze). Pegamos então o número na linha e coluna encontradas, e a representação desse número em 4 bits será o resultado. No exemplo, o número na linha 2 e coluna 11 é o 12, representado em quatro bits como 1100.



Após a redução completa, a função termina passando os 32 bits por uma última permutação, dessa vez na forma:

16-07-20-21-29-12-28-17-01-15-23-26-05-18-31-10-  
02-08-24-14-32-27-03-09-19-13-30-06-22-11-04-25


Agora resta apenas saber como as 16 subchaves de 48 bits são escolhidas a partir da chave de 64 bits, dos quais 8 servem apenas para verificação de erro (o que nos deixa com uma chave com 56 bits de tamanho útil).



A figura acima ilustra a escolha de cada subchave no DES. Primeiro, a chave passa por uma permutação  $P_1$  da seguinte forma:

57-49-41-33-25-17-09-01-58-50-42-34-26-18-  
10-02-59-51-43-35-27-19-11-03-60-52-44-36-  
63-55-47-39-31-23-15-07-62-54-46-38-30-22-  
14-06-61-53-45-37-29-21-13-05-28-20-12-04

Note que os bits da chave com índices 08, 16, 24, 32, 40, 48, 56 e 64 não estão incluídos na permutação, pois são os bits de verificação de erro.

O resultado é então dividido em duas partes iguais de 28 bits, que vão compor as 16 subchaves de 46 bits. Para a criação de cada subchave  $C_n$ , esses dois blocos passam por deslocamentos individuais a esquerda (denotados na figura pelo símbolo ) . Isso quer dizer que todos os bits do bloco vão “pular” um determinado número de casas para a esquerda. Conforme indicado na figura acima, todos os deslocamentos são de duas casas, exceto os de número 1, 2, 9 e 16, que são de apenas uma casa. Essa rotação é circular, então os bits deslocados para casas inexistentes à esquerda voltam para a extremidade direita (ou seja, após uma rotação, os bits 1, 2, 3, ..., 27, 28 ficam na ordem 2, 3, ..., 27, 28, 1). Finalmente, as duas partes permutadas e rotacionadas se juntam como uma única sequência de bits entre 1 e 48 e sofrem uma última permutação  $P_2$ , para se tornarem a subchave. Essa permutação é na forma:

14-17-11-24-01-05-03-28-15-06-21-10-23-19-12-04-  
26-08-16-07-27-20-13-02-41-52-31-37-47-55-30-40-  
51-45-33-48-44-49-39-56-34-53-46-42-50-36-29-32

Assim, o primeiro bit de  $C_n$  é o 14º bit da junção dos bits do lado esquerdo com os do lado direito, até o último bit da subchave, que é o 32º bit.

A vantagem de métodos criptográficos com rede de Feistel é que para decodificar a mensagem, basta aplicar o mesmo algoritmo a um bloco cifrado, e o resultado será o bloco original, desde que se tenha o cuidado de utilizar os mesmos pedaços da chave que foram utilizados para cada passo. Em outras palavras, se os pedaços da chave aplicados para codificar o bloco foram  $C_1, C_2, C_3, \dots, C_{15}, C_{16}$ , para decodificar o bloco elas deverão ser colocadas na ordem  $C_{16}, C_{15}, C_{14}, \dots, C_2, C_1$ , revertendo o processo.

**Variações:** Durante os 25 anos em que permaneceu no auge, o DES teve algumas variações de sucesso. Entre elas, o DES-X e o 3DES (ou TDEA) foram as que mais se destacaram. O DES-X foi proposto por Ronald Rivest em 1984, e funciona exatamente como o DES simples, mas adicionando duas novas chaves de 64 bits cada, uma aplicada ao bloco da mensagem em texto puro (antes da permutação inicial) através de um XOR e outra aplicada ao bloco final da mensagem, após a inversão da permutação inicial. Essas adições dificultam a quebra do algoritmo por força bruta. Já o 3DES (“Triplo DES” ou “Três DES”), conhecido oficialmente como TDEA (*Triple Data Encryption Algorithm*) foi concebido por Walter Tuchman e consiste na repetição do DES três vezes: a mensagem original é codificada com a chave 1, decodificada com a chave 2 e novamente codificada com a chave 3. Note que, se as chaves 1 e 2 forem diferentes, a “decodificação” do passo 2 não devolve a mensagem original, uma vez que são chaves diferentes. De fato, como visto anteriormente, a única diferença entre a codificação e decodificação no DES é a ordem das subchaves. Três variações são admitidas no 3DES: usar 3 chaves diferentes; duas chaves diferentes (nesse caso, a primeira e a última devem ser iguais, já que se a intermediária for igual a outra, ela reverterá o processo); ou usando a mesma chave, o que reduz o 3DES a um DES simples (a mensagem é codificada, decodificada corretamente, e então novamente codificada).

#### 4. Referências Bibliográficas:

Data Encryption Standard – NIST. <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>  
Wikipedia – <http://www.wikipedia.org>

## Apêndice: Funções de seleção do DES

$S_1$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S_2$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

$S_3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$S_4$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

$S_5$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

$S_6$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

$S_7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

$S_8$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11