Criptografia - DES/AES

Filipe Cabrita nº41892 Gonçalo Araújo nª41914

Abril 2017

Conteúdo

1 Conceitos Base **2**
- 1.1 Segurança de uma cifra . . . 2
  - 1.1.1 Confusão . . . 2
  - 1.1.2 Difusão . . . 2
- 1.2 Cifra de chave simétrica . . . 3

2 DES - Data Encryption Standard **4**
- 2.1 Introdução histórica . . . 4
- 2.2 Uma visão geral . . . 4
- 2.3 Encriptação . . . 6
  - 2.3.1 A permutação inicial e final . . . 8
  - 2.3.2 A Função f . . . 9
- 2.4 Geração das chaves . . . 14
- 2.5 Desencriptação . . . 17
  - 2.5.1 Geração das chaves pela ordem inversa . . . 17
  - 2.5.2 Desencriptação em redes de Feistel . . . 20

3 AES - Advanced Encryption Standard **21**
- 3.1 Introdução histórica AES . . . 21
- 3.2 Uma visão geral . . . 22
- 3.3 Ideia Base AES . . . 23
- 3.4 Encriptação AES . . . 24
  - 3.4.1 AddRoundKey - Operação XOR . . . 24
  - 3.4.2 SubBytes - 1º Substituição . . . 25
  - 3.4.3 ShiftRows - Permutação . . . 25
  - 3.4.4 MixColumns - 2º Substituição . . . 26
  - 3.4.5 Visão Global Ronda . . . 27
- 3.5 Desencriptação . . . 28
- 3.6 Conclusion . . . 29

Capítulo 1

Conceitos Base

1.1 Segurança de uma cifra

Claude Shannon, matemático e criptógrafo identificou em 1945 na sua obra *A Mathematical Theory of Cryptography* duas propriedades que uma cifra teria que possuir para ser considerada segura. Estas propriedades servem para dificultar métodos de análise estatísticos e outros de criptanálise. Estas duas propriedades são conhecidas como *Confusão* e *Difusão*.

1.1.1 Confusão

Confusão significa que cada digito binário do texto cifrado deve depender de várias partes da chave. Para além disso a ligação entre o texto cifrado, a chave e o texto original deve ser completamente obscuro.

1.1.2 Difusão

Difusão significa que a influência de um bit no texto não cifrado é espalhada sobre vários bits do texto cifrado, com o objectivo de esconder propriedades estatisticas do texto que foi cifrado.

Figura 1.1: Exemplo de difusão

1.2 Cifra de chave simétrica

Um algoritmo de chave simétrica usa a mesma chave tanto para encriptação, como para desencriptação. Esta chave considerada o "segredo"que terá que ser partilhado apenas pelas pessoas que estão a comunicar entre si.
Tanto o DES, como o AES são cifras de chave simétrica.

Capítulo 2

DES - Data Encryption Standard

2.1 Introdução histórica

No início da década de 70, o governo norte-americano sentiu a necessidade de definir um standard de encriptação.
Em 1973 o NBS(National Bureau of Standards), actualmente conhecido como NIST, pediu que fossem enviadas propostas para o que seria conhecido como ***Data Encryption Standard***. No entanto, nenhuma das propostas apresentadas se mostrou viável, o que levou a que fosse feito um novo pedido no verão de 74. Foi nesta altura que a IBM submeteu então a sua proposta, baseada num algoritmo mais antigo desenvolvido por Horst Feistel, também ele membro da IBM. O algoritmo final foi desenvolvido com muito feedback da Agência de Segurança Americana(NSA), o que levantou mais tarde várias suspeitas sobre a integridade da cifra. Este algoritmo foi na altura desenvolvido tendo em mente encriptação/desencriptação feita por meio directo de hardware.

2.2 Uma visão geral

DES é uma cifra que usa uma chave de 56 bits e encripta blocos de 64 bits de cada vez.

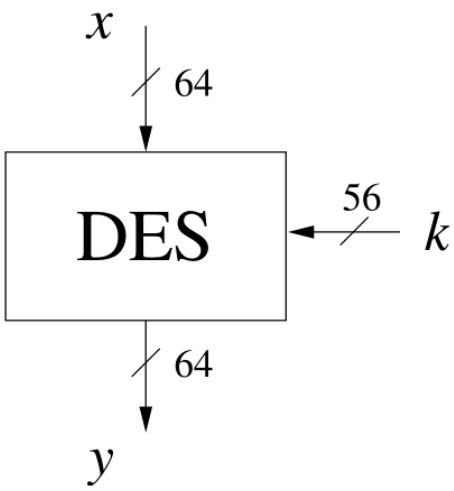

Figura 2.1: Uma visão geral da cifra DES

Este algoritmo é quase como todas as modernas cifras de bloco, um algoritmo iterativo, como se pode observar na figura 2.2 (esta estrutura é muitas vezes chamada de **Rede de Feistel**). Cada bloco, durante o seu processo de encriptação, passa por 16 rondas semelhantes, com a única diferença de cada ronda ter a sua subchave própria, que foi obtida a partir da chave principal.
Esta estrutura permite também que a desencriptação seja feita exactamente da mesma forma que a encriptação, usando apenas as chaves pela ordem inversa, como vamos ver mais tarde.

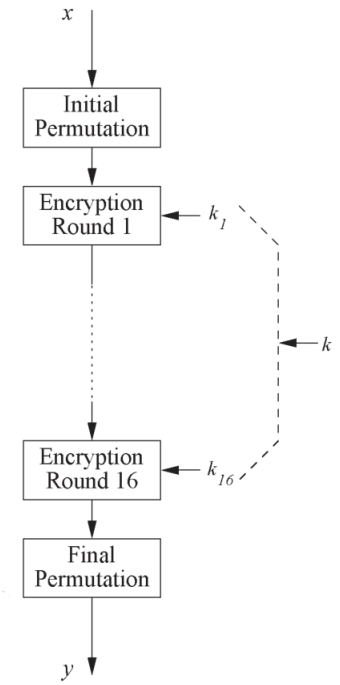

Figura 2.2: A estrutura iterativa do DES

2.3 Encriptação

Depois de uma permutação inicial, o texto é dividido em duas metades L_0 e R_0. Estas duas metades de 32 bits são então o *input* da nossa rede de Feistel, que consistirá de 16 rondas. A metade da direita R_i será dada como argumento da nossa função f e ao seu *output* será feito o XOR (ou exclusivo), com a metade do lado esquerdo L_i. Finalmente estas duas metades serão trocadas (Fig.2.3). Este processo é repetido na

iteração seguinte e pode ser escrito matematicamente da seguinte forma:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

onde i = 1..16. Depois da ronda 16, as metades L_{16} e R_{16} são trocadas novamente e é depois feita uma permutação final IP^{-1}, que inverte a permutação inicial IP.
Em cada ronda, uma chave de ronda de 48 bits k_i é criada a partir da chave principal de 56 bits.

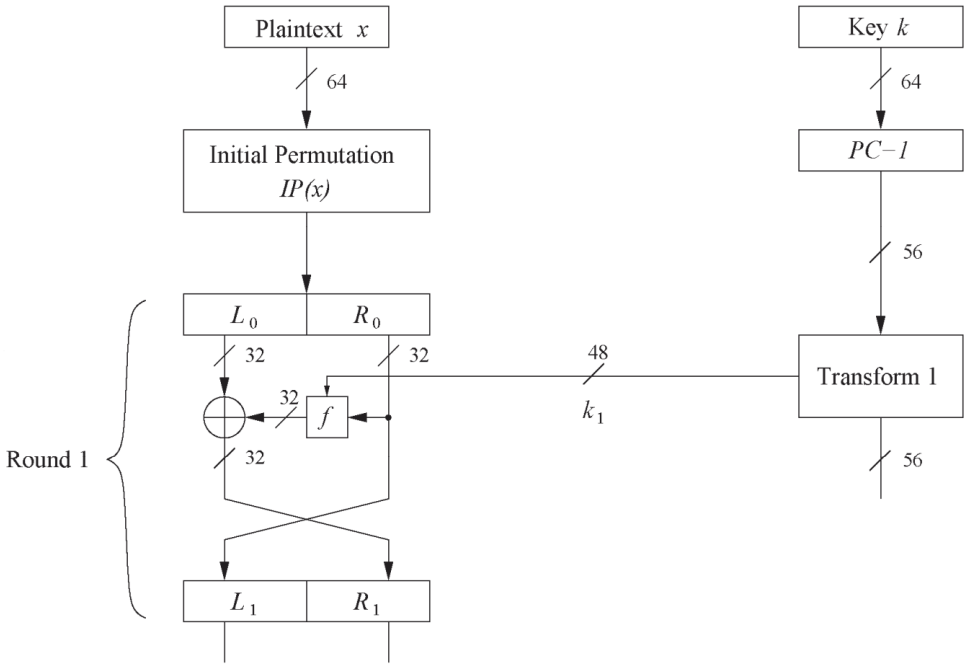

Figura 2.3: A primeira ronda da encriptação(Estrutura de feistel)

As duas propriedades básicas de uma cifra (confusão e difusão) são realizadas dentro da função f que será analisada em detalhe de seguida. Se a função f for considerada segura, então a segurança da cifra de Feistel aumentará com o numero de bits usados na chave e com o numero de rondas. A estrutura da cifra de Feistel de cada ronda mapeia bijectivamente um bloco de 64 bits de entrada a um bloco de 64 bits de saída. Este mapeamento continua bijectivo, mesmo quando a função f é sobrejectiva(mapeamento do tipo *muitos para um*), como acontece no caso do DES. Isto é conseguido porque são usadas internamente operações não lineares e são mapeados 32 bits de input em 32 de output usando uma chave k_i de 48 bits, com $1 \leq i \leq 16$.

2.3.1 A permutação inicial e final

Como se pode observar nas figuras 2.4 e 2.5 estas permutações são feitas bit a bit. Estas permutações não têm nenhum custo em hardware, o que não é verdade quando são implementadas em software.
É importante notar que estas permutações não têm efeito nenhum na segurança do DES. Uma vez que estas tabelas são conhecidas é trivial reverter o seu efeito. Embora não haja uma explicação oficial para este facto é tido como hipótese que tenha servido para facilitar a implementação em hardware.

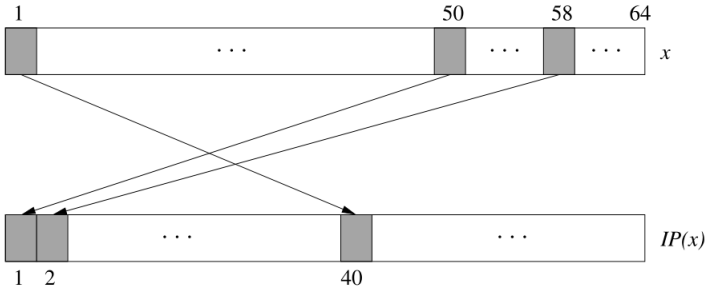

Figura 2.4: Exemplo de troca de bits de acordo com a tabela IP)

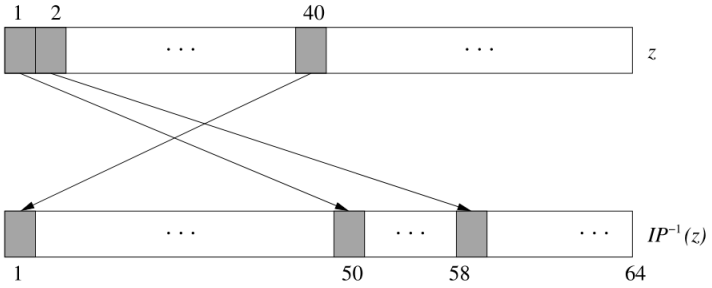

Figura 2.5: Exemplo de troca de bits de acordo com a tabela IP^{-1}

Como podemos ver de acordo com as figuras acima e como seria de prever, pelas siglas a operação IP^{-1} reverte a IP.

IP							
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

(a) Tabela da Permutação inicial IP

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

(b) Tabela da Permutação final IP^{-1}

Figura 2.6: As tabelas que codificam as permutações

2.3.2 A Função f

Como foi referido antes a função f tem extrema importância para a segurança do DES. Na ronda i, a metade direita R_{i-1} do *output* da ronda anterior e a chave da ronda actual k_i são dados como *input*. O *output* desta função é depois usado para fazer o XOR para encriptar a metade esquerda L_{i-1}, como se pode ver na figura 2.3

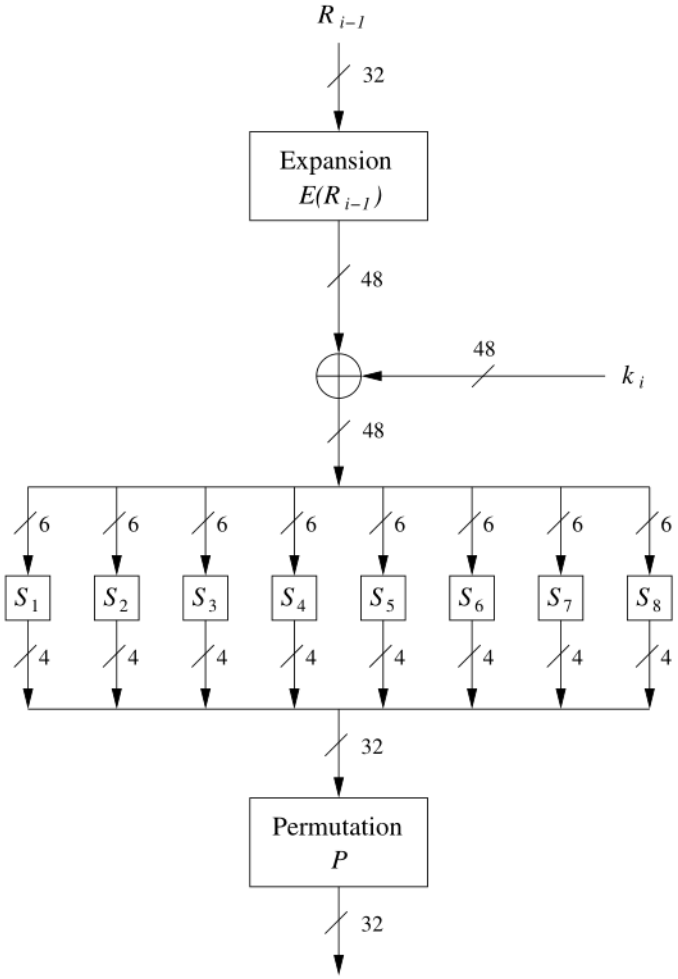

Figura 2.7: O funcionamento da função f

Podemos observar a estrutura da função f na figura 2.7.
Primeiro os 32 bits de entrada, são expandidos para 48 bits. Isto acontece de acordo com a função de expansão e podemos ver um exemplo na figura 2.8

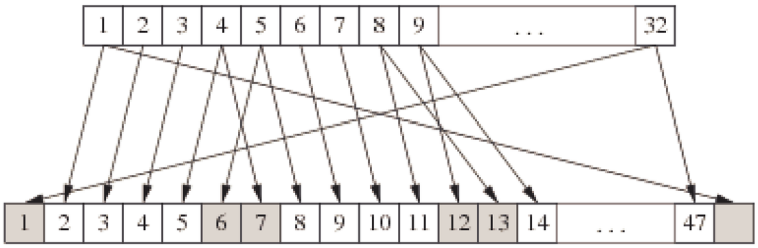

Figura 2.8: Exemplo da troca de bits na função de expansão

Na tabela da figura 2.9 podemos observar que 16 dos 32 bits de *input* aparecem duas vezes nos bits de *output*, no entanto um bit de *input* nunca aparece duas vezes no mesmo bloco de output de 6 bits.
Esta tabela serve para aumentar a difusão, uma vez que certos bits de *input* têm influência em 2 bits de *output*.

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	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	1

Figura 2.9: Tabela que codifica a função de expansão

De seguida, os 48 bits resultantes desta expansão são usados para fazer um XOR com a chave da ronda actual, e cada bloco de 6 bits que foi obtido é passado directamente para oito *caixas* de substituição. Cada caixa destas é uma tabela que mapeira um *input* de 6 bits num *output* de 4 bits. Todas estas tabelas são diferentes como podemos ver na figura 2.11.

Usando como exemplo a codificação do número 37 de acordo com a tabela de substituição S1: Os 4 bits do interior dizem-nos a que coluna iremos procurar (neste caso seria a terceira coluna), enquanto que os bits das pontas nos indicam qual a linha (neste caso seria a quarta). Usando então a tabela, verificamos $S_1(37{=}100101_2){=}8{=}100_2$

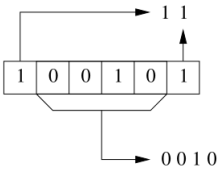

Figura 2.10: Codificação do número 37

Estas tabelas são o mecanismo central do DES em termos criptográficos. São o único elemento não linear do algoritmo e servem para criar *confusão*. A forma como foram escolhidas estas tabelas nunca foi completamente revelada, o que sempre levou a alguma desconfiança e especulação em relação à existência de alguma *backdoor* por parte da NSA.
No entanto hoje é sabido que estas tabelas foram criadas de acordo com os seguintes critérios:

- Cada tabela tem 6 bits de input e 4 de output;
- Nenhum bit de *output* deve estar demasiado perto de uma combinação linear dos bits de *input*
- Se o primeiro e o ultimo bit do *input* estão fixos e os 4 bits do meio são diferentes, então cada possivel combinação dos 4 bits de *ouput* deve ocorrer exactamente uma vez.
- Se dois *inputs* da tabela são diferentes em apenas 1 bit então, os seus outputs devem ser diferentes em pelo menos 2 bits.
- Se dois *inputs* da tabela são diferentes nos 2 bits do meio, então os seus outputs devem ser diferentes em pelo menos 2 bits.
- Se dois inputs da tabela diferem nos primeiros 2 bits e são iguais nos ultimos 2, os *outputs* devem ser diferentes
- Para qualquer diferença de 6 bits entre *inputs*, no máximo 8 dos 32 pares de *inputs* possiveis com essa diferença podem ter a mesma diferença nos *outputs*
- Uma colisão no output das 8 tabelas é apenas possível para três tabelas adjacentes

Como foi referido antes, estas tabelas são cruciais, pois introduzem **não-linearidade**, ou seja:

$$S(a) \oplus S(b) \neq S(a \oplus b)$$

		Column															
Box	**Row**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
S_1																	
	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	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	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	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	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	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	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	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

Figura 2.11: As 8 tabelas de substituição

Sem esta introdução de não-linearidade, um atacante poderia exprimir o *input* e *output* da cifra com um sistema de equações lineares onde os bits da chave seriam as variáveis que queriamos encontrar. No entanto estas tabelas foram desenhadas para prevenir este tipo de ataques matemáticos e em especial os de *criptanálise diferencial*.

O último passo da ronda é então o uso da tabela de Permutação (Figura 2.12).

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Figura 2.12: Tabela que codifica a função de Permutação

Ao contrário do que acontece com as tabelas de permutação IP e IP^{-1}, esta introduz difusão, porque os 4 bits de *output* de cada tabela S são permutados de forma a que afectem várias tabelas S na iteração seguinte. Esta difusão causada pela expansão, tabelas S e tabela P garantem que cada bit no final da quinta ronda é um produto de todos os bits das chave e todos os bits do texto que queriamos cifrar. Esta propriedade é conhecida como **efeito avalanche**.

2.4 Geração das chaves

O mecanismo de geração de chaves cria 16 chaves de ronda k_i, cada uma com 48 bits, a partir da chave original de 64 bits.
Embora o DES receba a chave com 64 bits inicialmente, destes 64, é removido 1 bit de oito em oito posições. Estes 8 bits que são removidos são *bits de paridade*.

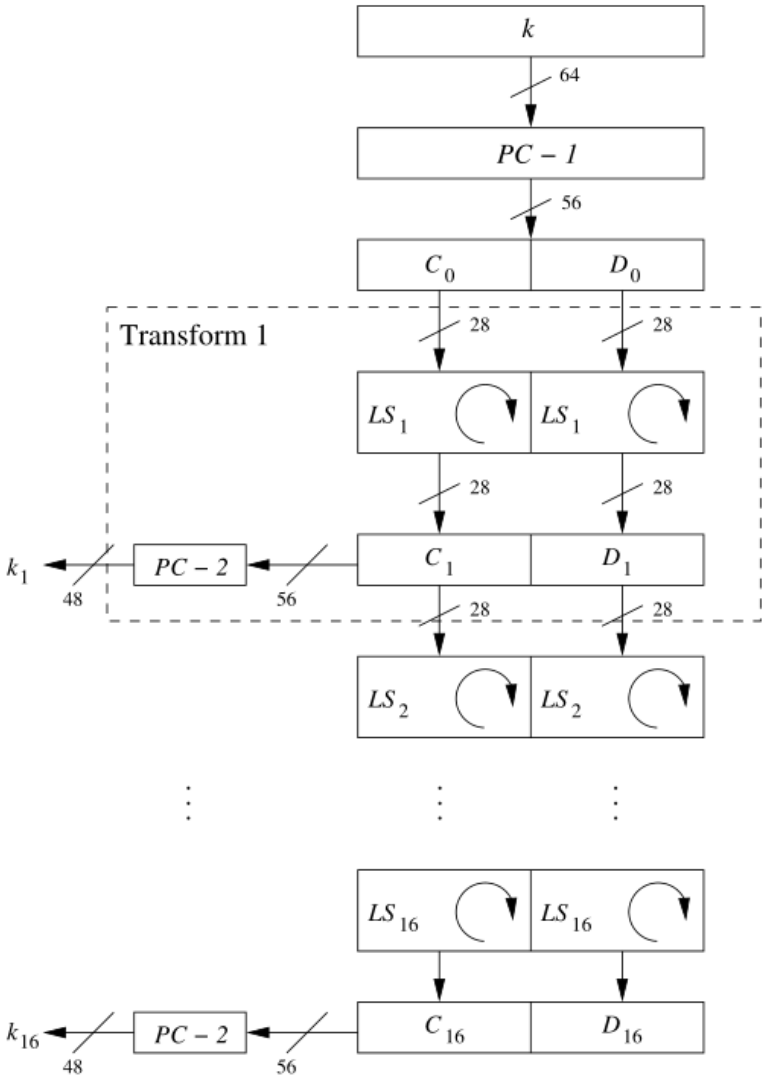

Figura 2.13: Esquema geração das chaves de ronda

Depois de removidos os bits de paridade é feita então uma permutação de acordo com a tabela da figura 2.14.

$PC-1$							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

Figura 2.14: A primeira permutação da chave $PC-1$

Os restantes 56 bits são divididos em duas metades C_0 e D_0. Estas duas metades de 28 bits são então *rodadas* ciclicamente para a esquerda, uma ou duas posições, dependendo da ronda i.

- Se i = 1,2,9 ou 16 as duas metades rodam para a esquerda uma posição
- Caso contrário são rodadas duas posições para a esquerda

Como cada metade é rodada individualmente e o número de rotações total é $4 \times 1 + 12 \times 2 = 28$, temos que $C_0 = C_{16}$ e $D_0 = D_{16}$

Para terminar, as duas metades são permutadas novamente de acordo com a tabela da figura 2.15. Esta tabela permuta 56 bits de $C_i = C_i$ e $D_i = D_i$, ignorando 8 deles.

$PC-2$							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Figura 2.15: A permutação $PC-2$

2.5 Desencriptação

A desencriptação do DES é essencialmente feita da mesma forma que é feita a encriptação, uma vez que se trata de uma cifra de feistel (Fig. 2.16). Comparando à encriptação, apenas a geração da chave é revertida, ou seja, na desencriptação da primeira ronda, iremos precisar da chave da ronda 16; na segunda, da chave da ronda 15, etc. Logo teremos que gerar as chaves pela ordem inversa.

2.5.1 Geração das chaves pela ordem inversa

Como foi apontado na secção 2.4, temos que $C_0 = C_{16}$ e $D_0 = D_{16}$, logo, podemos obter k_{16} directamente depois de PC-1.

$$\begin{aligned} k_{16} &= PC-2(C_{16}, D_{16}) \\ &= PC-2(C_0, D_0) \\ &= PC-2(PC-1(k)) \end{aligned}$$

Para obter k_{15} precisamos de ter as variáveis C_{15} e D_{15}, que podem ser obtidas a patir de C_{16} e D_{16} através de uma rotação para a direita *(RS)*:

$$\begin{aligned} k_{15} &= PC-2(C_{15}, D_{15}) \\ &= PC-2(RS(C_{16}), RS(D_{16})) \\ &= PC-2(RS(C_0), RS(D_0)) \end{aligned}$$

Todas as chaves das rondas anteriores são obtidas de forma semelhante, tendo em atenção para o número de bits que são rodados em cada ronda

- Na primeira ronda não é feita nenhuma rotação
- Na ronda 2,9 e 16, as duas metades rodam para a direita um bit
- Em todas as restantes, as duas metades rodam para a direita 2 bits

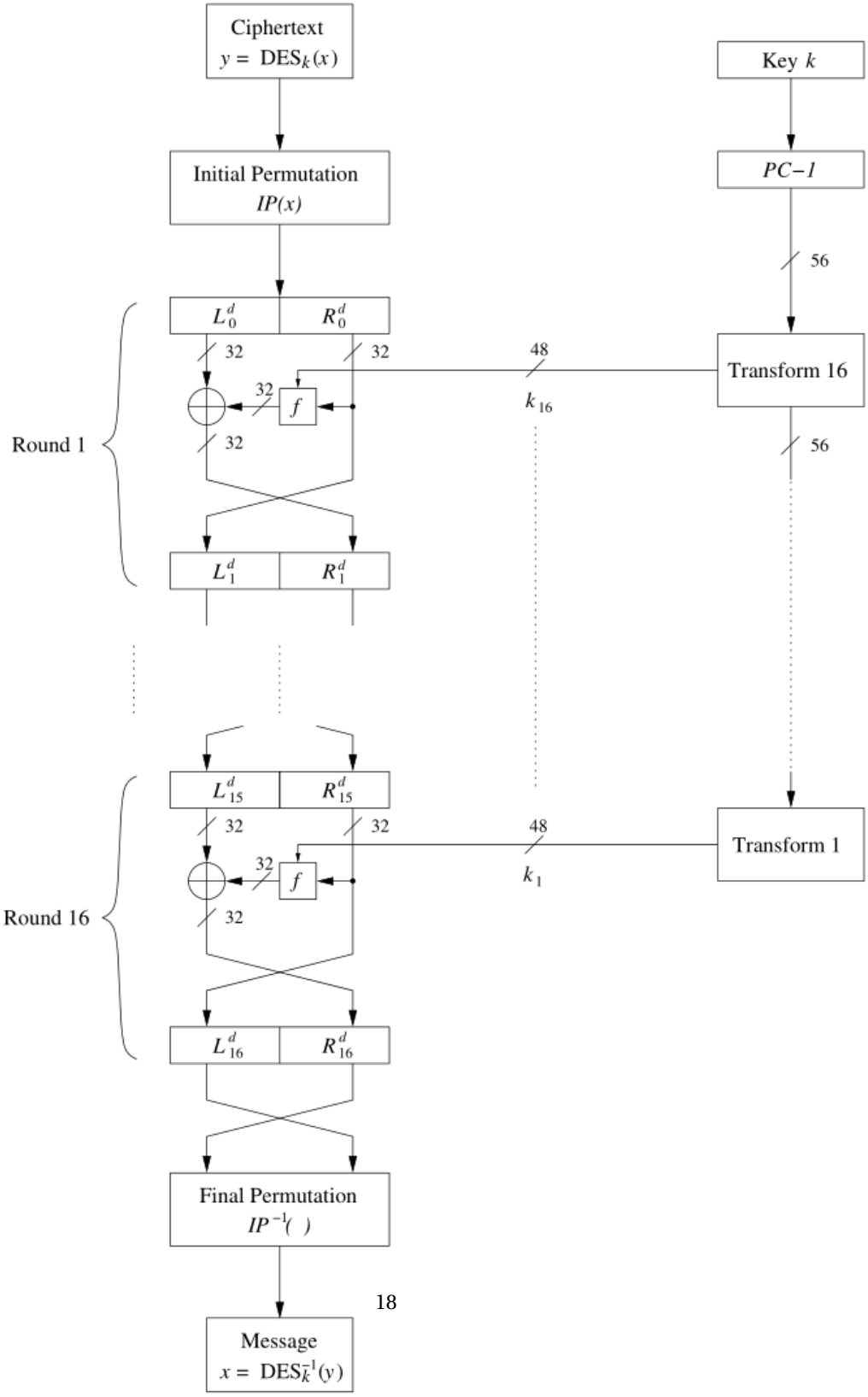

Figura 2.16: A desencriptação no DES

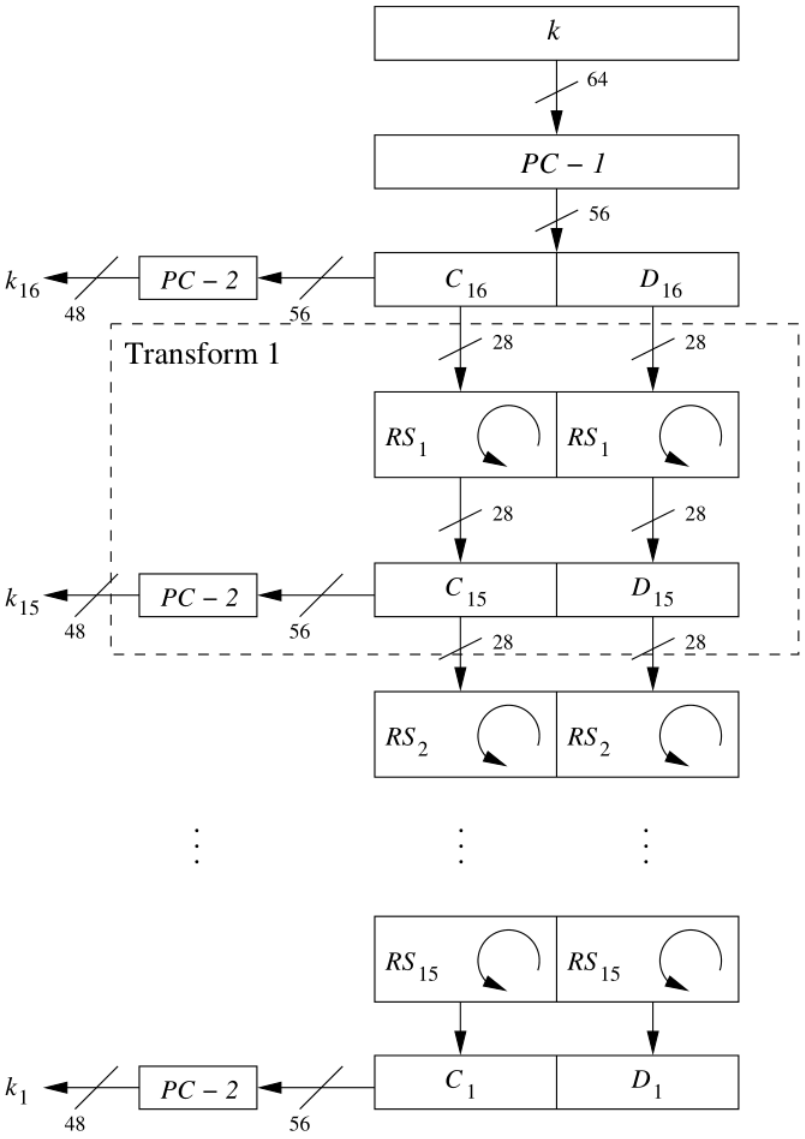

Figura 2.17: O mecanismo de reversão da chave

2.5.2 Desencriptação em redes de Feistel

Como podemos ver na Fig.2.16 a desencriptação é feita ronda a ronda, ou seja, a primeira ronda da desencriptação, desencripta a ultima da encriptação; a segunda de desencriptação, desencripta a 15ª da encriptação, etc. De acordo com a figura e uma vez que a função IP^{-1} inverte a IP temos:

$$(L_0^d, R_0^d) = IP(Y) = IP(IP^{-1}(R_{16}, L_{16})) = (R_{16}, L_{16})$$

Então:

$$L_0^d = R_{16}$$
$$R_0^d = L_{16} = R_{15}$$

Para provar que a primeira ronda da desencriptação, desencripta a ultima ronda da encriptação temos então que provar:

$$L_1^d = R_{15}^e \wedge R_1^d = L_{15}^e$$

A primeira parte é fácil de provar:

$$L_1^d = R_0^d = L_{16}^e = R_{15}^e$$

Para a segunda parte temos:

$$\begin{aligned}
R_1^d &= L_0^d \oplus f(k_{16}, R_0^d) \\
R_1^d &= L_{15}^e \oplus f(k_{16}, R_{15}^e) \oplus f(k_{16}, R_0^d) \\
R_1^d &= L_{15}^e \oplus f(k_{16}, R_{15}^e) \oplus f(k_{16}, R_{15}^e) \\
R_1^d &= L_{15}^e \oplus 0 \\
R_1^d &= L_{15}^e
\end{aligned}$$

E com isto provamos que a primeira ronda da desencriptação de facto desencripta com sucesso a ultima ronda de encriptação. Este processo continua iterativamente e pode ser expresso da seguinte forma:

$$L_1^d = R_{16-i}$$
$$R_i^d = L_{16-i}$$

Finalmente, no ultima ronda da desencriptação, temos que reverter a permutação inicial IP. Para isso:

$$IP^{-1}(R_{16}^d, L_{16}^d) = IP^{-1}(L_0^e, R_0^e) = IP^{-1}(IP(x)) = x$$

Sendo x o texto antes de ser cifrado.

Capítulo 3

AES - Advanced Encryption Standard

3.1 Introdução histórica AES

[] Em 1997 o DES já era visto como um algoritmo não seguro e foi lançado um concurso pelo NIST (National Institute of Standards and Technology), com as condições necessárias que o novo algoritmo deveria conter, algoritmo este que já tinha nome antes de existir, ficaria conhecido por Advanced Encryption Standard.
Este concurso teve umas quantas submissões, mas de todas destacou-se uma block cipher conhecida por Rijndael e em 2001 foi então definada como o novo AES. Os seus criadores foram dois Belgas, Vincent Rijmen e Joan Daemen.

3.2 Uma visão geral

AES é uma cifra de bloco, com blocos de 128 bits e tamanho de key variável com 128,192 ou 256 bits. A key utilizada é uma simetric key, logo um segredo partilhado entre quem envia a mensagem e quem a recebe.

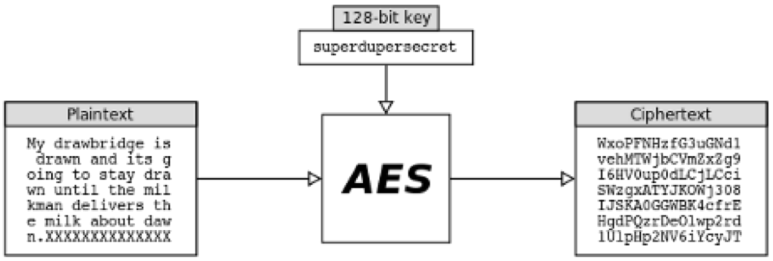

Figura 3.1: Conceito Base AES

Esta cifra baseia-se num tipo específico de cifras de blocos, as cifras de Substituição-Permutação, que tal como o nome indica apresentam apenas operações de sustituição e permutação de bits, e acrescentar a estas a utilização da operação XOR sendo esta a única que utiliza diretamente as keys.

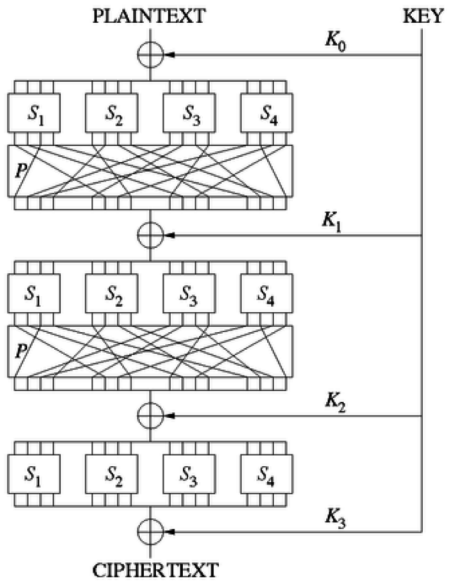

Figura 3.2: Cifra Substituição-Permutação

3.3 Ideia Base AES

Cada um dos blocos de 128 bits de plaintext entram no algoritmo e a primeira operação a que são sujeitos é um XOR com uma key gerada a partir da key original. Segue-se um ciclo com várias rondas onde vão ser repetidas sempre as mesmas 4 operações, duas substituições, uma permutação e um XOR. Exceptuando a última ronda na qual vamos fazer menos uma susbtituição.
O número de rondas vai depender diretamente do tamanho da key tendo para keys de 128 bits 10 rondas, para keys com 192 bits iremos ter 12 rondas, por fim as de 256 bits vão ter 14 rondas.

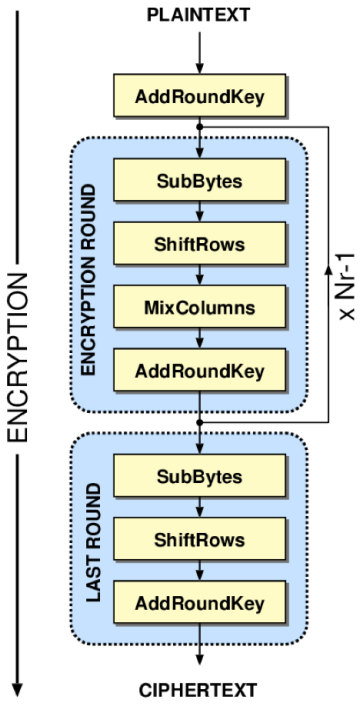

Figura 3.3: Esboço simples AES

3.4 Encriptação AES

Para perceber a encriptação do AES, é necessário primeiro perceber como vão ser compostos cada um dos blocos de 128 bits. Cada bloco de plaintext vai ser dividido em bytes, ficando então com 16 bytes, estes vão-se compor numa matriz 4x4. Importante também referir que as chaves geradas que vão ser utilizadas ao longo de todo o algoritmo vão ter exatamente o mesmo formato.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figura 3.4: Estado blocos 128 bits

3.4.1 AddRoundKey - Operação XOR

Esta para além de ser a primeira operação do algoritmo, é também a última de todas as rondas, fazendo então um XOR com uma key gerada pela Rijndael key schedule. Cada uma das keys geradas vai então combinar-se com o estado proveniente das operações anteriores em cada uma das rondas, uma key por ronda.
Esta operção é a única que utiliza chaves, logo na verdade esta operação é a que vai encriptar os estados de forma a que apenas alguém com exatamente a mesma chave a consiga desencriptar.

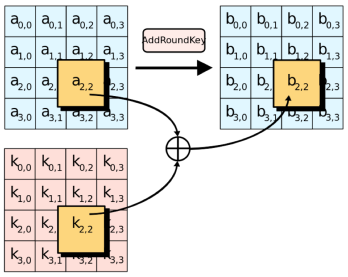

Figura 3.5: Exemplo AddRoundKey

3.4.2 SubBytes - 1º Substituição

As substituições são muito importantes para não existir uma linearidade na cifra, fazendo com que se torne quase impossível de perceber o acontecimento com criptoanálise, apenas sabendo exatamento os valores utilizados nas s-boxes para fazer estas subsituições é que se conseguiria perceber as alterações no estado. Por isso tanto esta como a ronda de mixColumns serem extremamente importantes para a complexidade do algoritmo.
Esta operação específica vai utilizar cada um dos sub-estados de 8 bits(1 byte) em cada um dos estados(128 bits), e combiná-los com s-boxes de 8 bits.

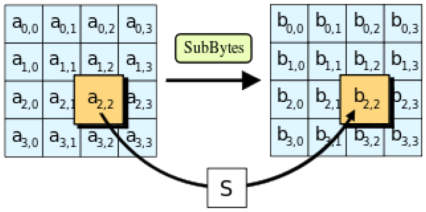

Figura 3.6: Exemplo SubBytes

3.4.3 ShiftRows - Permutação

A permutação vai acontecer em cada uma das linhas do estado, em que cada sub-estado na linha vai-se deslocar um offset de n-1 para a esquerda ,em que n é o número da linha. Logo iremos ter a primeira linha sem qualquer alteração, a segunda deslocar-se 1 casa para a esquerda, a terceira linha 2 casas e a terceira linha 3 casas para a esquerda.

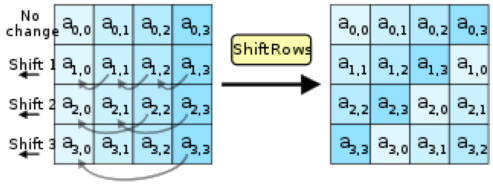

Figura 3.7: Exemplo ShiftRows

3.4.4 MixColumns - 2º Substituição

Esta segunda substituição vai-se processar de maneira diferente da outra, mas acaba por ter sempre a mesma finalidade a não linearidade da cifra.
Cada uma das colunas do estado vai ser multiplicada por uma matriz, matriz esta já pré-definida, criando uma linha totalmente nova com a linha antiga do estado. Esta operação é muito importante para a difusão, pois ao contrário da outra substituição que utiliza individualmente cada um dos sub-estados, esta combina vários sub-estados, logo uma pequena alteração em um bit num sub-estado pode levar a grandes alterações em todos os outros 3 sub-estados que constituem uma coluna.

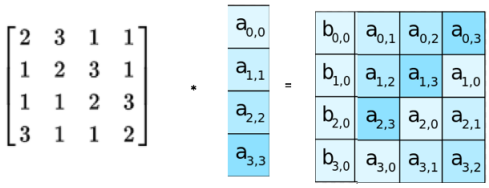

Figura 3.8: Exemplo MixColumns

3.4.5 Visão Global Ronda

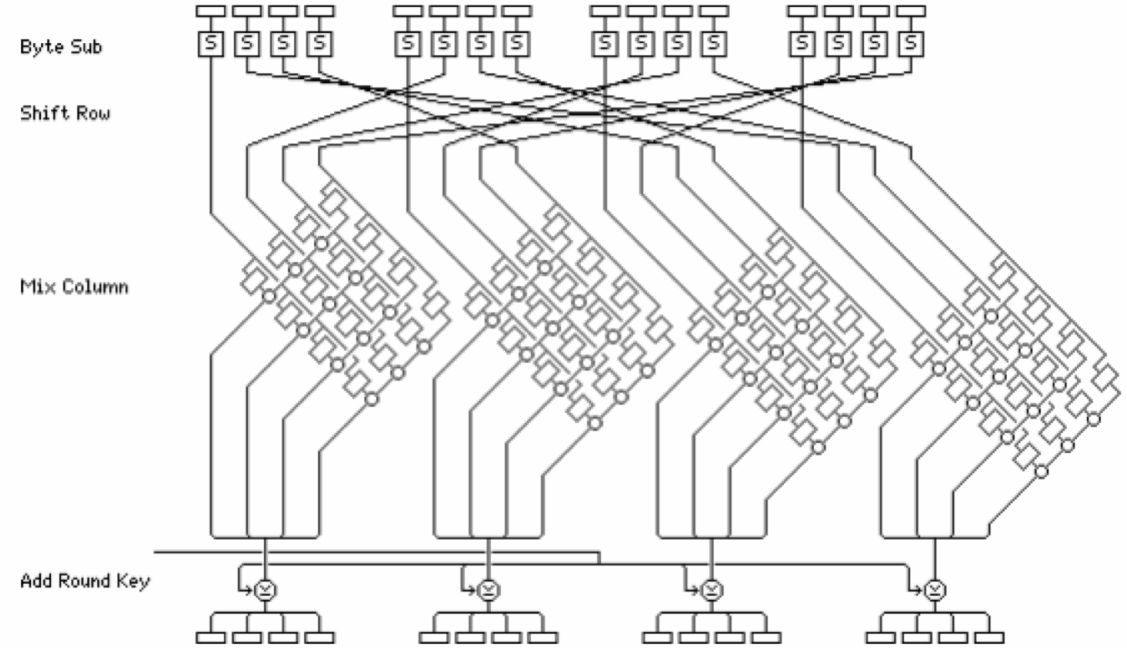

Figura 3.9: Esquema Ronda AES

3.5 Desencriptação

A desencriptação do AES vai de encontro com a noção base de desencriptação de uma cifra de substituição-permutação, ao contrário das de feistel que tinham funções não invertiveis, estas apresentavam as operações inversas de cada uma das operações existentes, fazendo literalmente o caminho inverso que foi feito na encriptação para obter o plaintext inicial.
As keys que foram utilizadas na encriptação vão ser novamente utilizadas mas agora pela ordem contrário, começando com a última e acabando na primeira key gerada na encriptação.

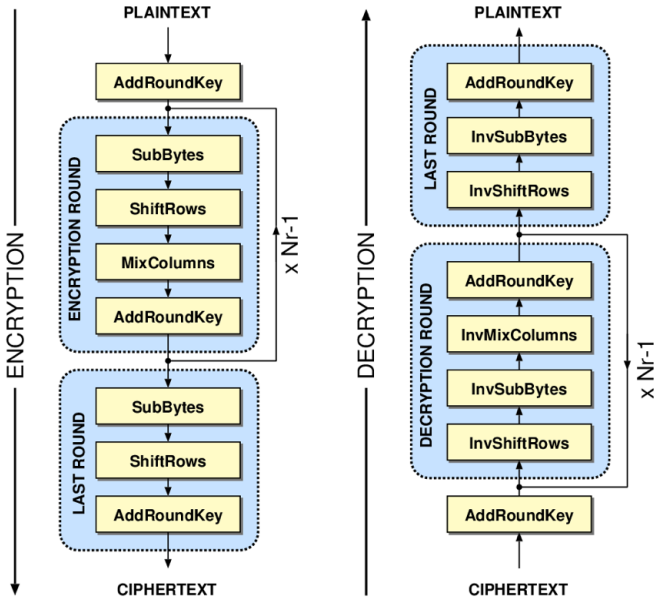

Figura 3.10: Encriptação vs Desencriptação

3.6 Conclusion

Quando nos referimos a algoritmos de encriptação, é necessário retirar certas palavras do dicionário tais como impenetrável, unhackable. Estes algoritmos são sempre falíveis quer a ataques de criptoanálise, quer de força bruta(computação), a complexidade existente dentro de cada um dos algoritmos tal como a suas pré-definições são extremamente importantes para combater este tipo não tornando a sua tarefa impossível mas sim o mais demorada possível

[**?**]

Tabela 3.1: My caption

Tamanho da chave (bits)	**Tempo necessário a 1 encriptação/ μs**	**Tempo necessário a 10^6 encriptações/μs**
32	2^{31} μs = 35.8 minutos	2.15 millisegundos
56	2^{55} μs = 1142 anos	10.01 horas
128	2^{127} μs = 5.4 x 10^{24} anos	5.4 x 10^{18} anos
168	2^{167} μs = 5.9 x 10^{36} anos	5.9 x 10^{30} anos

No final chegamos a conclusão que a criptoanálise teria pouco sucesso, para se tornar na forma standard de quebrar qualquer um destes algoritmos, seria então a key o fator decisivo para a decidir qual o algoritmo mais resistente a ataques neste caso de força bruta.
Como vimos na tabela a cima, é fácil de perceber a influencia do tamanho da key no tempo que demora a quebrar o algoritmo com computação, dai o DES ter acabado inevitávelmente por ser deixado de ser considerado seguro com os seus 56bits de key, aparecendo então o AES, que com os níveis de computação de hoje em dia demoraria cerca de$7.22 * 10^1 8$, quase o dobro da idade do Universo, para ser quebrado.
É impossível saber exatamente quando é que o algoritmo AES será quebrado, depende sempre da evolução da técnologia mas é impensável dizer que nunca o vai ser.

Bibliografia

[1] A.A. Bruen and M.A. Forcinito. *Cryptography, Information Theory, and Error-Correction: A Handbook for the 21st Century*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

[2] H. Silverman Jeffrey Hoffstein, Jill Pipher Joseph. *An Introduction to Mathematical Cryptography*. Springer, 2008.

[3] D. Lin, X.F. Wang, and M. Yung. *Information Security and Cryptology: 11th International Conference, Inscrypt 2015, Beijing, China, November 1-3, 2015, Revised Selected Papers*. Lecture Notes in Computer Science. Springer International Publishing, 2016.

[4] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2009.

[5] William Stallings. *Network security essentials*. Pearson Education, second edition, 2005.

[6] Wikipedia. Confusion and diffusion, 2017. [Online; acedido a 15-Abril-2017].

[7] Wikipedia. Data encryption standard, 2017. [Online; acedido a 15-Abril-2017].