CAPíTULO 1

Cifras de Blocos

Lucas de Carvalho Ferreira
04/05/08

1.1. Definição

Cifras de blocos simétricas são funções que mapeiam blocos de texto original de n bits em blocos cifrados também de n bits, onde n é dito tamanho do bloco. Cifras de blocos são então cifras por substituição cujo tamanho dos caracteres usados é n, que pode ser muito grande. A função de mapeamento é parametrizada por uma chave K de k bits que pode assumir valores de um conjunto $\mathcal{K}$. $\mathcal{K}$ é um subconjunto de V_k, o conjunto de todos os vetores de bits de tamanho k. $\mathcal{K}$ é chamado *espaço de chaves.* A relação entre K, $\mathcal{K}$ e V_k é:

$$K \in \mathcal{K} \subseteq V_k.$$

Para tornar possível o deciframento de uma mensagem, a função de ciframento deve ser uma bijeção, definindo uma permutação de vetores de n bits. Idealmente, cada chave define uma bijeção diferente. O número de chaves possíveis é $|\mathcal{K}|$ e o *tamanho efetivo das chaves* é $\lg|\mathcal{K}|$, que é igual ao tamanho das chaves se todos os vetores de k bits forem chaves válidas ($\mathcal{K} = V_k$). Se as chaves são equiprováveis, então o espaço de chaves tem *entropia* $\lg|\mathcal{K}|$.

Definition 1.1.1. Uma *cifra de blocos* de n bits é uma função $E : V_n \times \mathcal{K} \to V_n$, tal que para cada chave $K \in \mathcal{K}$, $E(P, K)$ é uma mapeamento inversível (a função de ciframento para K) de V_n para V_n, escrito como $E_K(P)$. O mapeamento inverso é a função de deciframento, denotada $D_K(C)$. $C = E_K(P)$ denota que o texto cifrado C é resultado do ciframento do texto em claro P usando K[1].

Uma cifra de blocos geral implementa todas as permutações possíveis, de acordo com a Definição 1.1.2. Para representar a chave de tal cifra, seriam necessários $\lg(2^n!) \approx (n - 1.44)2^n$ bits[2], ou aproximadamente 2^n vezes o número de bits num bloco da mensagem, o que torna

[1]Usamos aqui notação derivada dos termos em inglês: K para chave (*key*), P para texto em claro (*plaintext*), C para texto cifrado (*ciphertext*), E para cifrar (*encrypt*) e D para decifrar (*decrypt*).

[2]É necessária uma chave para cada permutação possível, e são $2^n!$ permutações possíveis. O mínimo número de bits para a representação de m chaves é $\lg m$.

esta cifra impraticável. Um princípio de projeto de cifras de blocos bem aceito afirma que a utilização de uma cifra com uma chave escolhida aleatoriamente deve *parecer* com a utilização uma permutação escolhida aleatoriamente.

DEFINITION 1.1.2. Uma *cifra aleatória* é uma cifra de blocos de n bits que implementa todas as $2^n!$ bijeções em 2^n elementos. Cada uma de $2^n!$ chaves especifica uma destas permutações.

Uma cifra de blocos cujo tamanho n do bloco seja muito pequeno é vulnerável a ataques baseados em análise estatística, como ocorre as cifras clássicas ou pré-computacionais. Este problema poderia ser minimizado pelo uso de encadeamentos. Por outro lado, a escolha de valores muito grandes para n é problemática, já que a complexidade de implementação cresce rapidamente com o tamanho do bloco. Na prática, é necessário utilizar funções que sejam facilmente implementáveis e que *pareçam* aleatórias.

Algumas características desejáveis em cifras de blocos são:

(1) Cada bit do texto cifrado deve depender dos valores de todos os bits da entrada.
(2) Não deve haver nenhuma relação estatística evidente entre o texto cifrado e o texto em claro.
(3) A alteração de um único bit do texto original ou da chave deve alterar cada bit do texto cifrado com probabilidade $\frac{1}{2}$.
(4) A alteração de um dos bits do texto cifrado deve levar a uma alteração imprevisível do resultado do deciframento.

1.2. Cifras de Feistel

Uma importante classes das cifras simétricas modernas segue o modelo chamado *cifra de Feistel*, em homenagem a Horst Feistel, o principal projetista das cifras *Lucifer* e *DES*. Tanto *Lucifer* quanto o *DES* são exemplos de cifras de Feistel. Estas são cifras multiplicativas (*product ciphers*) e também *redes de substituições-permutações*.

DEFINITION 1.2.1. Uma *rede de substituições-permutações (SP)* é uma cifra por produtos composta de um determinado número de estágios, onde cada estágio é composto de substituições e permutações.

Muitas redes SP são também cifras iterativas, como definido abaixo.

DEFINITION 1.2.2. Uma *cifra de blocos iterativa* é uma cifra de blocos que envolve a repetição seqüencial de uma função interna, chamada de *função de iteração*. Os parâmetros deste tipo de cifra incluem o número de iterações (*rounds*), o tamanho do bloco em bits e o tamanho, em bits, da chave K da qual as *sub-chaves* K_i serão derivadas para cada iteração i. Para conseguir inversibilidade, cada valor K_i deve transformar a função de iteração numa bijeção.

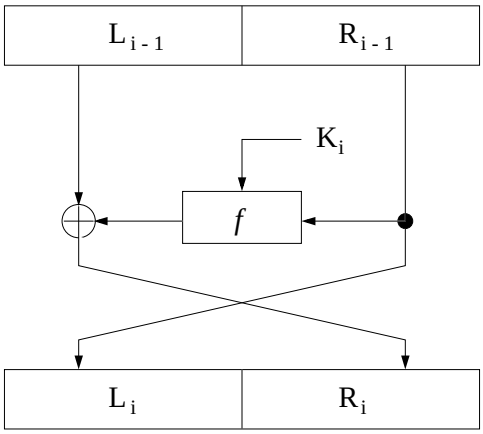

FIGURA 1.2.1. Um estágio de uma cifra de Feistel.

DEFINITION 1.2.3. Uma *cifra de Feistel* é uma cifra iterativa que mapeia um texto em claro de $2t$ bits (L_0, R_0), onde L_0 e R_0 são blocos de t bits, num texto cifrado (R_r, L_r), através de um processo de r iterações onde $r \geq 1$. Para $1 \leq i \leq r$, a iteração i mapeia $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ da seguinte forma:

$$L_i = R_{i-1} \tag{1.2.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i), \tag{1.2.2}$$

onde cada sub-chave K_i é derivada da chave K fornecida para a cifra. A figura 1.2.1 mostra graficamente uma iteração de uma cifra de Feistel.

Tipicamente, uma cifra de Feistel tem um número de iterações $r \geq 3$, com r par na maioria dos casos. Note que a estrutura das cifras de Feistel exige que a ordem dos sub-blocos de saída seja invertida: (R_r, L_r) ao invés de (L_r, R_r), como seria esperado.[3] A figura 1.3.1 mostra graficamente o funcionamento do DES, que é uma cifra de Feistel com $r = 16$ e com a adição de permutações inicial e final (que não fazem parte do modelo de cifra de Feistel). O deciframento da mensagem é realizado pela aplicação do mesmo processo iterativo mas com as chaves sendo aplicadas na ordem inversa, de K_r até K_1. Ou seja, usamos (R_r, L_r) como a entrada do algoritmo, representada como (R_0, L_0) na figura.

Vamos agora mostrar que o mesmo processo pode ser usado tanto para cifrar quanto para decifrar uma mensagem. Definiremos o mapeamento $\pi_f : (\mathbf{x}, \mathbf{x}') \to (\mathbf{x} \oplus f(\mathbf{x}'), \mathbf{x}')$, que é uma involução: $\pi_f^2(\mathbf{x}, \mathbf{x}') = \pi_f(\mathbf{x} \oplus f(\mathbf{x}'), \mathbf{x}') = (\mathbf{x} \oplus f(\mathbf{x}') \oplus f(\mathbf{x}'), \mathbf{x}') = (\mathbf{x}, \mathbf{x}')$. Lembre-se que $f(\mathbf{x}) \oplus f(\mathbf{x}) = 0$, qualquer que seja o valor de $f(\mathbf{x})$ e que $\mathbf{x} \oplus 0 = \mathbf{x}$. A transformação π_f corresponde a uma iteração de uma cifra de Feistel sem a etapa de inversão das posições das entradas.

[3]Usamos aqui a notação original do inglês, onde L e R são as iniciais de *left* (esquerda) e *right* (direita) respectivamente.

Definiremos outro mapeamento ϑ como sendo $\vartheta : (\mathbf{x}, \mathbf{x}') \to (\mathbf{x}', \mathbf{x})$, ou seja, uma troca de posição dos dois elementos da entrada. Por fim, vamos definir $\tau_f = \vartheta \times \pi_{f_n} \times \ldots \times \vartheta \times \pi_{f_1}$, uma seqüência de n aplicações de π_{f_i} e ϑ, onde π_{f_i} é a transformação π_f usando f_i como função interna. Note que $\vartheta \times \pi_{f_i}$ corresponde à i-ésima iteração de uma cifra de Feistel, mostrada na figura 1.2.1, e que τ_f corresponde à operação completa.

Como sabemos que $\pi_f^2 = \vartheta^2 = \mathbf{I}$, onde $\mathbf{I}$ é a transformação identidade, podemos facilmente construir τ_f^{-1}, a operação de deciframento:

$$\tau_f^{-1} = \pi_{f_1} \times \vartheta \times \ldots \times \pi_{f_n} \times \vartheta.$$

É fácil perceber que

$$\begin{aligned}
\tau_f^{-1} \times \tau_f &= \pi_{f_1} \times \vartheta \times \ldots \times \pi_{f_n} \times \vartheta \times \vartheta \times \pi_{f_n} \times \ldots \times \vartheta \times \pi_{f_1} \\
&= \pi_{f_1} \times \vartheta \times \ldots \times \pi_{f_n} \times \mathbf{I} \times \pi_{f_n} \times \ldots \times \vartheta \times \pi_{f_1} \\
&= \pi_{f_1} \times \ldots \times \vartheta \times \pi_{f_n}^2 \times \vartheta \times \ldots \times \pi_{f_1} \\
&= \pi_{f_1} \times \ldots \times \vartheta^2 \times \ldots \times \pi_{f_1} \\
&= \pi_{f_1} \times \ldots \times \pi_{f_{n-1}} \times \pi_{f_{n-1}} \times \ldots \times \pi_{f_1} \\
&\vdots \\
&= \mathbf{I}
\end{aligned}$$

Ou seja, aplicação de τ_f seguida da aplicação de sua inversa correspondem à transformação identidade (não altera a entrada). Como vimos que $\tau_f^{-1} \times \tau_f = \mathbf{I}$, podemos concluir que uma cifra de Feistel, cujos processos de ciframento e deciframento foram descritos acima, é uma involução, independente da função f utilizada. A função f influi apenas na dificuldade do processo de criptoanálise da cifra.

1.3. Data Encryption Standard (DES)

Em 1976 o governo dos Estado Unidos padronizou o Data Encryption Standard (DES) para aplicações que necessitem de proteção criptográfica mas que não lidem com segredos de estado e também para aplicações comerciais. O mesmo algoritmo foi depois adotado como padrão pela ANSI sob o nome de *Data Encryption Algorithm* (DEA) e pela ISO (DEA-1).

O DES é, sem dúvida, o algoritmo criptográfico mais importante dos últimos 50 anos e, ainda hoje, é uma cifra bastante popular em todo o mundo, em especial na variação 3DES ou *triplo DES*, que consiste na aplicação iterada do DES. Certamente nenhuma outra cifra recebeu tanta atenção dos especialistas ou foi sujeita a tantos e tão variados ataques quanto o DES.

1.3.1. Histórico. No início da década de 70, o governo e algumas empresas dos Estados Unidos perceberam que havia necessidade de definição de uma cifra padronizada para uso em comunicações civis e para dados governamentais não classificados. Os produtos criptográficos da época eram baseados em algoritmos proprietários e que, na maioria das vezes, eram mantidos em segredo pelos fabricantes.

Sabia-se que os militares estudavam criptografia e aplicações pelo menos desde a segunda guerra mundial e que o governo dos EUA contava com a *National Security Agency*, a agencia nacional especializada em interceptação de informações. O conhecimento disponível fora dos círculos militares e de espionagem era bastante limitado na época.

Em 1972, o *National Bureau of Standards* (NBS, hoje NIST, *National Institute of Standards and Technology*) dos EUA, que corresponderia à ABNT (Associação Brasileira de Normas Técnicas) no Brasil, iniciou o processo de padronização solicitando propostas de algoritmos para se tornarem o futuro padrão criptográfico dos Estados Unidos.

Dentre as respostas recebidas pelo NBS, havia uma da IBM. Era um algoritmo desenvolvido no início dos anos 70 chamado *Lucifer*. O NBS solicitou então a ajuda da NSA para avaliar a segurança desta cifra. A NSA fez algumas alterações pontuais no algoritmo, como o tamanho das chaves e algumas de suas funções internas, e deu um parecer favorável à sua adoção como padrão.

O fato do algoritmo ter sido alterado pela NSA foi muito criticado na época, principalmente pela suspeita de que a agência poderia ter incluído um *trap door*[4] no algoritmo para permitir a interceptação de comunicações privadas. O fato do tamanho das chaves ter diminuído também foi alvo de muitas críticas de pesquisadores que afirmavam que o espaço de chaves estava muito reduzido para que a cifra fosse resistente a um ataque por força bruta (tentativa de decifrar usando cada uma das possíveis chaves). Como veremos na Seção 1.3.3, algumas das alterações tornaram a cifra mais resistente a ataques desconhecidos do meio acadêmico da época.

O padrão que instituiu o DES requer que a cifra seja revisada a cada cinco anos. Desde que passou a valer, em 1978, o padrão foi renovado a cada período de cinco anos, a última renovação ocorrendo em 1998. O NIST escolheu o sucessor do DES, chamado de *Advanced Encryption Standard* (AES), que está lentamente substituindo o DES e o 3DES na prática.

1.3.2. Descrição do Algoritmo. O DES é uma cifra simétrica de blocos cujo tamanho dos blocos é 64 bits e o tamanho da chave é 56 bits. A chave é normalmente expressa com 64 bits, com oito bits

[4]do inglês, algo como porta dos fundos. Diz-se que existe uma *trap door* quando há uma maneira secreta de decifrar o texto cifrado com um algoritmo mesmo sem o conhecimento da chave usada.

de paridade além dos 56 que realmente compõem a chave. O DES foi desenvolvido usando dois conceitos definidos anteriormente: cifras multiplicativas e cifras de Feistel (Seção 1.2).

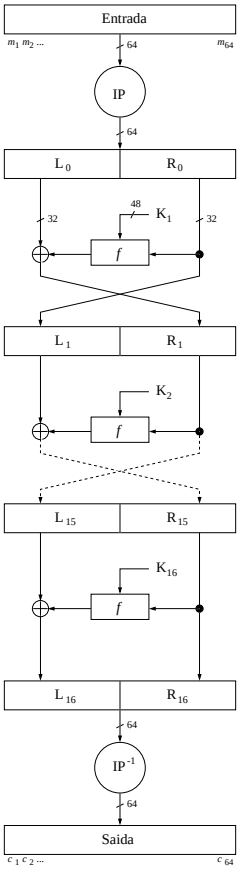

FIGURA 1.3.1. O DES mostrado como um diagrama.

O DES executa os seguintes passos (Figura 1.3.1):

(1) Dado bloco de entrada x, a cadeia x_0 é construída por permutação dos bits de x de acordo com uma *permutação inicial* IP. Escrevemos $x_0 = IP(x) = L_0R_0$ onde L_0 consiste dos primeiros 32 bits de x_0 e R_0 dos últimos 32 bits.

(2) 16 iterações (*rounds*) são executados, ou seja, calculamos L_iR_i, $1 \leq i \leq 16$ usando a fórmula das cifras de Feistel (seção anterior):

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned}$$

onde $\oplus$ corresponde à operação de ``ou exclusivo" de duas cadeias de bits. A função f será descrita em detalhes mais adiante e $K_1, K_2, \ldots, K_{16}$ são cadeias de 48 bits calculadas a partir da chave K.

(3) A permutação inversa IP^{-1} é aplicada à cadeia $R_{16}L_{16}$, obtendo o texto cifrado y. Ou seja, $y = IP^{-1}(R_{16}L_{16})$. Note que $R_{16}L_{16}$ são usados com sua ordem invertida.

A função f tem como argumentos A, que é uma cadeia de 32 bits e J, que é uma cadeia de 48 bits, para produzir como saída uma cadeia de 32 bits. Os passos envolvidos no calculo desta função são:

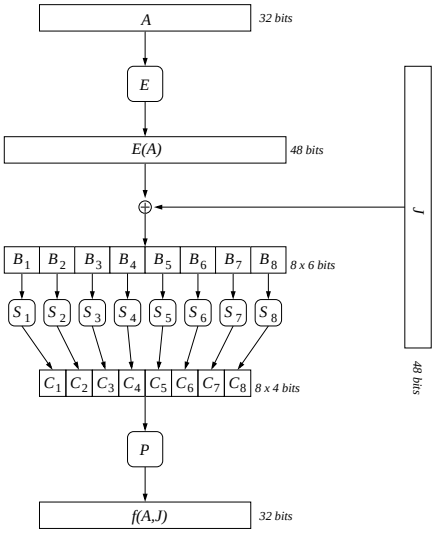

FIGURA 1.3.2. A função interna f usada no DES.

(1) O argumento A é "expandido", ou seja, transformado numa cadeia de 48 bits de acordo com uma *função de expansão* E. $E(A)$ consiste dos 32 bits de A permutados e com 16 destes bits aparecendo duas vezes.
(2) Calcular $E(A) \oplus J$ e escrever o resultado como a concatenação de oito cadeias de seis bits $B = B_1B_2B_3B_4B_5B_6B_7B_8$.
(3) Este passo usa oito *caixas de substituição* $S_1, \ldots, S_8$. Cada S_i é uma matriz 4×16 fixa cujas entradas são inteiros entre 0 e 15. Dada uma cadeia de seis bits $B_j = b_1b_2b_3b_4b_5b_6$, podemos calcular $C_j = S_j(B_j), 1 \leq j \leq 8$ da seguinte maneira:
   (a) os bits b_1b_6 determinam uma linha l de S_j $(0 \leq r \leq 3)$
   (b) os quatro bits $b_2b_3b_4b_5$ determinam uma coluna c de S_j $(0 \leq c \leq 15)$.
   (c) $S_j(B_j)$ é definido como a entrada $S_j(r, c)$, escrito em binário usando quatro bits. (Cada S_j pode ser tratado como uma função de dois argumentos, uma cadeia de dois bits e uma cadeia de quatro bits que retorna uma cadeia de quatro bits).
(4) A cadeia $C = C_1C_2C_3C_4C_5C_6C_7C_8$ de 32 bits é então permutada de acordo com uma permutação P fixa. Definimos $f(A, J)$ como sendo a cadeia resultante $P(C)$.

A função f está representada na figura 1.3.2. Esta função consiste de uma substituição, usando as *caixas de substituição*, seguida de uma permutação P. Esta função segue os princípios introduzidos por Shannon de *confusão* e *difusão*. As 16 iterações de f constituem uma cifra multiplicativa.

Apresentaremos agora os detalhes das funções usadas no DES, começando pela permutação inicial (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

Esta tabela indica que o 58º bit de x é o primeiro bit de $IP(x)$, o 50º bit de x é o segundo bit de $IP(x)$ etc.

A permutação inversa IP^{-1} é dada por:

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

A função de expansão E é especificada a seguir:

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

As oito caixas de substituição e a permutação P usadas para calcular f são:

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

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

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

Agora é necessário descrever como calcular as sub-chaves usadas em cada iteração do DES, que são derivadas da chave K. K é, na verdade, uma cadeia de 64 bits, dos quais 56 constituem a chave e 8 (aqueles que se encontram nas posições 8, 16, ..., 64) são usados para verificação da paridade (paridade ímpar). Os bits de paridade são ignorados no algoritmo abaixo, que calcula as chaves intermediárias:

(1) Dados os 64 bits, ignore os bits de paridade e permute os bits restantes de acordo com a permutação PC_1. Escrevemos $PC_!(K) = C_0D_0$, onde C_0 consiste dos primeiros 28 bits e D_0 dos outros 28 bits.

(2) Para cada iteração i, faça

$$\begin{aligned} C_i &= LS_i(C_{i-1}) \\ D_i &= LS_i(D_{i-1}) \\ K_i &= PC-2(C_iD_i), \end{aligned}$$

onde LS_i representa a rotação para a esquerda de uma ou duas posições, dependendo do valor de i: rotação de uma posição se $i = 1$, 2, 9 ou 16, e rotação de duas posições caso contrário. $PC-2$ é outra permutação fixa e é indicada abaixo.

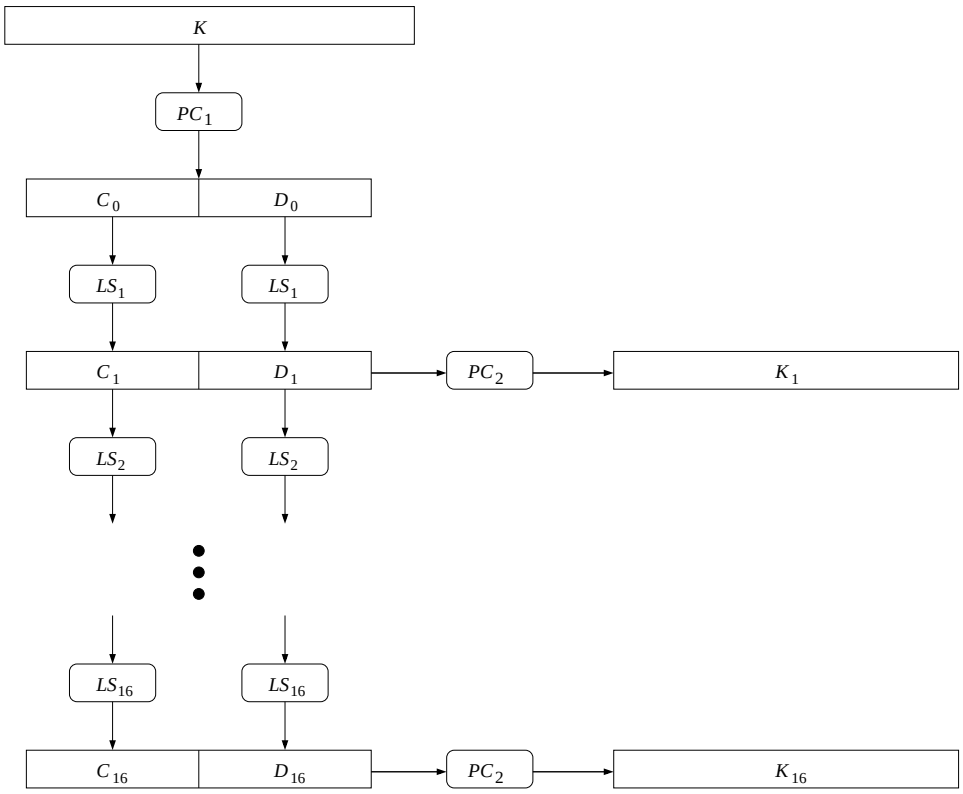

FIGURA 1.3.3. Calculo das chaves intermediárias do DES.

A figura 1.3.3 apresenta como são calculadas as chaves intermediárias. As permutações PC_1 e PC_2 são mostradas abaixo:

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

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

1.3.2.1. *Deciframento.* O DES é uma cifra de Feistel, o que indica que, para decifrar uma mensagem, basta usar o mesmo algoritmo com as chaves intermediárias na ordem contrária. Isto é, usamos a chave K_{16} no primeiro estágio de deciframento, a chave K_{15} no segundo estágio e

assim por diante até o uso da chave K_1 no último estágio. Isto é suficiente para obter a mensagem original, como foi mostrado na Seção 1.2 para toda cifra de Feistel. Para conseguirmos as chaves intermediárias na ordem necessária para o deciframento, basta usarmos o algoritmo de geração de sub-chaves mostrado acima para o ciframento com as seguintes alterações:

(1) usar rotações para a direita ao invés de rotações para a esquerda;
(2) usar os mesmos números de rotações, exceto na primeira iteração ($i = 1$), onde não será realizada nenhuma rotação.

1.3.3. Propriedades. Vamos agora apresentar algumas das principais propriedades e anomalias do DES. É importante lembrar que o DES satisfaz as características desejáveis para cifras de blocos, que foram apresentadas na Seção 1.1.

1.3.3.1. *Propriedade de Complementação.* Usando E para denotar o algoritmo DES e $\overline{x}$ para denotar o complemento de x, temos que $y = E_K(x)$ implica em $\overline{y} = E_{\overline{K}}(\overline{x})$. Isto significa que o complemento de um texto cifrado pode ser obtido pelo ciframento do complemento do texto de entrada com o complemento da chave original.

Vamos agora verificar porque isto ocorre. Suponha que a i-ésima iteração do DES recebe $\overline{L_{i-1}}$e $\overline{R_{i-1}}$ como entradas e uma chave $\overline{K_i}$. Neste caso, a entrada para a função f será $(\overline{R_{i-1}}, \overline{K_i})$. Observe na Figura 1.3.2 que a primeira operação da função f é uma expansão de $\overline{R_{i-1}}$, que apenas reordena e repete alguns bits. O resultado deste operação é então $\overline{E(R_{i-1})}$, que será usado como entrada para a operação de ou-exclusivo junto com a chave $\overline{K_i}$. Sabemos que, na operação de ou-exclusivo, $x \oplus y = \overline{x} \oplus \overline{y}$, o que nos permite inferir que as entradas apresentadas às caixas de substituição serão as mesmas se usarmos os valores originais ou seus complementos. Por conseqüência, o resultado da função f é o mesmo se usarmos R_{i-1} e K_i como entradas ou seus complementos.

Voltando a uma iteração do DES, agora já sabemos que se complementarmos todas as entradas não alteraremos o resultado da função f. Lembrando que $x \oplus \overline{y} = \overline{x \oplus y}$, concluímos que o resultado de uma iteração que tem como entradas $\overline{L_{i-1}}, \overline{R_{i-1}}$ e $\overline{K_i}$ resultará em dois blocos de 32 bits:

$$\begin{aligned} \overline{R_{i-1}} &= \overline{L_i} \\ \overline{L_{i-1}} \oplus f\left(\overline{R_{i-1}}, \overline{K_i}\right) = \overline{L_{i-1}} \oplus f\left(R_{i-1}, K_i\right) = \overline{L_{i-1} \oplus f(R_{i-1}, K_i)} &= \overline{R_i}. \end{aligned}$$

Desta forma percebemos que, se usarmos os complementos como entrada de uma iteração do DES, obteremos as saídas também complementadas. Como este resultado vale para cada uma das 16 iterações do DES, podemos concluir que este resultado será válido também para as 16 iterações em conjunto, ou seja, o algoritmo completo.

Esta propriedade permite reduzir o número de chaves necessárias a uma busca exaustiva de 2^{56} para 2^{55}, o que não chega a ser um ataque eficiente.

1.3.3.2. *Chaves fracas e semi-fracas.* No DES, se todas as sub-chaves $K_1, \ldots, K_{16}$ são iguais, então o algoritmo se resume à aplicar 16 vezes uma mesma iteração. A transformação inversa, o deciframento, será então a aplicação do mesmo algoritmo. Chaves que levam à geração de 16 sub-chaves iguais são chamadas *chaves fracas.*

DEFINITION 1.3.1. Uma *chave fraca* para o DES é uma chave K tal que $E_K(E_K(x)) = x$, para todo x, definindo uma involução. Um par de *chaves semi-fracas* é um par de (K_1, K_2) tal que $E_{K_1}(E_{K_2}(x)) = x$.

A operação de ciframento com uma das chaves de um par de chaves semi-fracas corresponde à operação de deciframento com a outra chave. O DES tem quatro chaves fracas e seis pares de chaves semi-fracas. As tabelas abaixo mostram as chaves fracas e semi-fracas do DES e os valores correspondentes às variáveis C_0 e D_0 usadas no algoritmo de geração de sub-chaves. Denotaremos por $\{0\}^j$ j repetições do bit 0 .

Chave fraca (hexadecimal)	C_0	D_0
`0101 0101 0101 0101`	$\{0\}^{28}$	$\{0\}^{28}$
`FEFE FEFE FEFE FEFE`	$\{1\}^{28}$	$\{1\}^{28}$
`1F1F 1F1F 0E0E 0E0E`	$\{0\}^{28}$	$\{1\}^{28}$
`E0E0 E0E0 F1F1 F1F1`	$\{1\}^{28}$	$\{0\}^{28}$

C_0	D_0	Chaves semi-fracas (hexadecimal)	C_0	D_0
$\{01\}^{14}$	$\{01\}^{14}$	`01FE 01FE 01FE 01FE, FE01 FE01 FE01 FE01`	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{01\}^{14}$	`1FE0 1FE0 0EF1 0EF1, E01F E01F F10E F10E`	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{0\}^{28}$	`01E0 01E0 01F1 01F1, E001 E001 F101 F101`	$\{10\}^{14}$	$\{0\}^{28}$
$\{01\}^{14}$	$\{1\}^{28}$	`1FFE 1FFE 0EFE 0EFE, FE1F FE1F FE0E FE0E`	$\{10\}^{14}$	$\{1\}^{28}$
$\{0\}^{28}$	$\{01\}^{14}$	`011F 011F 010E 010E, 1F01 1F01 0E01 0E01`	$\{0\}^{28}$	$\{10\}^{14}$
$\{1\}^{28}$	$\{01\}^{14}$	`E0FE E0FE F1FE F1FE, FEE0 FEE0 FEF1 FEF1`	$\{1\}^{28}$	$\{10\}^{14}$

Note que as variáveis C_0 e D_0 das chaves fracas são sempre cadeias apenas com 0 ou apenas com 1 e que o rotação destas cadeias não as altera, levando à geração de sub-chaves K_i que são sempre iguais.

DEFINITION 1.3.2. *Pontos fixos* são textos de entrada para os quais $E_K(x) = x$, ou seja, o texto cifrado é igual ao texto original.

Para cada uma das chaves fracas do DES, existem 2^{32} pontos fixos e quatro das doze chaves semi-fracas também têm 2^{32} *pontos fixos complementares*, ou seja, texto de entrada x tais que $E_K(x) = \overline{x}$. Estas quatro chaves semi-fracas são as que estão na parte superior da tabela acima.

1.3.3.3. *DES não é um grupo.* Para uma dada chave K, o DES define uma permutação de $\{0,1\}^{64}$ para $\{0,1\}^{64}$. O conjunto das chaves do DES define 2^{56} permutações (potencialmente diferentes). Se, para quaisquer chaves K_1 e K_2 dadas, existe uma terceira chave K_3 tal que $E_{K_3}(x) = E_{K_2}(E_{K_1}(x))$ para todo x, então o ciframento múltiplo usando o DES seria equivalente a um único ciframento. Felizmente, este não é o caso e o ciframento múltiplo pelo DES pode ser usado para aumentar a segurança da cifra.

1.3.3.4. *Criptanálise.* Se assumirmos que é possível obter uma quantidade enorme de textos conhecidos e seus respectivos ciframentos, então a técnica de criptanálise linear constitui o ataque mais poderoso conhecido contra o DES atualmente. Entretanto, dada a complexidade da obtenção de um tal banco de dados de textos conhecidos, este ataque não é considerado uma ameaça ao uso do DES na prática.

A criptanálise diferencial é uma das técnicas mais gerais conhecidas para atacar cifras de blocos iterativas modernas, incluindo o DES. Esta técnica é, no entanto, um ataque de texto escolhido, o que também dificulta sua aplicação na prática.

A tabela abaixo apresenta a complexidade dos melhores ataques conhecidos contra o DES. A coluna *complexidade de processamento* apresenta uma estimativa do custo esperado em termos de execuções do DES e as porcentagens indicam a probabilidade de sucesso.

Método de ataque	**complexidade de dados**		**complexidade de armazenamento**	**complexidade de processamento**
	conhecido	**escolhido**		
precomputação	–	1	2^{56}	1
busca exaustiva	1	–	negligível	2^{56}
criptanálise linear	2^{43} (85%)	–	para os textos	2^{43}
	2^{38}	–	para os textos	2^{50}
criptanálise diferencial	–	2^{47}	para os textos	2^{47}
	2^{55}	–	para os textos	2^{55}

Em relação ao armazenamento, tanto a criptanálise linear quanto a diferencial requerem pouco espaço. Os textos podem ser processados individualmente e descartados, embora na prática, se o texto cifrado for obtido antes do início do ataque, será necessário o armazenamento dos textos.

A possibilidade de executar 2^{56} vezes o DES para um ataque por busca exaustiva é, na prática, bem maior que a possibilidade de obtenção de 2^{43} pares de textos conhecidos. E, a partir do momento em que se obteve cerca de 2^{32} textos cifrados, torna-se mais eficiente a execução de ataques por dicionário para recuperar partes do texto cifrado.

1.3.3.5. *Tamanho da Chaves.* A forma original do DES submetida ao NBS usava, assim como seu predecessor *Lucifer*, chaves de 112 bits.

A redução para 56 bits ocorreu durante o processo de avaliação e adaptação do DES para se tornar um padrão e foi duramente criticada pela comunidade científica já nesta época.

Durante muitos anos especulou-se qual a complexidade, e conseqüente custo, de construção de uma máquina capaz de quebrar o DES por busca busca exaustiva. Das estimativas iniciais de alguns milhões de dólares na década de 80 à construção e demonstração de tal equipamento em 1997 muitas teorias e projetos foram apresentados. Em 1997, a *Electronic Frontier Foundation* (EFF) demonstrou e publicou os detalhes de construção de um equipamento de US$ 10.000,00 capaz de realizar busca exaustiva por uma chave DES em algumas horas, levando à conclusão de que o DES não era mais seguro o suficiente.

A partir disto, popularizou-se o uso do Triplo DES, que consiste na aplicação do algoritmo DES padrão por três vezes no texto original. A divulgação do *DES cracker* acelerou também o processo para substituição do DES pelo AES.

1.3.3.6. *Número de iterações.* O DES foi especificado com 16 iterações e as razões que levaram seus projetistas a escolher este número nunca foram esclarecidas oficialmente. Avaliações empíricas mostram que, após cinco iterações, cada bit da saída é função de todos os bits da entrada (texto original e chave). Após oito iterações, o texto cifrado é essencialmente uma função aleatória de todos os bits da entrada. Esta dependência crescente dos bits da saída em uma quantidade cada vez maior de bits da entrada é conhecida como *efeito avalanche*[5].

Desde então, diversas variações do DES com menos iterações foram sendo quebradas. Uma variação com três iterações foi quebrada em 1982 e seis iterações algum tempo depois. Em 1990 Biham e Shamir apresentam a criptanálise diferencial e mostram que qualquer variação do DES com menos de 16 iterações pode ser quebrada por um ataque de texto conhecido mais eficientemente que pelo uso de força bruta (busca exaustiva de chaves). Desde então especula-se que a NSA já conhecia este ataque nos anos 1970 e que o número de iterações do DES não foi escolhido de forma aleatória, mas sim para evitar o uso de criptanálise diferencial.

1.3.3.7. *As caixas de substituição.* Com exceção das caixas de substituição usadas na função f, todas as operações envolvidas no DES são lineares com relação à entrada. Isto significa que calcular um ou-exclusivo de duas saídas é equivalente a fazer um ou-exclusivo das duas entradas e então calcular o resultado. Esta certamente é uma caraterística indesejável num criptossistema.

Como as caixas de substituição são as únicas operações do DES não lineares, a segurança destas operações é vital ao algoritmo. Mas não se

[5] O efeito avalanche também implica em pequenas alterações na entrada levarem a grandes alterações na saída. A alteração de um único bit da entrada apresentada ao DES em média afeta mais da metade dos bits de saída.

sabe quais os critérios usados na construção das caixas de substituição do DES, o que impede uma análise mais ampla de suas características e de sua segurança criptográfica.

Muitas críticas ao DES são devidas ao segredo que envolve as caixas de substituição e à possibilidade de que tenham sido preparadas *trap doors* nestas operações. Em 1976, a NSA divulgou uma lista parcial de critérios usados na construção destas caixas de substituição, mas não se sabe se esta lista inclui todos os critérios usados durante a elaboração do algoritmo.

1.4. International Data Encryption Algorithm

O *International Data Encryption Algorithm* (IDEA) foi proposto por Lai e Massey em 1990 sob o nome PES (*Proposed Encryption Standard*) e revisto em 1991, após a publicação da criptanálise diferencial. Este algoritmo tem forte embasamento teórico e é considerado bastante seguro, sendo usado em diversos produtos comerciais. Este algoritmo usa blocos de 64 bits e chaves de 128 bits, o que garante boa resistência a ataques por busca exaustiva.

Quando o IDEA foi publicado pela primeira vez, em 1990, Shamir e Biham ainda não haviam publicado a criptanálise diferencial, o que ocorreria ainda no mesmo ano. Os autores do IDEA usaram então o conceito de criptanálise diferencial para melhorar a segurança de seu algoritmo, introduzindo inclusive o conceito de **Cifras de Markov** que permite classificar cifras de acordo com a sua capacidade de resistir à criptanálise diferencial. Esta segunda versão da cifra ficou inicialmente conhecida como IPES (*Improved PES*) e mais tarde como IDEA.

Dois princípios básicos nortearam o projeto do IDEA:

(1) A implementação deveria ser eficiente tanto em *software* quanto em *hardware;*
(2) A cifra iria "misturar operações de diferentes grupos algébricos com o mesmo número de elementos."

A implementação em *software* é facilitada pelo uso de sub-blocos de 16 bits (cada bloco é dividido em 4 sub-blocos de 16 bits), que são "naturais" em implementações por software e pelo uso de operações simples sobre estes blocos, como ou-exclusivo e soma. O IDEA também faz uso de operações de multiplicação, que podem ser implementadas a partir de primitivas simples, como mostraremos na Seção 1.4.2. Para facilitar a implementação em *hardware*, a estrutura da cifra é regular, composta da execução de iterações de um mesmo bloco de operações, e as operações de ciframento e deciframento são similares, baseadas no uso da mesma estrutura com sub-chaves diferentes.

Para conseguir a propriedade de **confusão**, o IDEA mistura operações de três grupos algébricos diferentes mas de mesma cardinalidade cujas operações são incompatíveis. Os grupos usados no IDEA são:

(1) ou-exclusivo (xor) bit a bit de dois blocos de 16 bits, denotado $\oplus$.
(2) adição de inteiros módulo 2^{16} onde cada sub-bloco é tratado como um inteiro sem sinal. Esta operação é denotada $\boxplus$.
(3) Multiplicação de inteiros módulo $2^{16}+1$ onde os sub-blocos são tratados como inteiros sem sinal, exceto pelo zero, que irá representar 2^{16}. Esta operação será denotada $\odot$.

Estas operações são incompatíveis, no sentido que:

(1) Se tomadas duas a duas, estas operações nunca são distributivas. Por exemplo:

$$a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c).$$

(2) Se tomadas duas a duas, estas operações não são associativas. Por exemplo:

$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c.$$

No arranjo proporcionado pelo processo de ciframento (Figura 1.4.2) o resultado de uma operação nunca é usado como entrada para uma operação do mesmo tipo. Esta combinação das diferentes operações proporciona uma transformação complexa da entrada, aumentando a dificuldade de criptanálise da cifra.

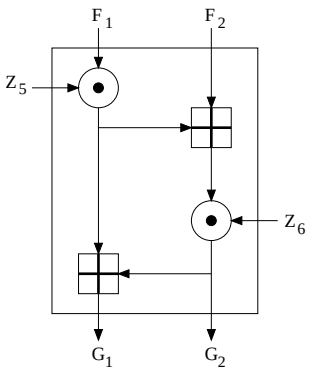

FIGURA 1.4.1. A estrutura de multiplicação/adição (MA)

Para conseguir satisfazer o critério de **difusão**, a cifra faz uso da estrutura de multiplicação/adição (MA), mostrada na Figura 1.4.1. Esta estrutura transforma dois blocos F_1 e F_2 de entrada de 16 bits em dois blocos G_1 e G_2 de saída de 16 bits usando duas sub-chaves Z_5 e Z_6 de 16 bits como controle. Esta estrutura tem como propriedades:

- qualquer que seja a escolha das sub-chaves Z_5 e Z_6, $MA(\cdot,\cdot,Z_5,Z_6)$ é uma transformação inversível. O mesmo vale para as escolhas de F_1 e F_2.
- esta estrutura tem "difusão completa", no sentido em que cada bit da saída depende de todo bit da entrada.

- esta estrutura usa o número mínimo de operações necessário para conseguir tal "difusão completa".

1.4.1. Descrição do Algoritmo. O algoritmo do IDEA consiste de 8 iterações e uma transformação de saída. A entrada é dividida em quatro sub-blocos de 16 bits e cada iteração usa quatro blocos de 16 bits como entrada e gera quatro blocos de 16 bits como saída. A transformação final também gera quatro blocos de 16 bits como saída e estes são concatenados para formar o texto cifrado de 64 bits. Cada iteração usa seis sub-chaves e a transformação final usa quatro sub-chaves num total de 52 sub-chaves de 16 bits, geradas a partir da chave original de 128 bits.

1.4.1.1. *Funcionamento de cada iteração.* Como todas iterações no IDEA tem a mesma estrutura, iremos descrever apenas o funcionamento da primeira iteração. O funcionamento das demais difere apenas no uso das sub-chaves. A figura 1.4.2 mostra o funcionamento desta cifra, onde podemos observar que o IDEA não segue a estrutura das cifras de Feistel.

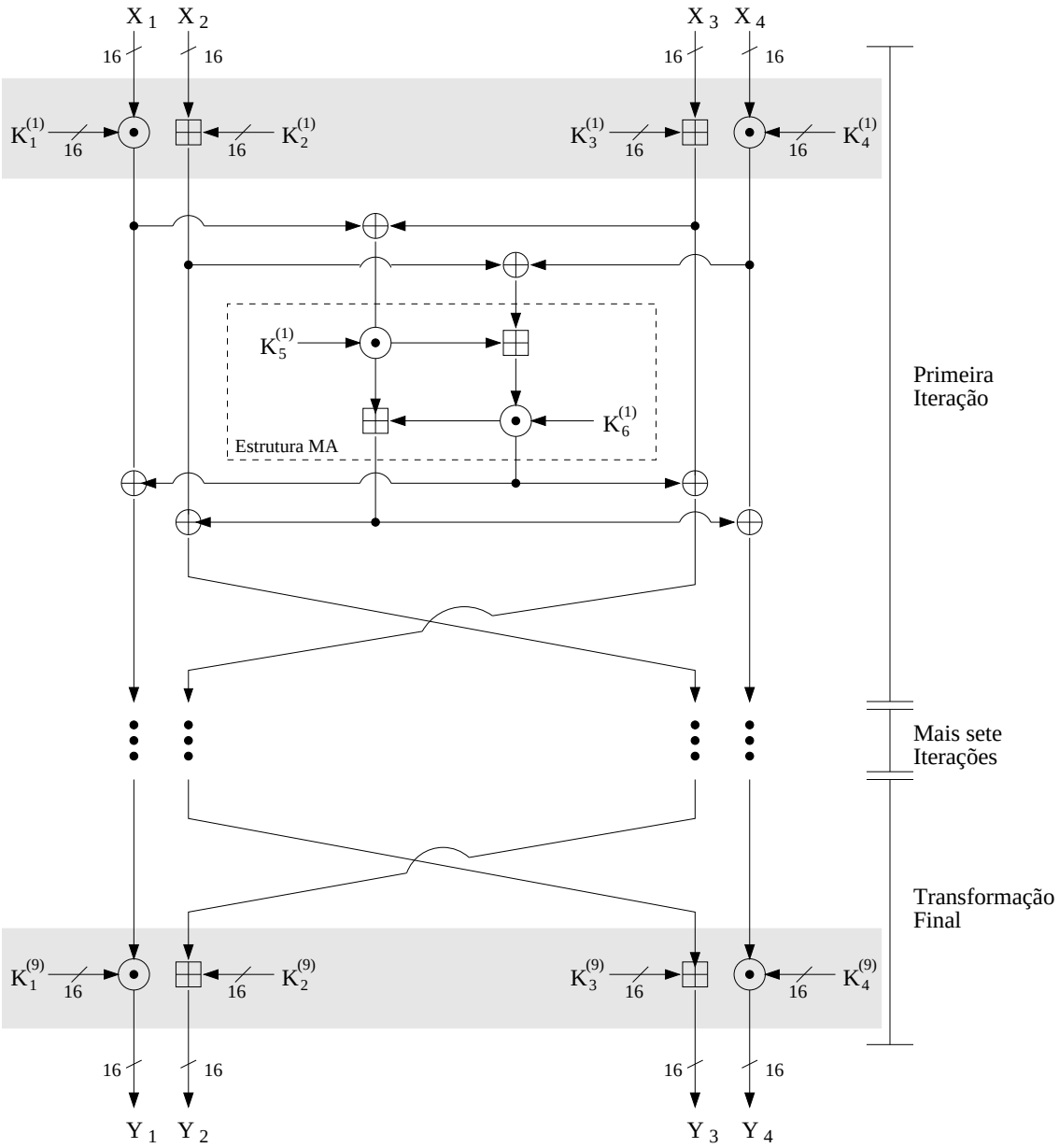

FIGURA 1.4.2. O diagrama de computação do IDEA

Cada iteração começa com uma transformação que combina os quatro blocos de entrada com quatro sub-chaves, usando operações de adição e multiplicação. Esta transformação está destacada na parte superior da figura 1.4.2. Os blocos resultantes desta transformação são então combinados dois a dois por operações de ou-exclusivo e usados como entrada da estrutura MA, mostrada na figura como um retângulo pontilhado. A estrutura MA usa estas entradas e duas sub-chaves para gerar dois blocos de 16 bits.

Em seguida, os blocos gerados pela estrutura MA são combinados por operações de ou-exclusivo com os quatro blocos produzidos pela transformação inicial para produzir quatro blocos de 16 bits, que serão o resultado desta iteração. O segundo e o terceiro blocos da saída tem sua ordem invertida para aumentar a "mistura" dos bits e melhorar a resistência da cifra a criptanálise.

A última iteração do algoritmo é chamada de transformação final está destacada na parte inferior da figura 1.4.2. É fácil perceber que esta estrutura é idêntica à transformação inicial de cada iteração (em destaque na parte superior da figura). A diferença consiste em inverter a ordem do segundo e terceiro blocos da entrada antes de aplicar a transformação. Isto permite que a seqüência de operações do deciframento seja idêntica à do ciframento. Esta nona iteração necessita apenas de quatro sub-chaves, e não seis como todas as outra iterações.

1.4.1.2. *Geração das Sub-chaves.* Para o funcionamento da cifra, é necessário gerar 52 sub-chaves de 16 bits a partir da chave original de 128 bits. As sub-chaves serão denotadas por $Z_j^{(i)}$ onde i indica a iteração em que a chave será usada, podendo ser $1, 2, \ldots, 9$ e j indica a ordem em que a chave será usada em cada iteração, com $1 \leq j \leq 6$ exceto na última iteração onde são usadas apenas quatro chaves. Por exemplo, as chaves usadas na primeira iteração são $Z_1^{(1)}, Z_2^{(1)}, Z_3^{(1)}, Z_4^{(1)}, Z_5^{(1)}, Z_6^{(1)}$ e as chaves usadas na última iteração são $Z_1^{(9)}, Z_2^{(9)}, Z_3^{(9)}, Z_4^{(9)}$.

A chave original é particionada em 8 sub-blocos que serão usados diretamente como as primeiras oito sub-chaves (6 para a primeira iteração e duas para a segunda iteração).

A chave original sofre então uma rotação para a esquerda de 25 posições e é novamente particionada em oito sub-blocos que serão usados como as próximas oito sub-chaves. O bloco de 128 bits obtido pela rotação sofre nova rotação de 25 posições para a esquerda e é novamente particionado em 8 sub-chaves. Este processo se repete 7 vezes até que todas as sub-chaves necessárias tenham sido geradas.

Este processo de geração de sub-chaves apresenta duas propriedades interessantes:

- a primeira sub-chave usada em cada iteração usa um subconjunto diferente dos bits da chave original.
- não há uma relação simples de deslocamento entre as sub-chaves usadas em duas iterações do algoritmo. Isto se deve ao fato de que cada iteração não necessita dos 128 bits da chave original e do deslocamento de 25 posições a cada iteração da geração de chaves.

1.4.2. Deciframento. O processo de deciframento no IDEA é essencialmente igual ao ciframento, com alterações na geração das sub-chaves. Para gerar as sub-chaves, primeiro calculamos todas as chaves

de ciframento $K_i^{(r)}$ e, a partir destas, calculamos as chaves de deciframento $K_i'^{(r)}$ com base na tabela 2. De posse de todas as chaves de deciframento, podemos usar o texto cifrado como entrada para o algoritmo e usamos a chave $K_i'^{(r)}$ no lugar de $K_i^{(r)}$.

iteração r	$K_1'^{(r)}$	$K_2'^{(r)}$	$K_3'^{(r)}$	$K_4'^{(r)}$	$K_5'^{(r)}$	$K_6'^{(r)}$
$r = 1$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$2 \leq r \leq 8$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$r = 9$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	-	-

TABELA 2. As chaves usadas no processo de deciframento.

Para obter algumas das chaves de deciframento, pode ser necessário calcular o inverso aditivo $-K \pmod{2^{16}}$ ou o inverso multiplicativo $K^{-1} \pmod{2^{16}+1}$ de alguma chave K. O inverso aditivo u de uma chave K pode ser calculado como

$$u = \left(2^{16} - K\right) \textbf{ and } \text{0xFFFF}, \;\; 0 \leq u \leq 2^{16} - 1.$$

O inverso multiplicativo pode ser calculado pelo Algoritmo Estendido de Euclides, que recebe dois valores de entrada $a \geq b \geq 0$ e retorna dois inteiros x e y tais que $ax + by = \gcd(a, b)$. Se usarmos $a = 2^{16} + 1$ e $b = K$, $\gcd(a, b)$ será sempre 1, (exceto quando $K = 0$) e então $K^{-1} = y$ ou $2^{16} + 1 + y$ se $y < 0$. Quando $K = 0$, então K representa 2^{16} e sabemos que $(2^{16})^{-1} \pmod{2^{16}+1} = 2^{16}$, ou seja, quando $K = 0$, $K^{-1} = 0$.

Para verificar que o mesmo algoritmo usado com as chaves definidas acima é capaz de decifrar corretamente uma mensagem, vamos usar a Figura 1.4.3, que mostra o processo de ciframento de baixo para cima no lado esquerdo e o processo de deciframento de cima para baixo no lado direito. Na figura, cada um das oito primeiras iterações foi quebrada em duas partes: uma a primeira chamada transformação e a segunda chamada de sub-iteração. A transformação corresponde ao bloco destacado na parte superior da Figura 1.4.2 e a sub-iteração ao restante do processamento da mesma iteração.

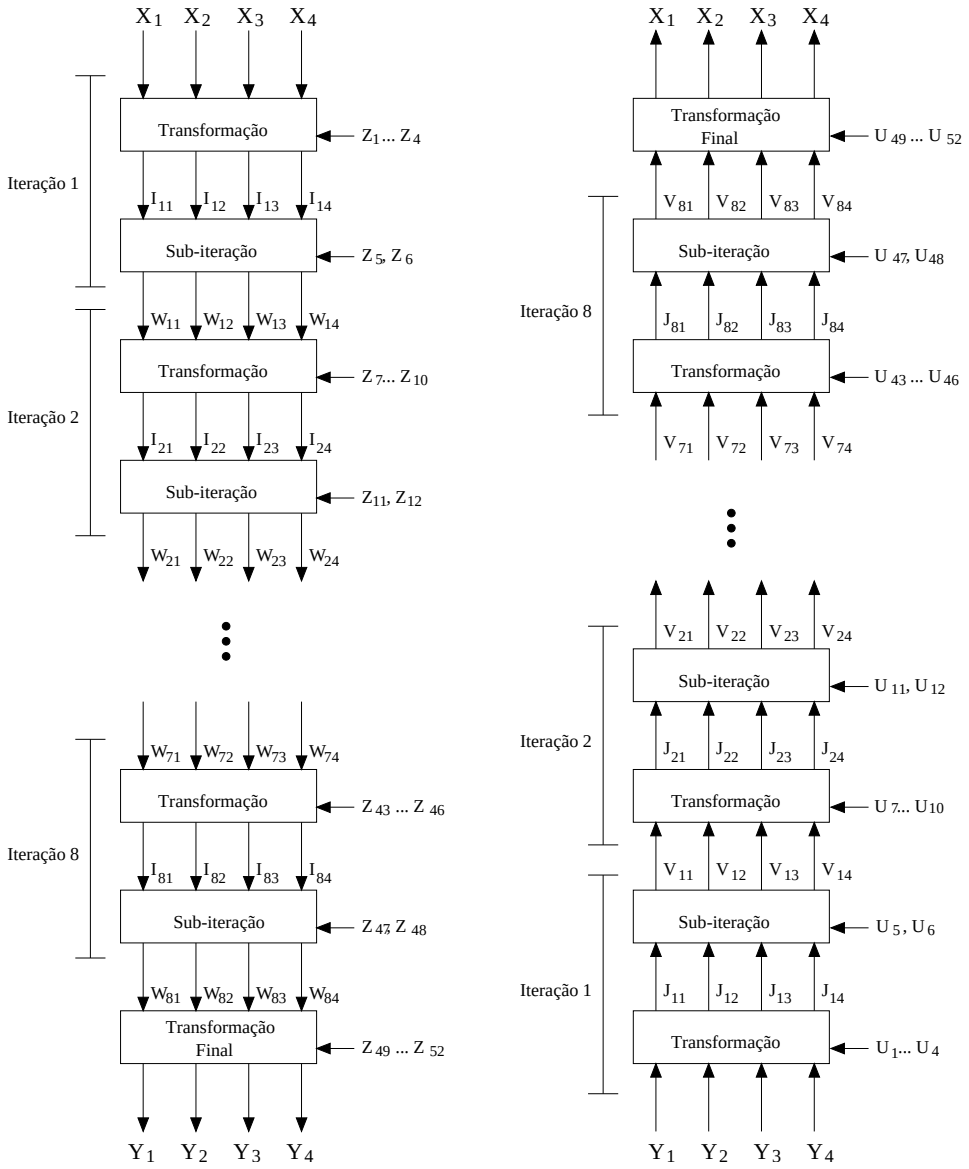

FIGURA 1.4.3. Ciframento de deciframento no IDEA.

Considerando as caixas na parte inferior dos dois diagramas mostrados na Figura 1.4.3, podemos concluir que, no ciframento :

$$Y_1 = W_{81} \odot Z_{49} \qquad Y_3 = W_{82} \boxplus Z_{51}$$
$$Y_2 = W_{83} \boxplus Z_{50} \qquad Y_4 = W_{84} \odot Z_{52}$$

E o primeiro estagio da primeira iteração do processo de deciframento nos dá:

$$J_{11} = Y_1 \odot U_1 \qquad J_{13} = Y_3 \boxplus U_3$$
$$J_{12} = Y_2 \boxplus U_2 \qquad J_{14} = Y_4 \odot U_4$$

Que substituindo, obtemos:

$$\begin{aligned}
J_{11} &= Y_1 \odot Z_{49}^{-1} = W_{81} \odot Z_{49} \odot Z_{49}^{-1} = W_{81} \\
J_{12} &= Y_2 \boxplus -Z_{50} = W_{83} \boxplus Z_{50} \boxplus -Z_{50} = W_{83} \\
J_{13} &= Y_3 \boxplus -Z_{51} = W_{82} \boxplus Z_{51} \boxplus -Z_{51} = W_{82} \\
J_{14} &= Y_4 \odot Z_{52}^{-1} = W_{84} \odot Z_{52} \odot Z_{52}^{-1} = W_{84}
\end{aligned}$$

O que nos indica que o resultado do primeiro estágio do deciframento é igual à entrada usada no último estágio do ciframento, exceto pela inversão do segundo bloco com o terceiro. Consideremos agora as seguintes equações, derivadas da Figura 1.4.3:

$$\begin{aligned}
W_{81} &= I_{81} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \\
W_{82} &= I_{83} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \\
W_{83} &= I_{82} \oplus MA_E\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \\
W_{84} &= I_{84} \oplus MA_E\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right)
\end{aligned}$$

onde $MA_D(X, Y)$ é a saída mostrada no lado direito da estrutura MA da figura 1.4.1, usando entradas X e Y, e $MA_E(X, Y)$ é a saída mostrada no lado esquerdo correspondente. Então temos:

$$\begin{aligned}
V_{11} &= J_{11} \oplus MA_D\left(J_{11} \oplus J_{13}, J_{12} \oplus J_{14}\right) \\
&= W_{81} \oplus MA_D\left(W_{81} \oplus W_{82}, W_{83} \oplus W_{84}\right) \\
&= I_{81} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \oplus \\
&\quad MA_D[I_{81} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \oplus I_{83} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right), \\
&\qquad I_{82} \oplus MA_E\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \oplus I_{84} \oplus MA_E\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right)] \\
&= I_{81} \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \oplus MA_D\left(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}\right) \\
&= I_{81}
\end{aligned}$$

De maneira similar, temos que

$$\begin{aligned}
V_{12} &= I_{83} \\
V_{13} &= I_{82} \\
V_{14} &= I_{84}
\end{aligned}$$

O que nos indica que a saída do segundo estágio do processo de deciframento é igual à entrada usada no penúltimo estágio do processo de ciframento, com o segundo e terceiro blocos invertidos. Usando o mesmo raciocínio, podemos chegar a

$$\begin{aligned}
V_{81} &= I_{11} \\
V_{82} &= I_{13} \\
V_{83} &= I_{12} \\
V_{84} &= I_{14}
\end{aligned}$$

Finalmente, como a transformação final da cifra inverte os blocos na segunda e terceira posições, temos que a saída correspondente a todo o processo de deciframento é igual à entrada usada no processo de ciframento.

1.4.3. Propriedades. Para as oito iterações do IDEA, nenhum ataque publicado é mais eficiente que uma busca exaustiva no espaço de chaves, embora tenham sido detectadas chaves fracas. A principal fraqueza do IDEA parece ser o tamanho reduzido dos blocos de dados usados no algoritmo.

1.4.3.1. *Características criptográficas.* As seguintes características do IDEA estão relacionadas com a dificuldade em atacá-lo:

Tamanho do bloco: o tamanho dos blocos de uma cifra deve ser suficiente para dificultar análises estatísticas, mas a complexidade de implementação parece crescer com a tamanho do bloco. O tamanho de 64 bits escolhido para o IDEA é ainda suficiente, embora a diferença entre o tamanho dos blocos e o tamanho das chaves do IDEA possa ser usada em ataques por busca exaustiva. Neste caso, uma busca no espaço de blocos envolve 2^{64} possibilidades contra 2^{128} possibilidades de chaves.

Tamanho da chave: o tamanho das chaves deve ser suficiente para evitar busca exaustiva. Os 128 bits usado no IDEA são suficientes hoje e devem continuar suficientes ainda por bastante tempo. Supondo que pudéssemos construir uma máquina capaz de testar um bilhão de chaves diferentes por segundo (o que é impossível atualmente) e usássemos um bilhão destas máquinas em paralelo, seriam necessário aproximadamente 10^{13} segundos para testar todas as chaves possíveis, o que é mais do que a idade do universo.

Confusão: o princípio de confusão afirma que a relação entre o texto cifrado e o texto em claro deve ser complicada, objetivando dificultar a determinação de relações estatísticas entre o texto cifrado e o texto em claro. No IDEA, este princípio é alcançado pela mistura das três operações básicas mostradas acima (adição, multiplicação e xor).

Difusão: cada bit da entrada deve influenciar cada um dos bits da saída, assim como cada bit da chave também deve ter influência sobre todos os bits da saída. Os autores do IDEA realizaram diversos experimentos e concluíram que o IDEA atinge este objetivo. A estrutura MA, mostrada acima, é responsável pela boa característica de difusão do IDEA.

1.4.3.2. *Cifras de Markov.* Biham e Shamir publicaram os princípios da criptanálise diferencial logo depois da publicação da primeira versão do IDEA. Os autores do IDEA aplicaram a criptanálise diferencial em sua própria cifra e foram capazes de definir uma nova classe de cifras, chamadas de **cifras de Markov**. Usando o modelo de cifras de Markov, eles puderam também mostrar que é possível quantificar a resistência destas cifras à criptanálise diferencial.

No mesmo artigo, Lai, Massey e Murphy mostram que o IDEA é uma cifra de Markov e indicam algumas alterações que tornam a cifra

mais resistente contra a criptanálise diferencial. Esta cifra alterada, inicialmente denominada IPES, viria mais tarde a ser conhecida como IDEA. Este artigo afirma também que o DES é uma cifra de Markov, embora não proponha alterações no DES com base neste fato.

1.4.3.3. *Chaves fracas.* Em 1993, Daemen, Govaerts e Vandewalle publicam a descoberta de uma classe de chaves fracas para o IDEA. O conceito de chaves fracas usado não é exatamente o mesmo que para o DES, onde as chaves fracas geram funções de ciframento que são suas próprias inversas. Se as chaves fracas do IDEA forem usadas, é possível identificá-las por um ataque do texto conhecido. Ou seja, se for possível obter o resultado do ciframento de blocos de entrada escolhidos criteriosamente, é possível detectar que uma determinada chave fraca foi usada.

As chaves fracas do IDEA tem a forma (em hexadecimal):

$$0000\,0000\,0x00\,0000\,0000\,000x\,xxxx\,x000,$$

onde x pode ser qualquer número. A probabilidade de geração de tal chave aleatoriamente é de 2^{-96}, uma chance bastante reduzida. E ainda é possível modificar o IDEA para eliminar a existência de chaves fracas: basta aplicar um ou-exclusivo de cada sub-chave com o valor 0x0dea (em hexadecimal) antes de usar as sub-chaves.

1.4.3.4. *Variações.* A maneira mais simples de obter variantes do IDEA é alterar o número de iterações. Como ainda não se conhece nenhum ataque eficiente contra as oito iterações da cifra completa, as variantes normalmente envolvem a redução do número de iterações. Para a variante de até 3 iterações mais a transformação de saída, existem ataques mais eficientes que força bruta. Estes ataques são variações das criptanálises diferencial e linear, mas não parece ser possível estendê-los ao IDEA completo com 8 iterações.

Outra variação possível é aumentar ou diminuir o tamanho dos blocos. Na verdade, varia-se o tamanho dos sub-blocos usados internamente no algoritmo, o que tem o efeito de alterar o tamanho dos blocos vistos externamente. A diminuição do tamanho dos blocos é possível e foi proposta e analisada pelos autores da cifra. Já o aumento do tamanho dos sub-blocos não é possível, já que parte do algoritmo depende de propriedades relacionadas ao tamanho dos grupos usados em suas operações internas. Em especial, o número de elementos no grupo multiplicativo deve ser primo para que haja apenas um inverso para cada um de seus elementos. Sabemos que $2^{16} + 1$ é primo, o que nos permite usar sub-blocos de 16 bits, mas $2^{32} + 1$ não é primo, o que impede o uso de sub-blocos de 32 bits.

Uma terceira variação do algoritmo seria o uso de sub-chaves independentes, ou seja, seria necessária uma chave com 832 bits para que dela fossem extraídas 52 sub-chaves de 16 bits. Esta variante é mais

segura que o algoritmo original, embora não seja muito prático lidar com chaves deste tamanho.

1.4.3.5. *Velocidade e implementações.* Em *software,* as implementações do IDEA são aproximadamente duas vezes mais lentas que implementações do DES, em máquinas Pentium. Em hardware, a velocidade máxima do IDEA é bem próxima das mais velozes implementações do DES.

A operação mais lenta usada no IDEA é a multiplicação, que normalmente leva vários ciclos para ser concluída na maior parte das arquiteturas de processadores. No entanto, o aumento do uso de multimídia e a conseqüente necessidade de execução rápida de instruções contendo multiplicações tem levado os projetistas de microprocessadores a otimizar estas operações. Assim, em algumas arquiteturas, a operação mais lenta usada no IDEA pode ser realizada muito rapidamente, permitindo um ganho expressivo de velocidade na execução total do algoritmo. Lipmaa relata ganhos de até quatro vezes pelo uso das instruções MMX, que é o conjunto de instruções especiais para multimídia da família Pentium de processadores. Lipmaa afirma ter conseguido uma implementação capaz de cifrar a 78 Mbits/s num Pentium MMX de 166 Mz, contra cerca de 18 Mbits/s conseguido em um Pentium de 166Mz sem o conjunto de instruções MMX.