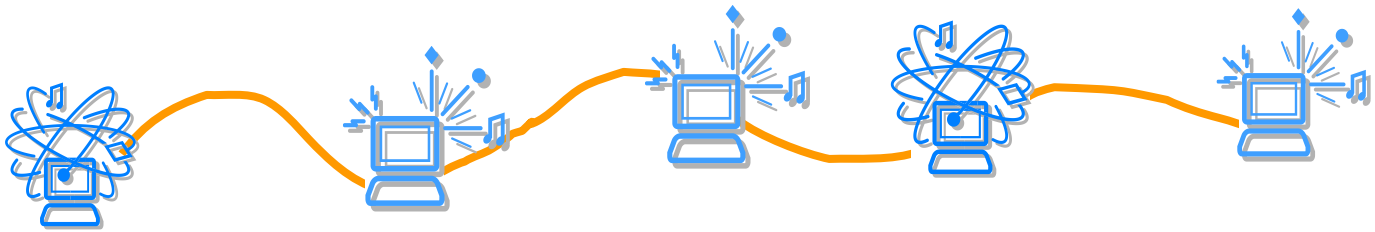
Segurança em Redes
Cifra simétrica



Redes de Comunicação de Dados

Área Departamental de Engenharia da Eletrónica e das Telecomunicações e de Computadores

Instituto Superior de Engenharia de Lisboa

Criptografia

Os sistemas criptográficos costumam ser classificados de acordo com os seguintes pontos:

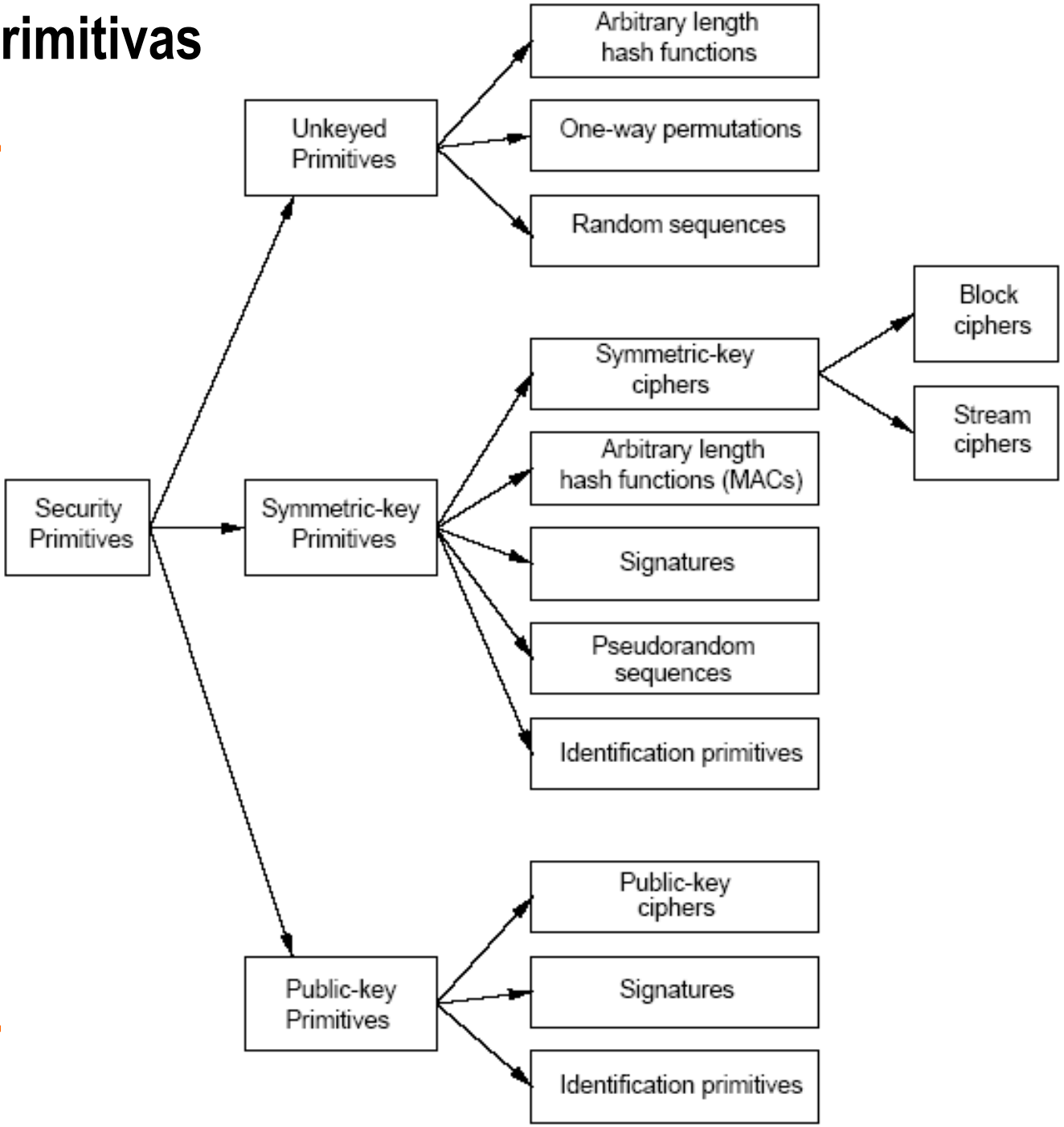
1. **Tipo de operações usadas** para transformar o texto em claro no texto cifrado
   - Substituição e transposição ou permutação
2. **Número de chaves utilizadas**
   - Uma chave – cifra simétrica, chave secreta ou convencional
   - Duas chaves – cifra assimétrica ou de chave pública
3. A **forma como o texto em claro é processado**
   - Cifra em bloco (*block cipher*)
   - Cifra sequencial, *fluxo ou continua* (*stream cipher*)

Segurança dos algoritmos de cifra

- ***Incondicionalmente seguro*** *(Unconditionally secure)*
  - Se, independentemente da quantidade de texto cifrado que o criptoanalista tiver, nunca existir informação suficiente para recuperar o texto cifrado

- **Computacionalmente seguro** (*computationally secure*)
  - Um esquema de encriptação diz-se computacionalmente seguro se o texto cifrado estiver de acordo com um ou ambos os critérios seguintes:
    - O custo de quebrar a cifra excede o valor da informação cifrada
    - O tempo necessário à quebra da cifra excede o tempo de vida útil da informação

Taxonomia das primitivas criptográficas



Algoritmos de cifra convencionais

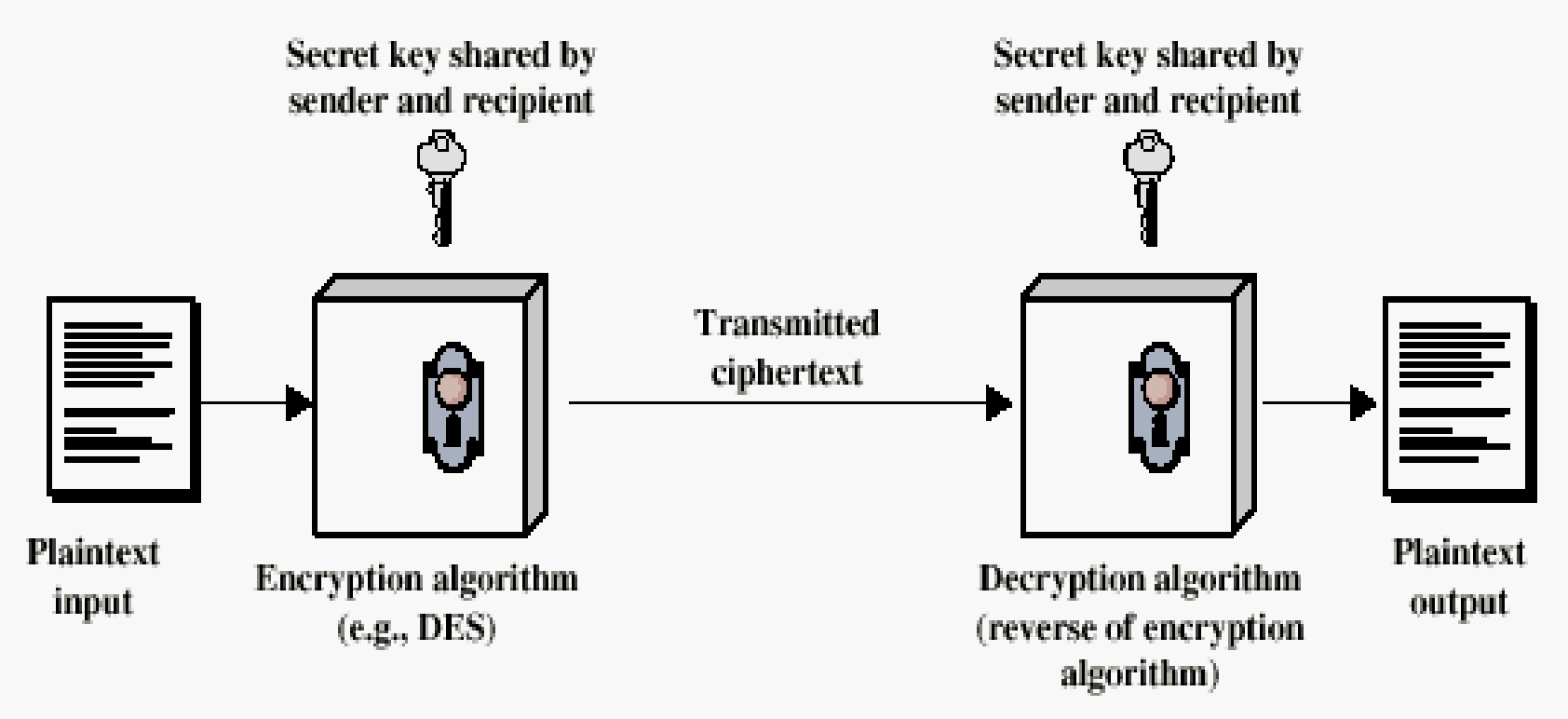
Algorithm	Key Size (bits)	Block Size (bits)	Number of Rounds	Applications
DES	56	64	16	SET, Kerberos
Triple DES	112 or 168	64	48	Financial key management, PGP, S/MIME
AES	128, 192, or 256	128	10, 12, or 14	Intended to replace DES and 3DES
IDEA	128	64	8	PGP
Blowfish	variable to 448	64	16	Various software packages
RC5	variable to 2048	64	variable to 255	Various software packages

Princípios de cifra convencional

- Um esquema de cifra tem 5 ingredientes:
  - Texto em claro
  - Algoritmo de cifrar
  - Chave secreta
  - Texto cifrado
  - Algoritmo de decifrar

- A segurança depende do segredo da chave, não no segredo do algoritmo. Este deve poder ser do conhecimento público.

Princípios de cifra convencional



Modelo convencional de cifra convencional

Definição – Cifra simétrica

- Considere um esquema de cifra consistindo nos conjuntos de transformações de cifra e decifra $\{E_e : e \in K\}$ e $\{D_d : d \in K\}$ respectivamente, onde *K* é o espaço das chaves. O esquema diz-se de chave simétrica se, para cada par {*e, d*} de cifra/decifra for computacionalmente "fácil" determinar *d* conhecendo-se apenas *e* e também determinar *e* a partir de *d*. Dado que *e*=*d* na maioria dos esquemas de cifra simétrica o termo chave simétrica parece apropriado.

  Outros termos utilizados são **chave única** (*single-key*), **chave secreta**, **chave única** (*one-key*), **chave privada** (*private key*) e **cifra convencional**.

Mais definições

- Cifra em bloco ("***Block cipher***")
  - Os dados são separados em blocos de dimensão fixa e cifrados um bloco de cada vez

- Cifra em fluxo ("***Stream cipher***")
  - Os dados são cifrados um bit de cada vez, conforme vão sendo apresentados ao dispositivo de cifragem

- A maioria dos algoritmos utilizados actualmente implementam cifra em bloco.

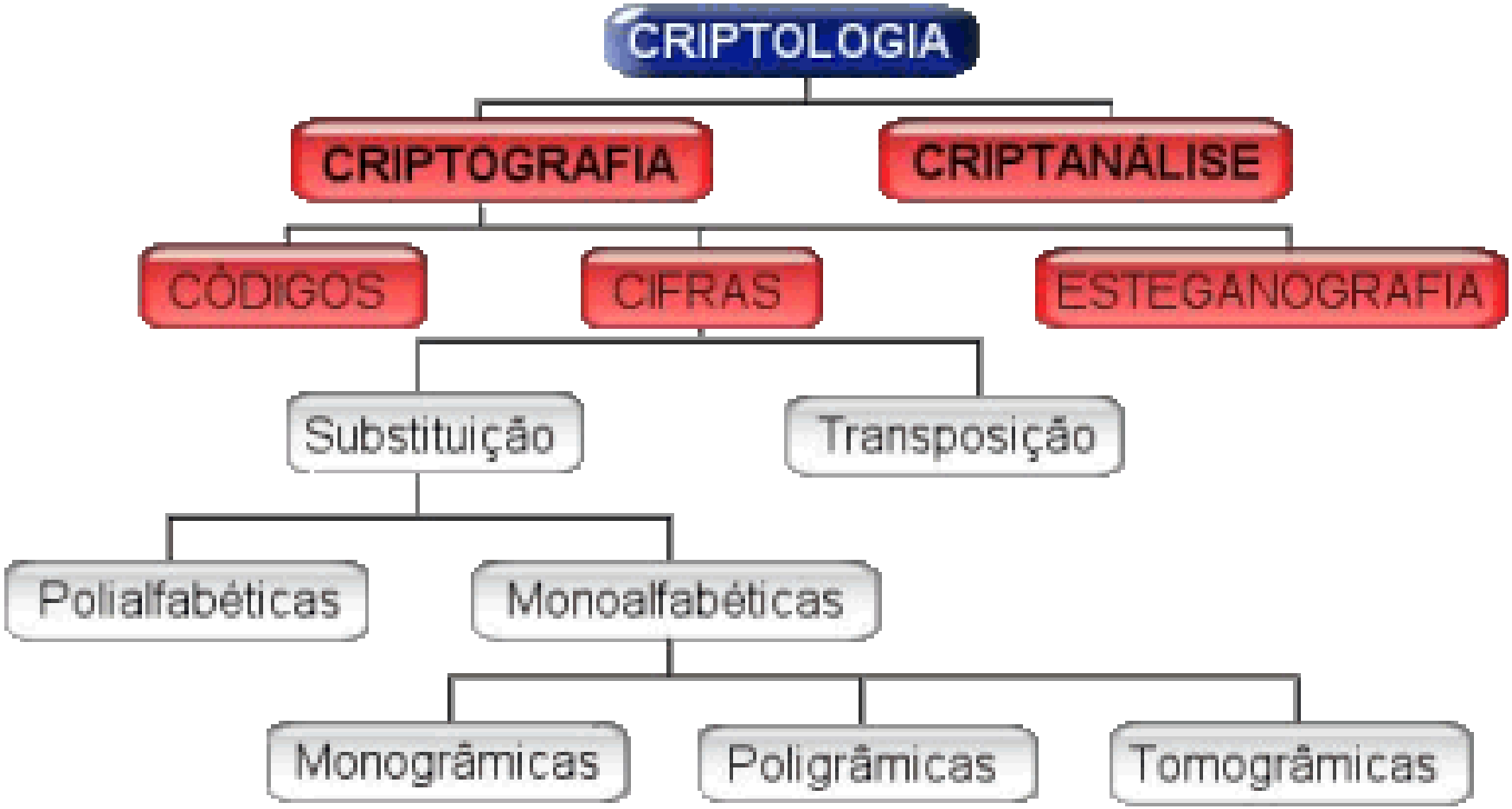
Cifra em bloco

- **Definição**: Uma cifra em bloco (*block cipher*) é um esquema de encriptação que parte as mensagens de texto em claro, a serem transmitidas, em conjuntos mais pequenos (designados por blocos) de dimensão fixa *t* sobre um alfabeto *A*, e encripta um bloco de cada vez.

- Duas importantes classes de cifra em bloco são as:
  - **cifras por substituição,** e as
  - **cifras por transposição**.

As **cifras de produto** combinam as anteriores.

Organograma



[http://www.numaboa.com.br/criptologia/intro.php]

Cifras de substituição simples

Simple substitution ciphers

Definition Let $\mathcal{A}$ be an alphabet of q symbols and $\mathcal{M}$ be the set of all strings of length t over $\mathcal{A}$. Let $\mathcal{K}$ be the set of all permutations on the set $\mathcal{A}$. Define for each $e \in \mathcal{K}$ an encryption transformation E_e as:

$$E_e(m) = (e(m_1)e(m_2)\cdots e(m_t)) = (c_1c_2\cdots c_t) = c,$$

where $m = (m_1m_2\cdots m_t) \in \mathcal{M}$. In other words, for each symbol in a t-tuple, replace (substitute) it by another symbol from $\mathcal{A}$ according to some fixed permutation e. To decrypt $c = (c_1c_2\cdots c_t)$ compute the inverse permutation $d = e^{-1}$ and

$$D_d(c) = (d(c_1)d(c_2)\cdots d(c_t)) = (m_1m_2\cdots m_t) = m.$$

E_e is called a *simple substitution cipher* or a *mono-alphabetic substitution cipher*.

Exemplo de cifra simétrica

- Seja *A*={a,b,c,...,x,y,z} o alfabeto inglês. Seja *M* e *C* o conjunto de todas as palavras de dimensão cinco sobre *A*. A chave *e* é escolhida para ser uma permutação de *A*. Para cifrar uma mensagem em inglês esta é dividida em grupos de cinco letras (com o enchimento apropriado se a mensagem não for múltipla de cinco). E a permutação *e* é aplicada a cada letra, uma de cada vez. Para decifrar é aplicada a permutação inversa $d=e^{-1}$ a cada letra do texto cifrado.

$$e = \begin{pmatrix} \mathrm{A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z} \\ \mathrm{D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z\ A\ B\ C} \end{pmatrix}$$

Mensagem de texto em claro *m:*

$$m = \mathrm{THISC\ IPHER\ ISCER\ TAINL\ YNOTS\ ECURE}$$

Mensagem de texto cifrado *c:*

$$c = E_e(m) = \mathrm{WKLVF\ LSKHU\ LVFHU\ WDLQO\ BQRWV\ HFXUH}$$

Cifras por substituição polialfabéticas

Polyalphabetic substitution ciphers

Definition A *polyalphabetic substitution cipher* is a block cipher with block length t over an alphabet $\mathcal{A}$ having the following properties:

(i) the key space $\mathcal{K}$ consists of all ordered sets of t permutations $(p_1, p_2, \ldots, p_t)$, where each permutation p_i is defined on the set $\mathcal{A}$;

(ii) encryption of the message $m = (m_1 m_2 \cdots m_t)$ under the key $e = (p_1, p_2, \ldots, p_t)$ is given by $E_e(m) = (p_1(m_1)p_2(m_2)\cdots p_t(m_t))$; and

(iii) the decryption key associated with $e = (p_1, p_2, \ldots, p_t)$ is $d = (p_1^{-1}, p_2^{-1}, \ldots, p_t^{-1})$.

Cifras por substituição polialfabéticas

Example (*Vigenère cipher*) Let $A = \{A, B, C, \ldots, X, Y, Z\}$ and $t = 3$. Choose $e = (p_1, p_2, p_3)$, where p_1 maps each letter to the letter three positions to its right in the alphabet, p_2 to the one seven positions to its right, and p_3 ten positions to its right. If

$$m = \text{THI SCI PHE RIS CER TAI NLY NOT SEC URE}$$

then

$$c = E_e(m) = \text{WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO.}$$ □

Esta cifra tem a vantagem de não manter a frequência dos caracteres.

Tabela de Vigenère

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Cifra por transposição

Definition Consider a symmetric-key block encryption scheme with block length t. Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, \ldots, t\}$. For each $e \in \mathcal{K}$ define the encryption function

$$E_e(m) = (m_{e(1)} m_{e(2)} \cdots m_{e(t)})$$

where $m = (m_1 m_2 \cdots m_t) \in \mathcal{M}$, the message space. The set of all such transformations is called a *simple transposition cipher*. The decryption key corresponding to e is the inverse permutation $d = e^{-1}$. To decrypt $c = (c_1 c_2 \cdots c_t)$, compute $D_d(c) = (c_{d(1)} c_{d(2)} \cdots c_{d(t)})$.

A simple transposition cipher preserves the number of symbols of a given type within a block, and thus is easily cryptanalyzed.

Confusão e difusão

- Uma **substituição** num "*round*" diz-se adicionar **confusão** ao processo de encriptação enquanto a **transposição** adiciona **difusão**.
- A **confusão** pretende tornar a relação entre a chave e o texto cifrado tão complexa quanto possível.
- A **difusão** refere-se ao rearranjo ou espalhamento dos bits da mensagem de maneira a que qualquer redundância no texto em claro se espalhe pelo texto cifrado. A estrutura estatística do texto em claro é dissipada.
- Um "*round*" pode então espalhar confusão e difusão na encriptação.

- Os sistemas de cifra em bloco aplicam uma série de "*rounds*" em sucessão para cifrarem o texto em claro.

Diffusion and Confusion

- Diffusion
  - Statistical nature of plaintext is reduced in ciphertext
  - E.g. A plaintext letter affects the value of many ciphertext letters
  - How: repeatedly apply permutation (transposition) to data, and then apply function
- Confusion
  - Make relationship between ciphertext and key as complex as possible
  - Even if attacker can find some statistical characteristics of ciphertext, still hard to find key
  - How: apply complex (non-linear) substitution algorithm

Cifra de fluxo/sequencial (*stream cipher*)

- As ***stream ciphers*** formam uma classe importante dos esquemas de encriptação por chave simétrica. São, de certa forma, cifras de bloco muito simples com blocos de dimensão igual a um (1).

- O que as torna úteis é o facto da transformação usada na encriptação poder trocar sequencialmente cada símbolo de texto em claro a ser cifrado.

- Em transmissões onde os erros de transmissão são elevados as *stream ciphers* são vantajosas dado não propagarem os erros. Também podem ser utilizadas quando é necessário processar um símbolo de cada vez (devido a, por exemplo, memória limitada).

Cifra sequencial (*stream cipher*)

Definition Let $\mathcal{K}$ be the key space for a set of encryption transformations. A sequence of symbols $e_1 e_2 e_3 \cdots e_i \in \mathcal{K}$, is called a *keystream*.

Definition Let $\mathcal{A}$ be an alphabet of q symbols and let E_e be a simple substitution cipher with block length 1 where $e \in \mathcal{K}$. Let $m_1 m_2 m_3 \cdots$ be a plaintext string and let $e_1 e_2 e_3 \cdots$ be a keystream from $\mathcal{K}$. A *stream cipher* takes the plaintext string and produces a ciphertext string $c_1 c_2 c_3 \cdots$ where $c_i = E_{e_i}(m_i)$. If d_i denotes the inverse of e_i, then $D_{d_i}(c_i) = m_i$ decrypts the ciphertext string.

Cifra sequencial (*stream cipher*) – Cifra de Vernam

Definition The *Vernam Cipher* is a stream cipher defined on the alphabet $A = \{0, 1\}$. A binary message $m_1 m_2 \cdots m_t$ is operated on by a binary key string $k_1 k_2 \cdots k_t$ of the same length to produce a ciphertext string $c_1 c_2 \cdots c_t$ where

$$c_i = m_i \oplus k_i, \quad 1 \leq i \leq t.$$

Quando a chave é escolhida aleatoriamente e é utilizada uma única vez e nunca mais, a cifra de Vernam é designada por ***one-time system*** ou ***one-time pad***.

Este tipo de cifra é, em teoria, inquebrável, se a chave for aleatória.

Cifra sequencial (*stream cipher*) – Cifra de Vernam

Devido às caracteristicas do XOR se se obtiver o texto em claro e o texto cifrado consegue-se obter a chave dado que:

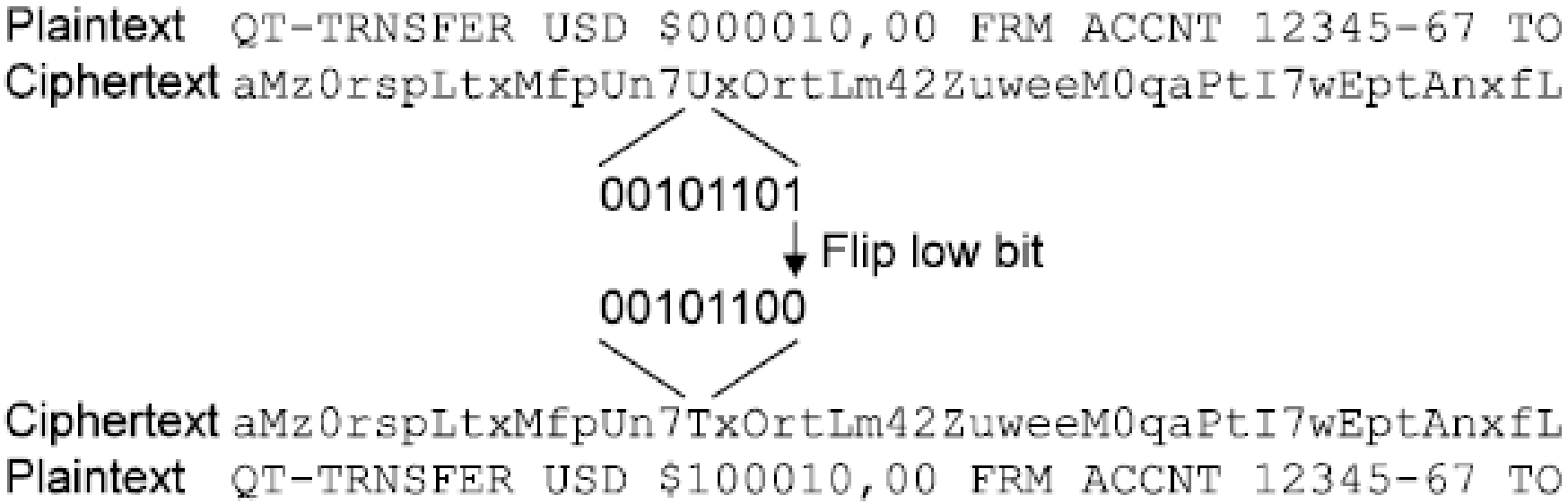
A xor B = C **=>** A xor C = B **=>** B xor C = A

Ou seja, possuíndo quaisquer dos dois componentes, consegue-se recuperar o terceiro, pelo que:

- Nunca se deve reutilizar uma chave numa cifra sequencial
- Ou melhor, nunca reutilizar uma chave.
- A chave a usar deve ser aleatória criando assim um sistema de cifra *one-time pad*.

Cifra sequencial (*stream cipher*) – Cifra de Vernam

Vulnerável a ataques de alteração de bit



Confidencialidade não é garantia de integridade

Cifra sequencial/fluxo (*stream cipher*) – RC4

Cifra sequencial optimizada para implementação rápida em software
Chave de 2046 bits, 8 bits de saída
O algoritmo secreto da RSADSI foi descoberto através de "*reverse engineering*" e divulgado em 1994

```
while( length-- )
{
    x++; sx = state[ x ]; y += sx;
    sy = state[ y ]; state[ y ] = sx; state[ x ] = sy;
    *data++ ^= state[ ( sx+sy ) & 0xFF ];
}
```

Leva pouco tempo a fazer de memória.

Cifra sequencial (*stream cipher*) – RC4

- Extremamente rápido
- Usado no SSL (Netscape, MSIE), Lotus Notes, Windows, encriptação de *passwords*, MS Access, Adobe Acrobat, MS PPTP, Oracle Secure SQL, ...
- Usualmente utilizado o que permite recuperar a chave (encriptação de *passwords* do Windows, autenticação no Windows server, SYSKEY do Windows NT, encriptação de chaves nos servidores Netscape, algumas encriptações utilizadas nos servidores / *browsers* MS, MS PPTP, MS Access, ...)
- Todos os produtos MS que o utilizam implementaram-no de forma errada em alguma altura
- Ilustra o problema de tratar uma cifra como uma caixa preta mágica

Espaço de chaves

- A **dimensão do espaço de chaves** é o número de pares de chaves de encriptação/decriptação que estão disponíveis no sistema de cifra. Uma chave é tipicamente uma forma compacta de especificar a transformação da encriptação (de todo o conjunto de todas as transformações da encriptação) a ser utilizada. Por exemplo, uma cifra de transposição de bloco com dimensão *t* tem *t*! funções de encriptação que pode seleccionar. Cada uma pode ser descrita por uma permutação que é chamada chave.

- É uma grande tentação relacionar a segurança do esquema de encriptação com a dimensão do espaço de chaves. **Só por si a dimensão do espaço de chaves não é suficiente para garantir a segurança de uma cifra**. É necessário mas, usualmente, não suficiente que o espaço de chaves seja suficientemente grande para evitar ataques de força-bruta.

Espaço de chaves

- **Exemplo**:

  Se cifrarmos utilizando a cifra de César (cifra de substituição [c=f(m)=(m+3) mod 26] o espaço de pesquiza é de $26! \approx 4 \times 10^{26}$.

  A cifra de substituição polialfabética, em blocos de três símbolos, tem um espaço de chaves de $(26!)^3 \approx 7 \times 10^{79}$.

  Um ataque por força bruta a estas cifras não é realizável, no entanto são bastante fáceis de quebrar por outros tipos de ataques.

Requisitos para utilização segura de encriptação simétrica

- É necessário um **algoritmo de encriptação robusto**
  - O oponente não deve conseguir descobrir a chave mesmo que tenha acesso a vários textos a cifrar, às correspondentes versões cifradas e ao algoritmo utilizado.

- O emissor e o recetor devem obter cópias das chaves secretas de forma segura e devem manter as chaves seguras. As chaves devem possuir uma **dimensão do espaço de chaves elevado.**

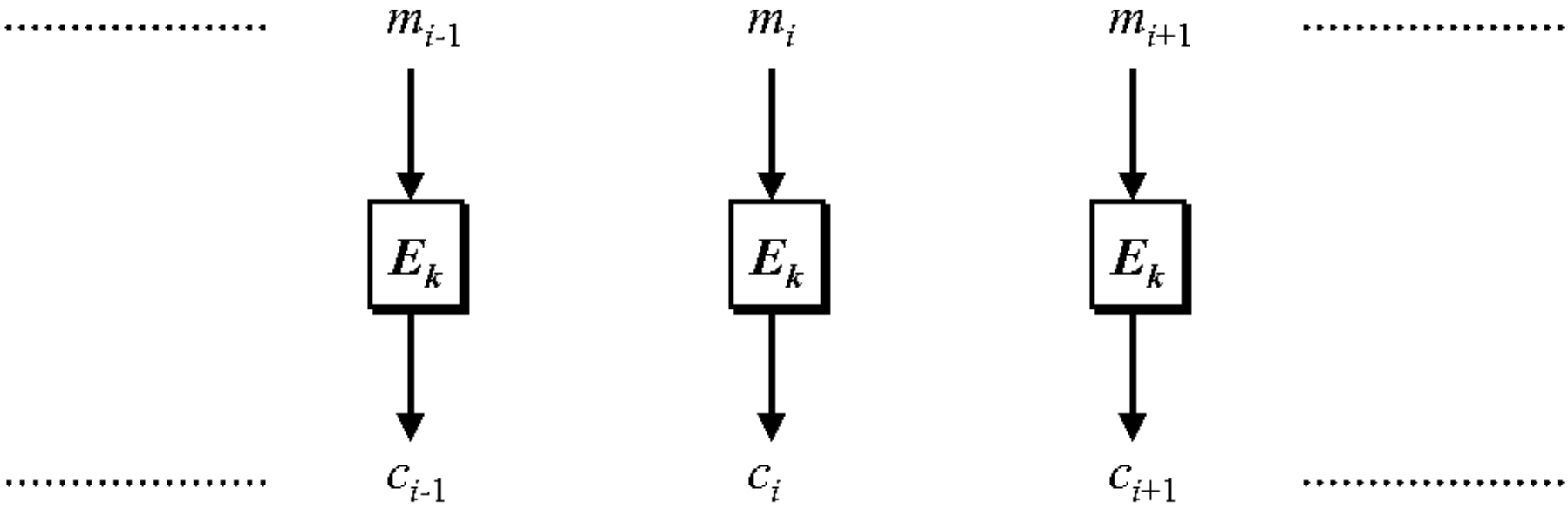
Modos de operação

- In cryptography, a **mode of operation** is an algorithm that uses a block cipher to provide an information service such as confidentiality or authenticity.[1] A block cipher by itself is only suitable for the secure cryptographic transformation (encryption or decryption) of one fixed-length group of bits called a block.[2] A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.
- Most modes require a unique binary sequence, often called an initialization vector (IV), for each encryption operation. The IV has to be non-repeating and, for some modes, random as well. The initialization vector is used to ensure distinct ciphertexts are produced even when the same plaintext is encrypted multiple times independently with the same key.[6] Block ciphers have one or more block size(s), but during transformation the block size is always fixed. Block cipher modes operate on whole blocks and require that the last part of the data be padded to a full block if it is smaller than the current block size.[2] There are, however, modes that do not require padding because they effectively use a block cipher as a stream cipher.

Electronic Code Book (ECB)

- **Cifra um bloco de cada vez** com a chave seleccionada
- Implementação mais simples do DES
- Vulnerabilidade: Texto em claro repetido pode revelar a chave e a partir daí todos os blocos de cifra podem ser decifrados
- Blocos identicos de texto em claro geram blocos de texto cifrado identicos pelo que não esconde bem padrões de dados.

ECB

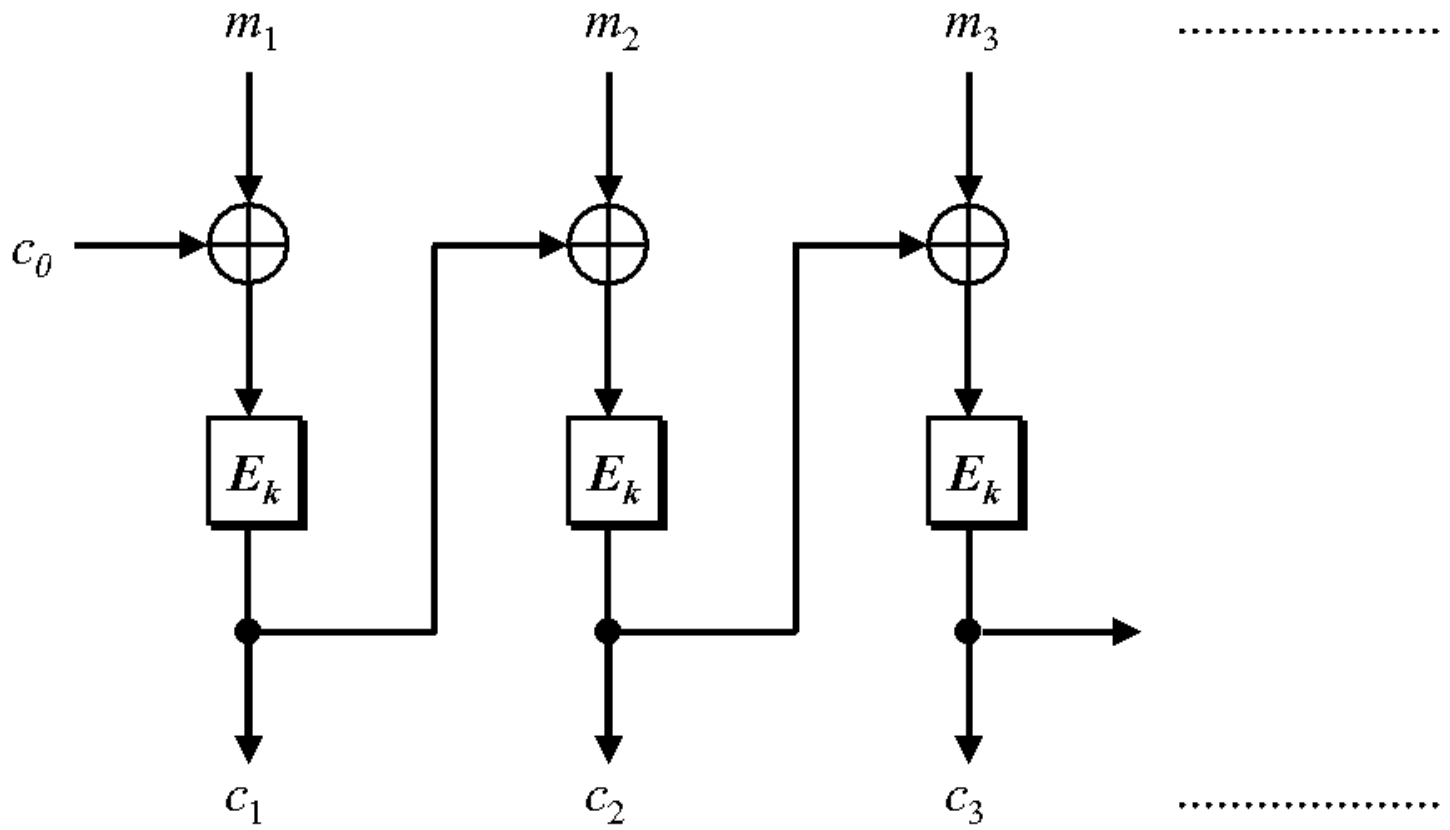


Cipher Block Chaining (CBC)

- A entrada de cada bloco é a saída do bloco anterior adicionado (XOR) com o texto em claro desse bloco
- O bloco inicial é adicionado (XOR) com um vetor de inicialização (*Initialization Vector (IV)*)
- CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. One way to handle this last issue is through the method known as ciphertext stealing. Note that a one-bit change in a plaintext or IV affects all following ciphertext blocks.

Este modo melhora a segurança do DES no que respeita à pesquisa de chaves

CBC

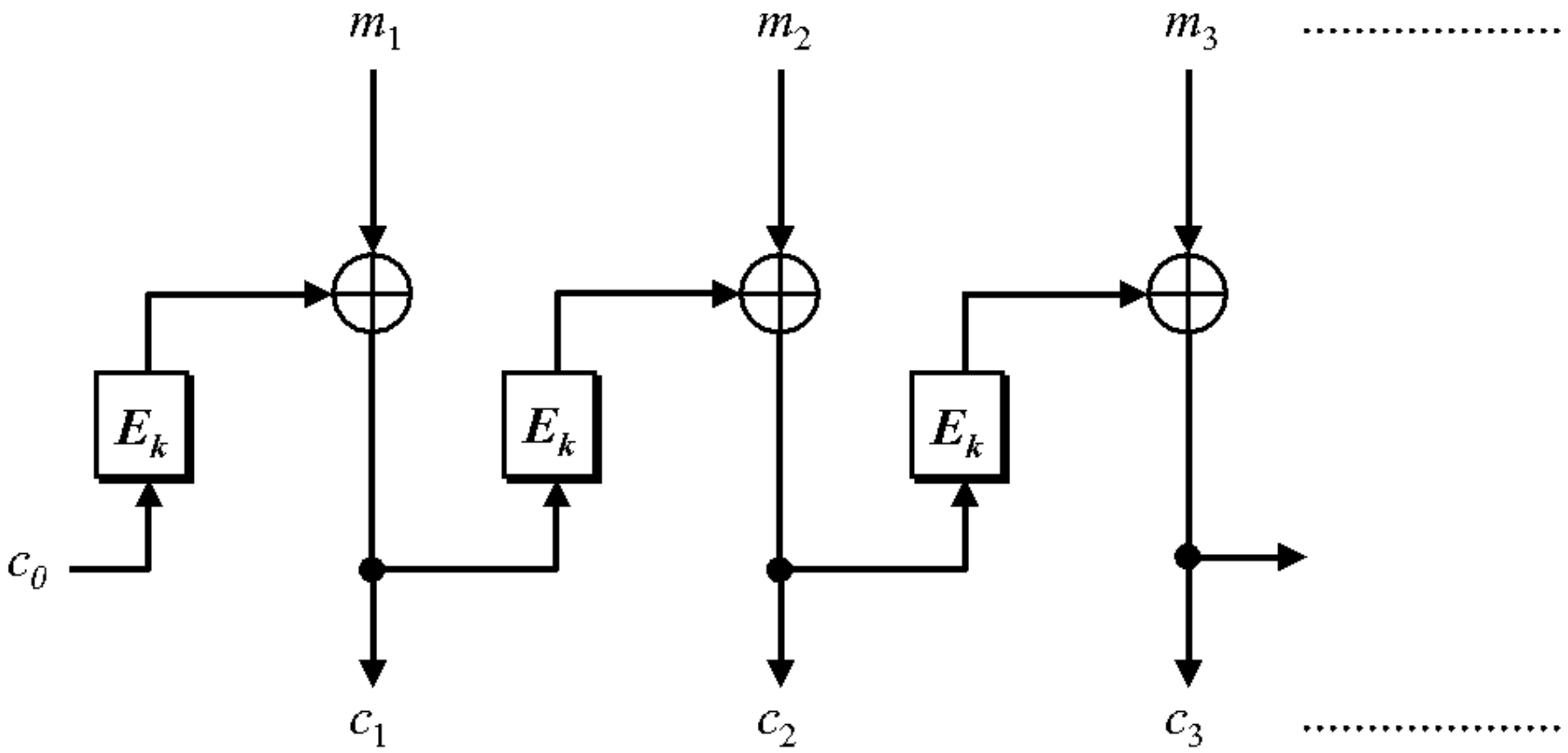


C_0 – Vetor de inicialização (IV)

Cipher Feedback Mode

- O bloco de texto cifrado precedente é adicionado (XOR) com o bloco de texto em claro para produzir o bloco cifrado actual
- Pode utilizar *feedback* o qual é menos que um bloco completo de dados
- O vector de inicialização (IV) é utilizado como "semente" do processo
- The *Cipher Feedback* (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher.

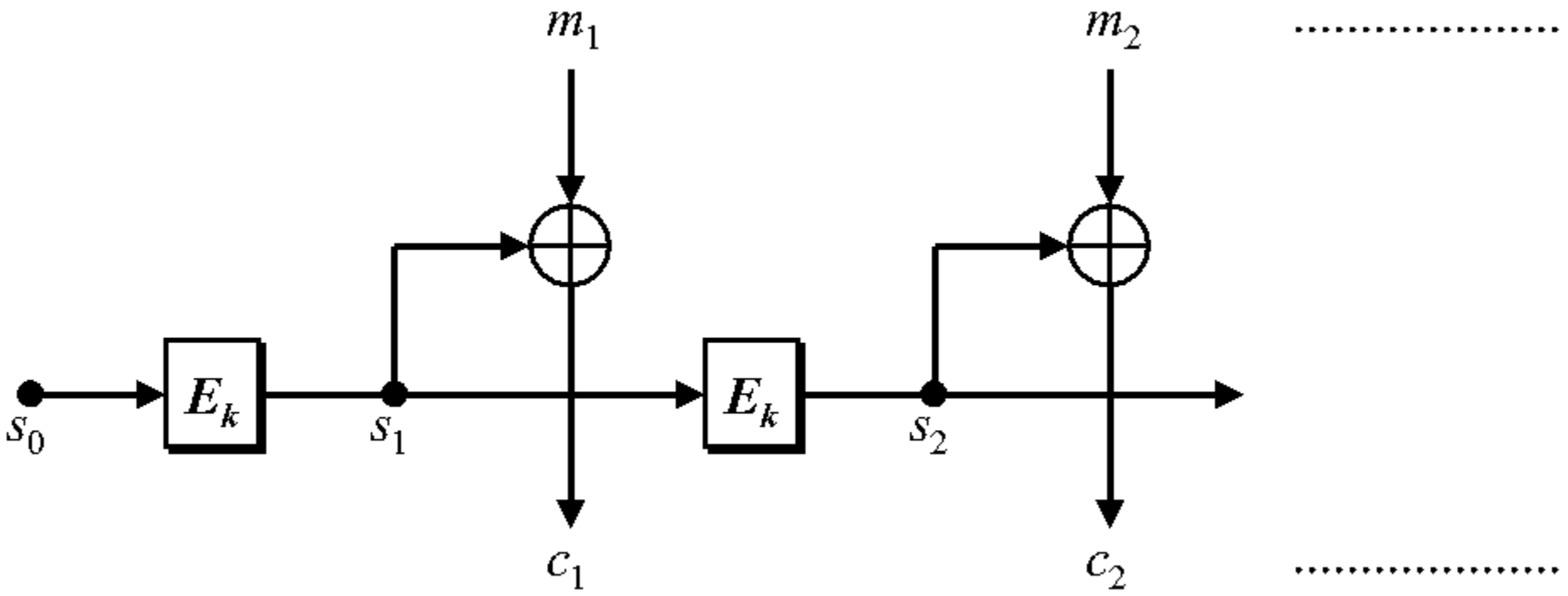
CFB



Output Feedback Mode (OFB)

- Semelhante ao CFB excepto que os dados adicionados (XOR) com cada bloco de texto em claro é gerado independentemente dos blocos de texto em claro e de texto cifrado
- O vector de inicialização s_0 é usado como "semente" para a sequência de blocos de dados s_i
- Cada bloco de dados s_i deriva da encriptação de anteriores blocos de dados s_{i-1}
- The *Output Feedback* (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption.

OFB



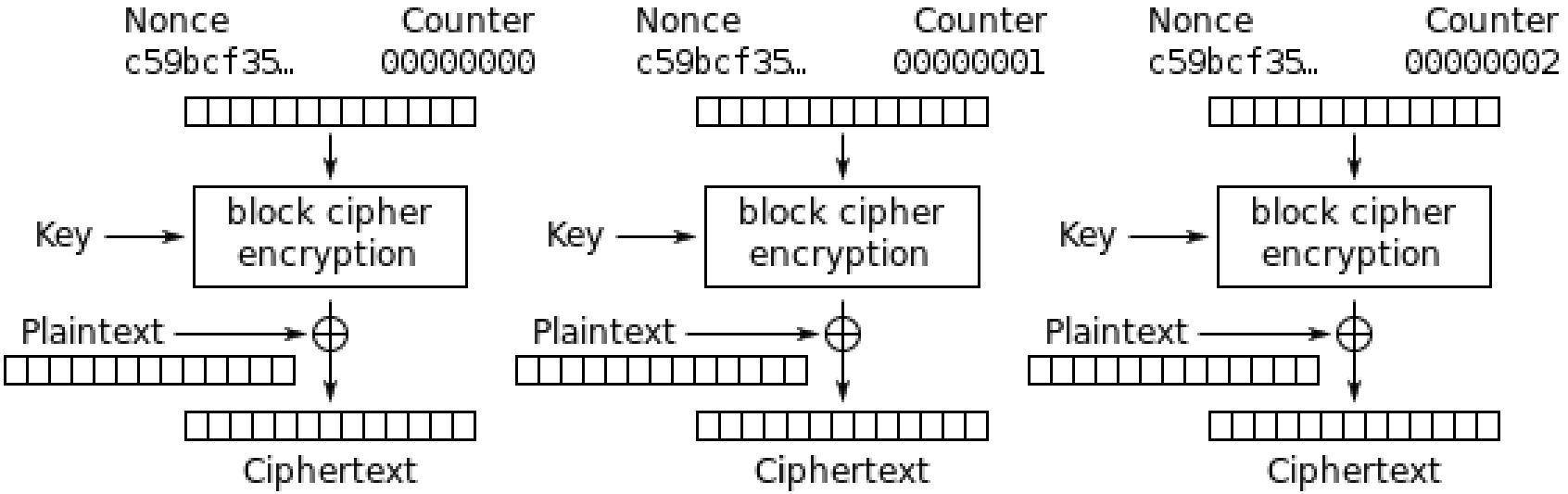
Counter (CTR)

CTR mode (CM) is also known as integer counter mode *(ICM) and* segmented integer counter *(SIC) mode*

Like OFB, Counter mode turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter". The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment-by-one counter is the simplest and most popular.

CTR mode has similar characteristics to OFB, but also allows a random access property during decryption. CTR mode is well suited to operate on a multi-processor machine where blocks can be encrypted in parallel. Furthermore, it does not suffer from the short-cycle problem that can affect OFB.

Counter (CTR)



Counter (CTR) mode encryption

Operation mode comparation

	CBC	CFB	OFB	CTR
Latency	duration of AES encryption	low	low	low
Sync requirements	block boundaries	block or stream data boundaries	block boundaries	block or stream data boundaries
Behaviour on sync loss	self syncs after one block	self syncs after one block	self syncs after one block	must be manually synched with new IV
Behaviour with bit errors	single bit error propagates to two entire blocks	single bit error propagates to one block plus one stream data value	no error propagation	no error propagation
Behaviour with IV errors	self corrects after one block	self corrects after one block	never corrects	never corrects
Underlying AES function required	encryptor for encryption and decryptor for decryption	encryptor supports both encryption and decryption	encryptor supports both encryption and decryption	encryptor supports both encryption and decryption
Padding requirements	must pad to next block boundary	no padding required	no padding required	no padding required

Initialization vector (IV)

An initialization vector (IV) or starting variable (SV) is a block of bits that is used by several modes to randomize the encryption and hence to produce distinct ciphertexts even if the same plaintext is encrypted multiple times, without the need for a slower re-keying process.

An initialization vector has different security requirements than a key, so the IV usually does not need to be secret. However, in most cases, it is important that an initialization vector is never reused under the same key. For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. For OFB and CTR, reusing an IV completely destroys security.

Padding

A block cipher works on units of a fixed size (known as a *block size*), but messages come in a variety of lengths. So some modes (namely ECB and CBC) require that the final block be padded before encryption. Several padding schemes exist. The simplest is to add null bytes to the plaintext to bring its length up to a multiple of the block size, but care must be taken that the original length of the plaintext can be recovered; this is trivial, for example, if the plaintext is a C style string which contains no null bytes except at the end. Slightly more complex is the original DES method, which is to add a single one bit, followed by enough zero bits to fill out the block; if the message ends on a block boundary, a whole padding block will be added. Most sophisticated are CBC-specific schemes such as ciphertext stealing or residual block termination, which do not cause any extra ciphertext, at the expense of some additional complexity. Schneier and Ferguson suggest two possibilities, both simple: append a byte with value 128 (hex 80), followed by as many zero bytes as needed to fill the last block, or pad the last block with *n* bytes all with value *n*.

Padding

Official messages often start and end in predictable ways: *My dear ambassador, Weather report, Sincerely yours*, etc. The primary use of padding with classical ciphers is to prevent the cryptanalyst from using that predictability to find known plaintext[1] that aids in breaking the encryption. Random length padding also prevents an attacker from knowing the exact length of the plaintext message.

Padding

Bit padding

... | 1011 1001 1101 0100 0010 011**1 0000 0000** |

This padding is the first step of a two-step padding scheme used in many hash functions including MD5 and SHA. In this context, it is specified by RFC1321 step 3.1.
This padding scheme is defined by ISO/IEC 9797-1 as Padding Method 2.

Byte padding

Byte padding can be applied to messages that can be encoded as an integral number of bytes.

Padding

ANSI X.923

In ANSI X.923 bytes filled with zeros are padded and the last byte defines the padding boundaries or the number of padded bytes.
Example: In the following example the block size is 8 bytes, and padding is required for 4 bytes (in hexadecimal format.

... | DD DD DD DD DD DD DD DD | DD DD DD DD **00 00 00 04** |

ISO 10126

ISO 10126 (withdrawn, 2007[2][3]) specifies that the padding should be done at the end of that last block with random bytes, and the padding boundary should be specified by the last byte.
Example: In the following example the block size is 8 bytes and padding is required for 4 bytes

... | DD DD DD DD DD DD DD DD | DD DD DD DD **81 A6 23 04** |

Padding

PKCS7

PKCS#7 is described in RFC 5652.
Padding is in whole bytes. The value of each added byte is the number of bytes that are added, i.e. *N* bytes, each of value *N* are added. The number of bytes added will depend on the block boundary to which the message needs to be extended.
The padding will be one of:

01
02 02
03 03 03
04 04 04 04
05 05 05 05 05
etc.

If the original data is a multiple of *N* bytes, then an extra block of bytes with value *N* is added.

This padding method (as well as the previous two) is well-defined if and only if *N* is less than 256.
Example: In the following example the block size is 8 bytes and padding is required for 4 bytes: ... | DD DD DD DD DD DD DD DD | DD DD DD DD **04 04 04 04** |

Estrutura de cifra de Feistel

- A maioria dos algoritmos de cifra convencional por blocos, incluindo o DES, utilizam uma estrutura descrita por Horst Feistel, da IBM, em 1973

Feistel Structure for Block Ciphers

- Feistel proposed applying two or more simple ciphers in sequence so final result is cryptographically stronger than component ciphers
- n-bit block length; k-bit key length; 2k transformations
- Feistel cipher alternates: substitutions, transpositions (permutations)
- Applies concepts of diffusion and confusion
- Applied in many ciphers today
- Approach:
  - Plaintext split into halves
  - Subkeys (or round keys) generated from key
  - Round function, F, applied to right half
  - Apply substitution on left half using XOR
  - Apply permutation: interchange to halves

Using the Feistel Structure

- Exact implementation depends on various design features
  - Block size, e.g. 64, 128 bits: larger values leads to more diffusion
  - Key size, e.g. 128 bits: larger values leads to more confusion, resistance against brute force
  - Number of rounds, e.g. 16 rounds
  - Subkey generation algorithm: should be complex
  - Round function F: should be complex
- Other factors include fast encryption in software and ease of analysis
- Trade-off: security vs performance

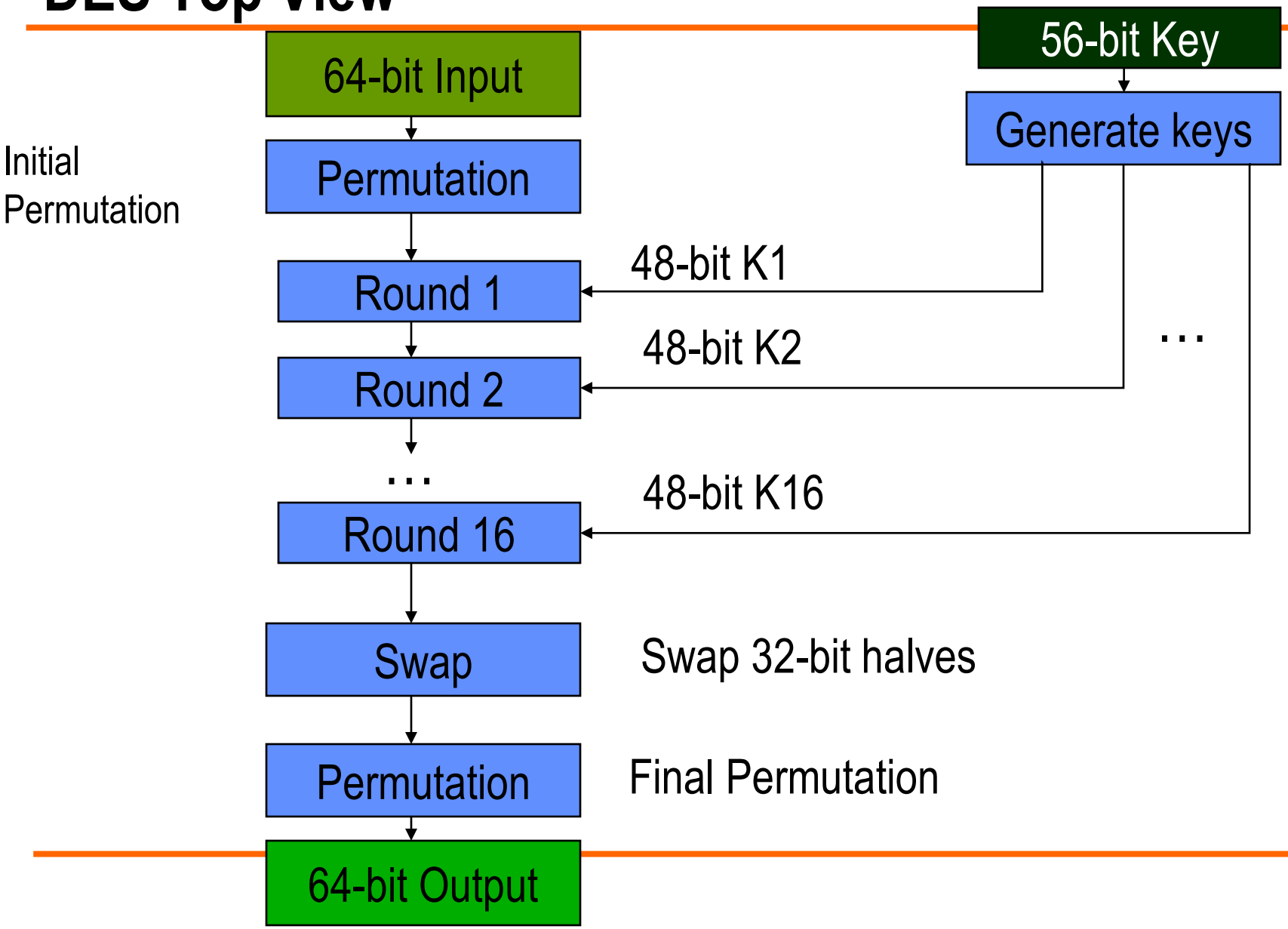
Estrutura da cifra de Feistel

- A realização exacta de uma rede de Feistel depende da escolha dos parâmetros:
  - **Dimensão do bloco:** Maior dimensão do bloco implica maior segurança
  - **Dimensão da chave:** Chaves maiores significam maior segurança. A maioria das chaves actuais é de 128 bits ou mais
  - **Número de "*rounds*":** Múltiplos "*rounds*" oferecem maior segurança. Um número comum é 16
  - **Algoritmo de geração da sub-chaves:** Maior complexidade leva a maior dificuldade na criptanálise
  - **Função "*round*":** Quanto mais complexa, mais dificil a criptoanálise
  - **Software de encriptação/desencriptação:** A velocidade de execução do algoritmo de cifra/decifra é uma preocupação

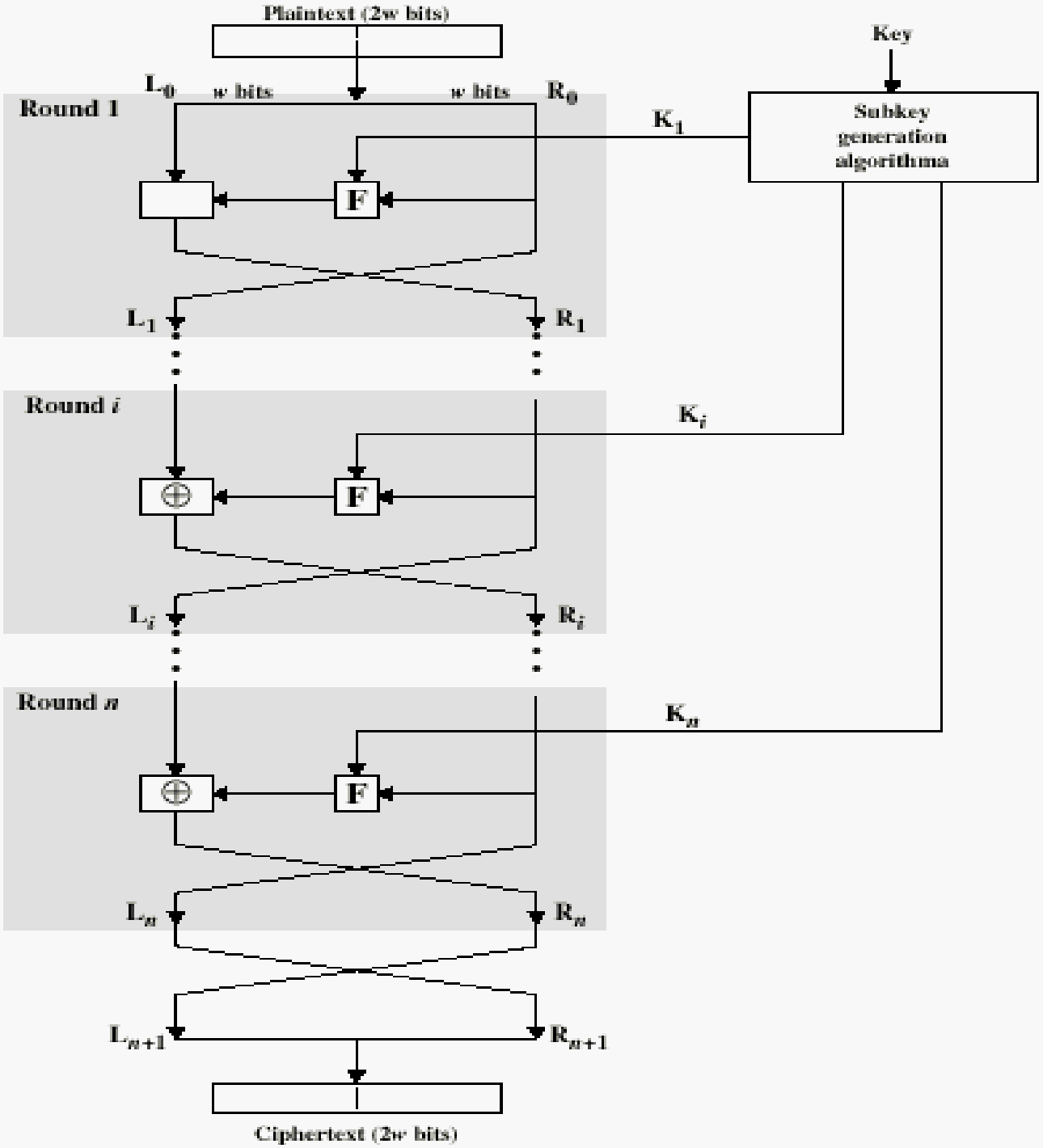
Estrutura da cifra de Feistel

- Outras considerações a ter no desenho de uma cifra de Feistel:
  - **Software de encriptação/desencriptação rápido:** A velocidade de execução do algoritmo de encriptação/desencriptação é uma preocupação
  - **Facilidade de análise:** O algoritmo utilizado deve ser fácil de analisar e explicar

DES Top View

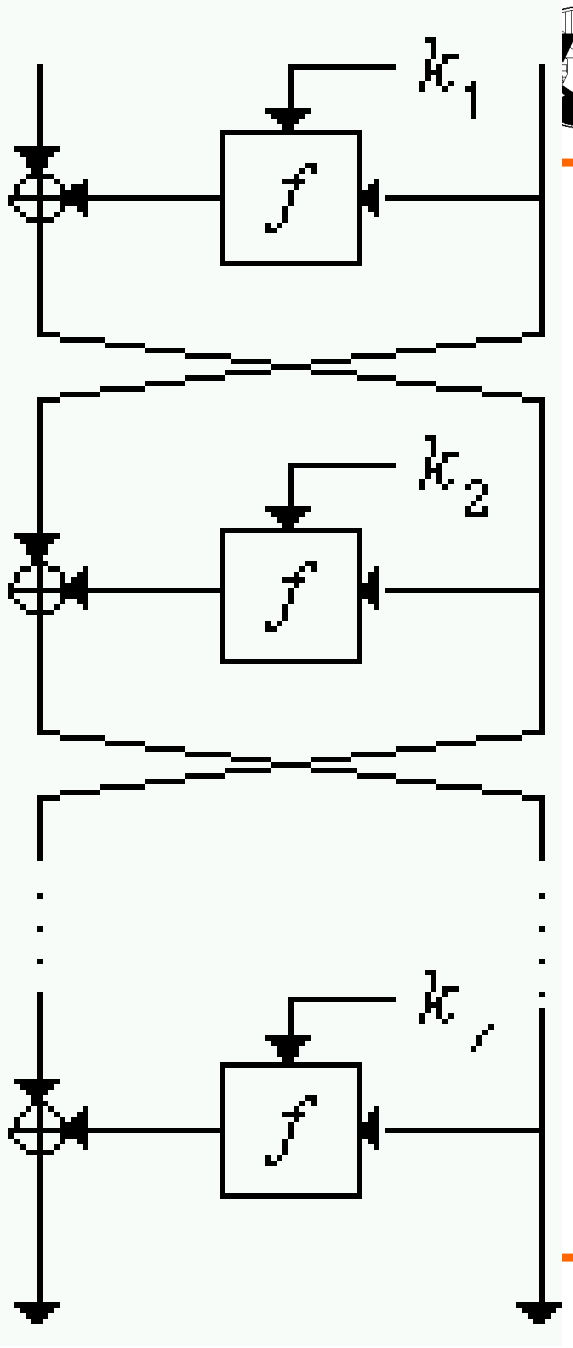

64-bit Input
Initial Permutation
Permutation
Round 1
Round 2
...
Round 16
Swap
Permutation
64-bit Output
56-bit Key
Generate keys
48-bit K1
48-bit K2
...
48-bit K16
Swap 32-bit halves
Final Permutation

Rede de Feistel clássica



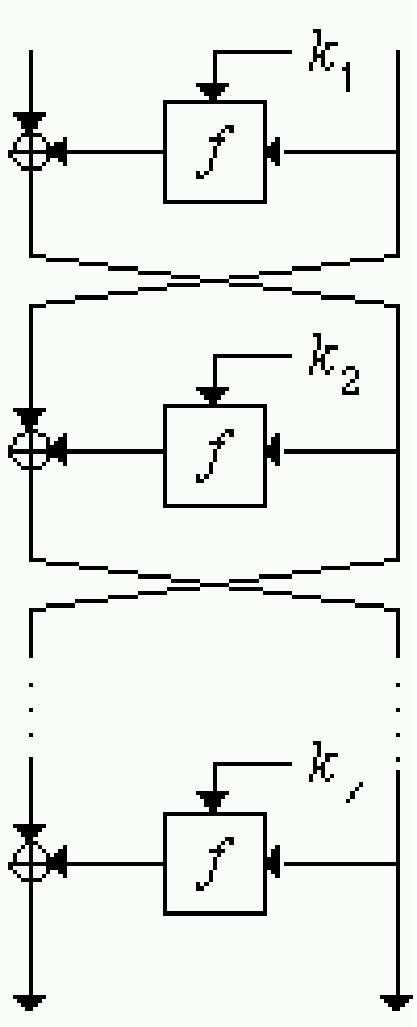
Características da cifra de Feistel

- Tipo iteração de blocos de cifra
- Texto cifrado calculado a partir do texto em claro por aplicações repetidas da mesma transformação ou função "*round*"
- **Encriptação e desencriptação são estruturalmente idênticas (na desencriptação a ordem das subchaves é invertida)**
- Rápida, mesmo quando implementada em software
- Facilmente analisada (por exemplo: Deficiências encontram-se melhor através de análise)



Cifra de Feistel: Passo a passo

- O texto em claro é dividido em duas metades
- A função "*round*" ***f*** é aplicada a uma das metades utilizando uma subchave
- A saída de ***f*** é adicionada (XOR) com a outra metade do texto em claro
- As duas metades são trocadas (*swapped*)
- O processo é repetido para *n* "*rounds*"
- Não há troca depois do último "*round*"



Geração das subchaves

- A criação das subchaves numa cifra de Feistel tem um efeito importante na segurança do algoritmo
  - Possibilidade de criar chaves fracas
  - Alterações no algoritmo das subchaves pode resultar em implementações diferentes do algoritmo de cifra

- O DES é baseado nos "*rounds*" de Feistel e utiliza um método sofisticado de geração das subchaves

Importância das cifras de Feistel

- Base do DES e outros algoritmos importantes
  - Horst Feistel trabalhou para a IBM em 1973
  - O algoritmo *Lucifer* (1974) da IBM, baseado nos "*rounds*" de Feistel, evoluíu para o DES em 1977

- Muitos outros autores de algoritmos de cifra em bloco utilizaram os "*rounds*" de Feistel ou variantes nos seus algoritmos

- As cifras de Feistel não são o único tipo de cifras em bloco iterativas

DES: *Data Encryption Standard*

DES: Aplicação do algoritmo de Feistel

- Desenvolvido em 1977 pela NBS/NIST
- Desenhado pela IBM (Lucifer) com contributos da NSA

Especificação formal do DES: FIPS PUB 46-3, última atualização 25 Outubro1999

http://www.csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

- Descreve dois algoritmos criptográficos
  - DES
  - TDEA (usualmente referido como 3DES - *triple* DES)

- O DES é baseado no *Lucifer* de 1974 da IBM

DES: Características

- Blocos de cifra de 64 bit
- Chave de 56 bit, com 8 bits adicionais utilizados para teste de erro (paridade ímpar em cada byte)
- M**odos de operação**:
    - *Electronic Code Book* (ECB)
    - *Cipher Block Chaining* (CBC)
    - *Cipher Feedback* (CFB)
    - *Output Feedback* (OFB)

O modo de operação Counter mode (CTR) também pode ser usado com o DES, tal como com outras cifras de bloco.

Exemplo: Cifrar com o DES

Se cifrarmos a mensagem de texto em claro (em hexadecimal):

M = 8787878787878787 (64 bits)

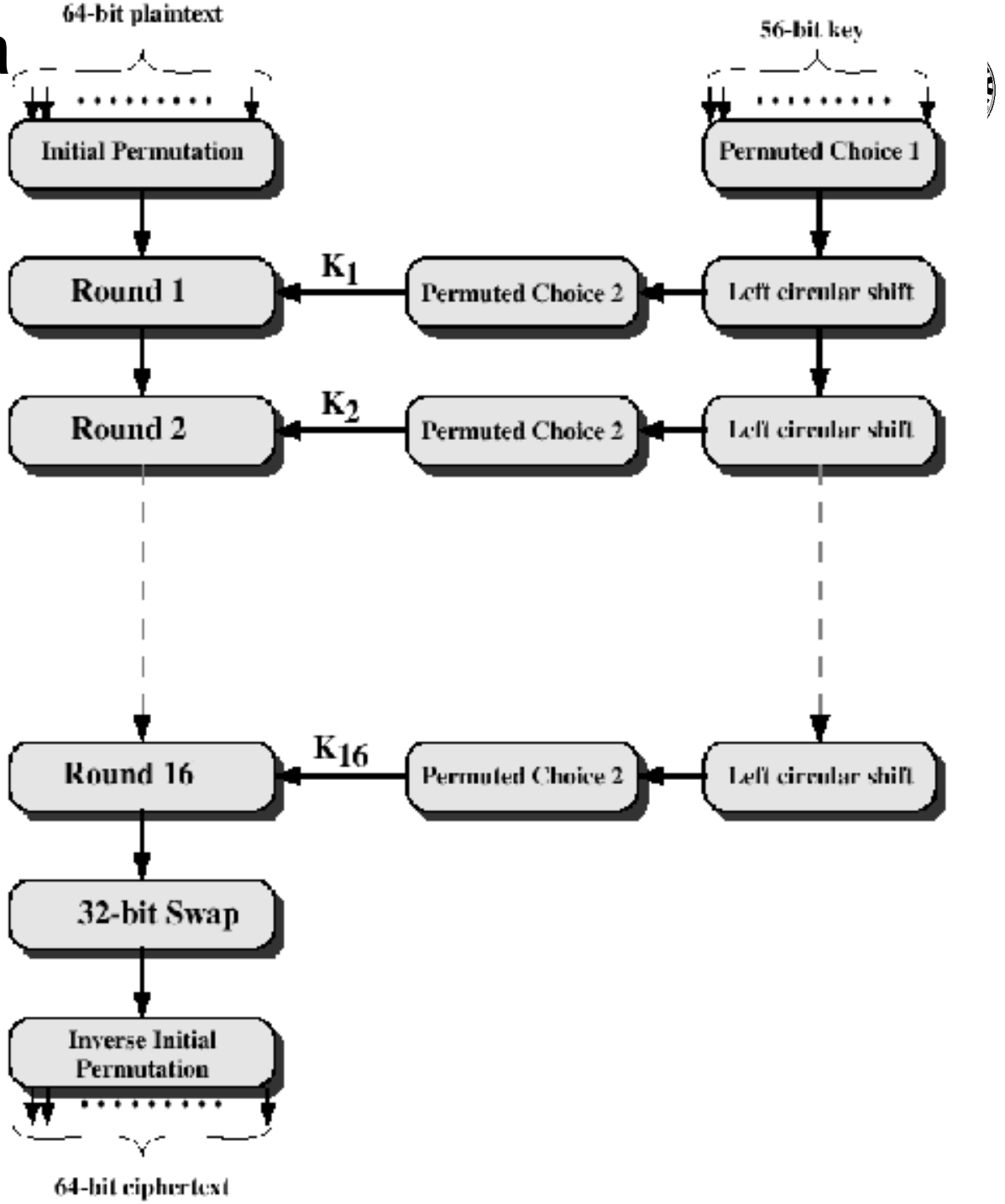
com a chave:

K = 0E329232EA6D0D73 (64 bits)

utilizando o DES, obtemos o texto cifrado:

C = 0000000000000000 (64 bits)

Descrição genérica do DES

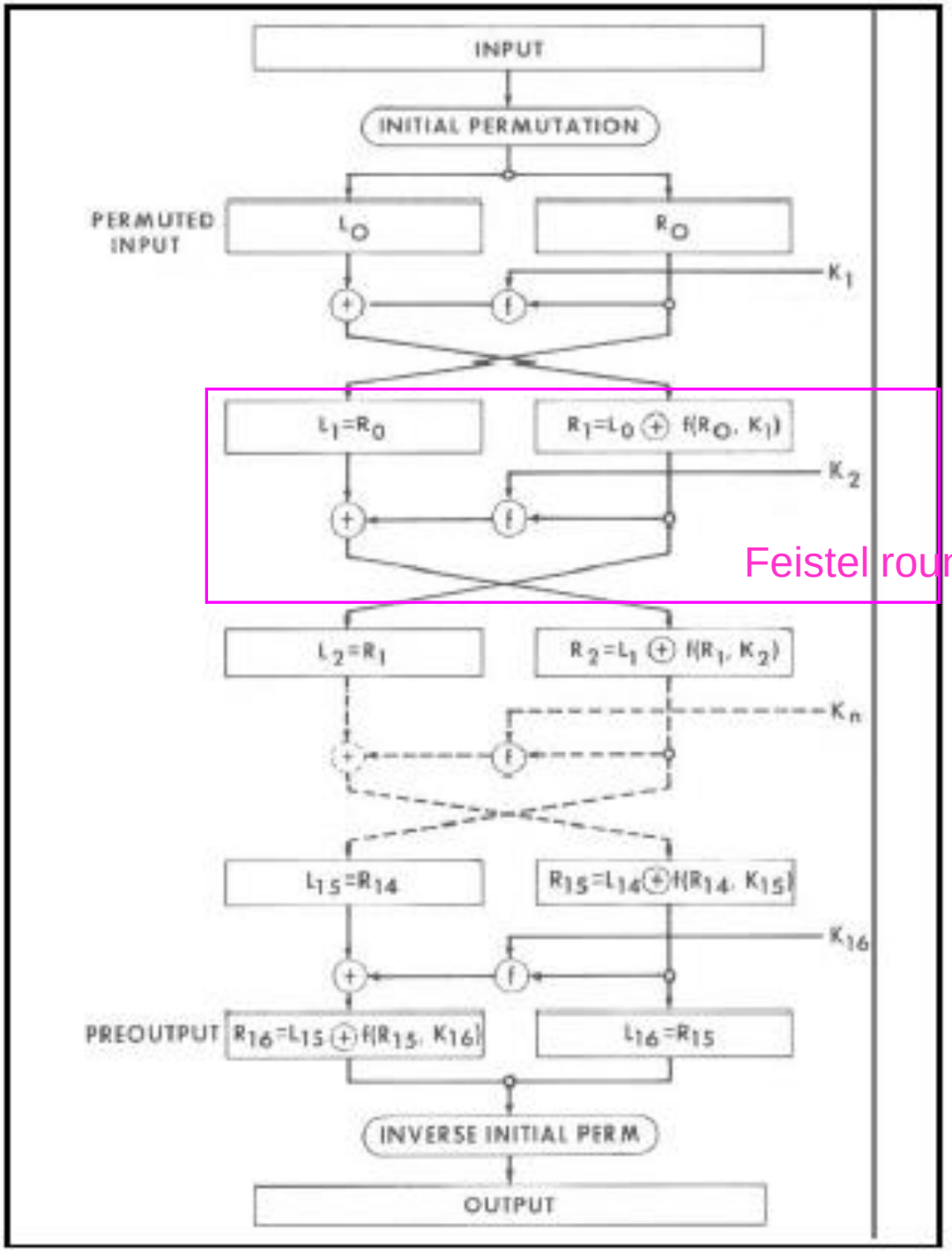


DES

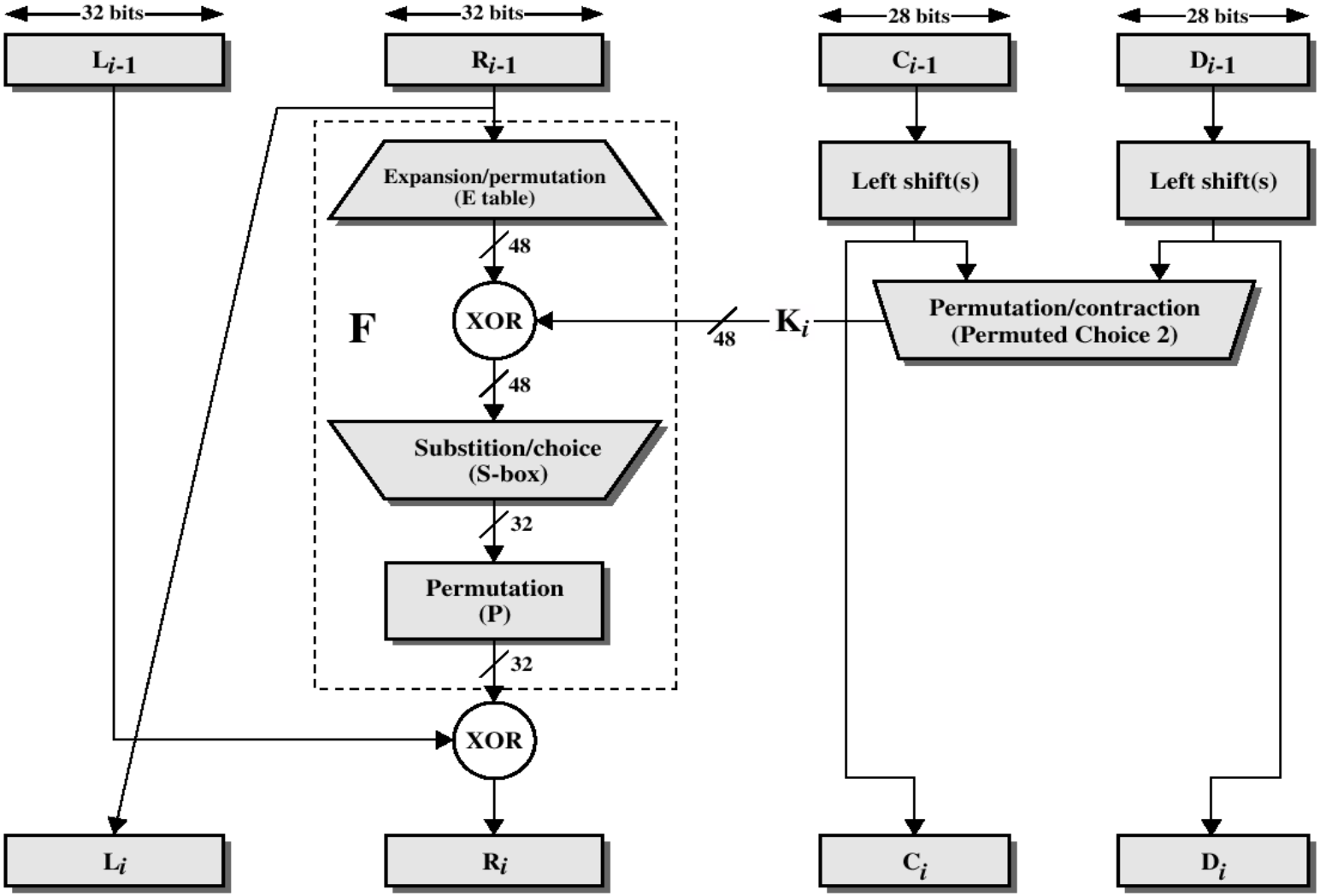
- Processamento em cada iteração (”*round*”):
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

- Preocupações:
  - O algoritmo e o comprimento da chave (56 bits)

DES: Cifra


INPUT
INITIAL PERMUTATION
PERMUTED INPUT
L_0
R_0
K_1
$L_1=R_0$
$R_1=L_0 \oplus f(R_0, K_1)$
K_2
$L_2=R_1$
$R_2=L_1 \oplus f(R_1, K_2)$
K_n
$L_{15}=R_{14}$
$R_{15}=L_{14} \oplus f(R_{14}, K_{15})$
K_{16}
PREOUTPUT
$R_{16}=L_{15} \oplus f(R_{15}, K_{16})$
$L_{16}=R_{15}$
INVERSE INITIAL PERM
OUTPUT
Feistel round

Um "*round*" do algoritmo DES

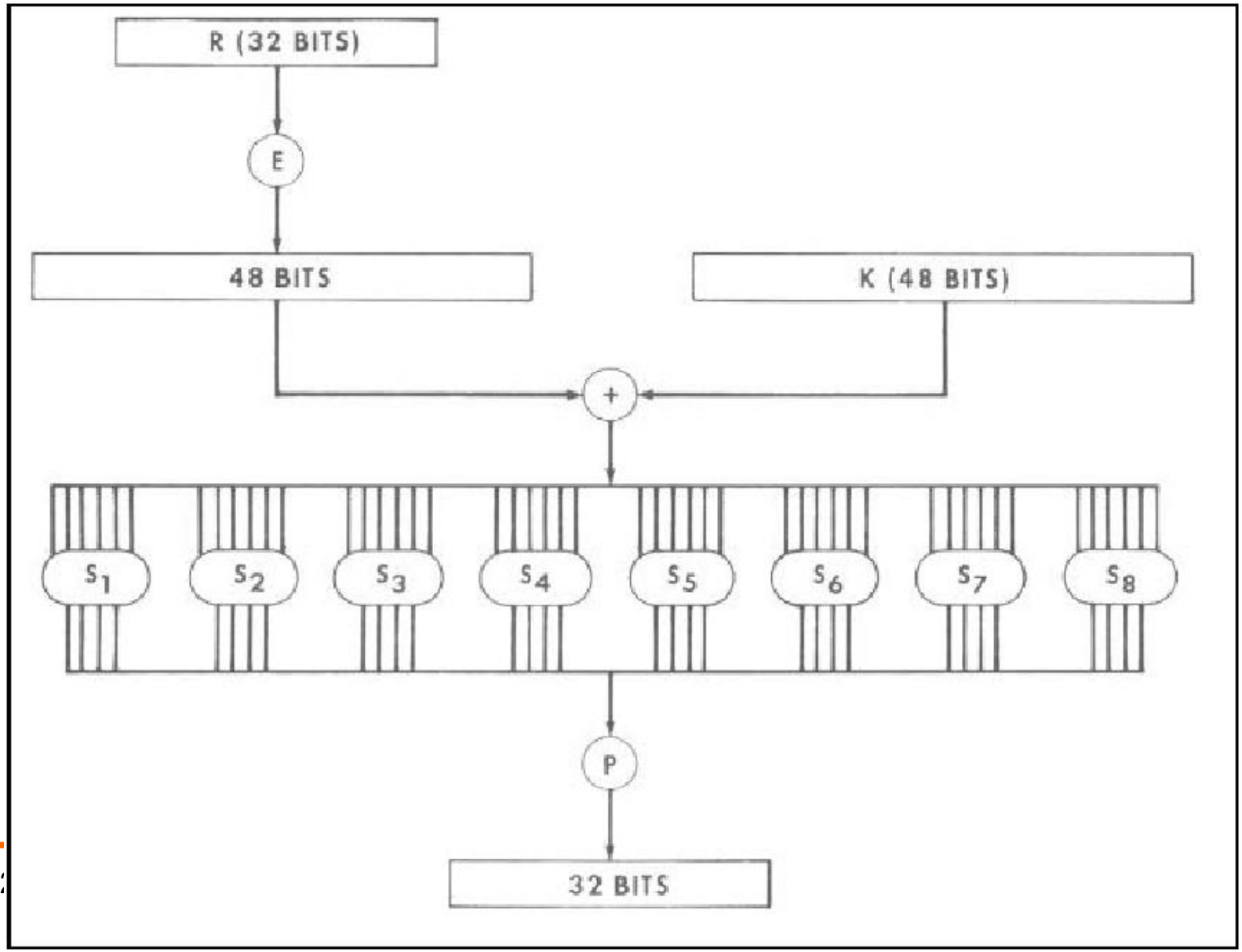


Permutação inicial

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

Função de cifra, $f(R_n,K_n)$



Como é realizado?

- Passar de 32 bits de texto em claro para 48 bits à saída
- Adicionar a chave de 48 bits
- Obter 32 bits à saída

Detalhes da função de cifra

- A **função *E*** aceita a entrada do "*round*" de Feistel e expande-a para 48 bits

- As ***S-boxes*** *(*caixas S) (para selecção, normalmente referida como substituição) permutam bits para produzir a saída necessária

- A **função *P*** permuta a saída a 32 bits das *S-boxes*

- Uma **permutação inversa (*IP^{-1}*)** repõe a ordem dos bits depois dos 16 "*rounds*" de Feistel

Função *E*

E BIT-SELECTION TABLE

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

Repare-se que existem bits do bloco de entrada que são repetidos no bloco de saída

Função *P*

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

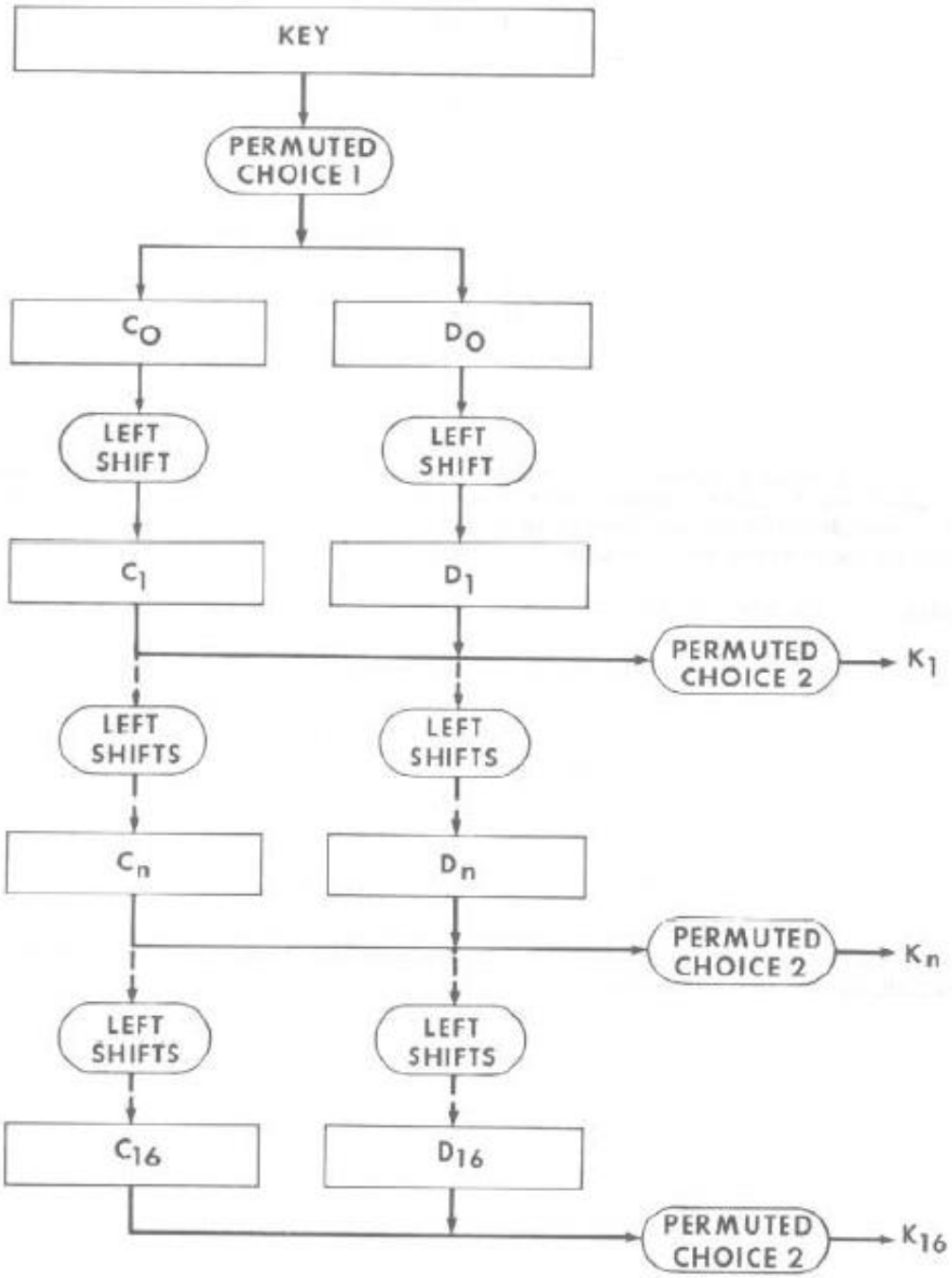
Exemplo de *S-box*

Cada uma das funções de selecção S1, S2, ... , S8 aceita 6 bits à entrada e dá como resultado um conjunto de 4 bits à saída. Dos 6 bits de entrada o de maior peso e o de menor peso servem de índice de linha, os 4 restantes como índices de coluna.

Row No.	Column Number															
	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

Resultado das 8 *S*-boxes: 48 bits ➡ 32 bits

Geração de subchaves



Permuted Choice 1

PC-1

57	49	41	33	25	17	9	$C_{()}$
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	$D_{()}$
7	62	54	46	38	30	22	
14	6	61	53	45	37	29	
21	13	5	28	20	12	4	

Tabela de *rotates* à esquerda

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Permuted Choice 2

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

DES: Decifração

- Como o DES é uma cifra de Feistel a decifração usa o mesmo dispositivo que a cifra
- Para decifrar:
  - O dispositivo DES é precisamente o mesmo que o utilizado para cifrar – não corre ao contrário (por exemplo, com os dados a entrar por "baixo")
  - Em vez disso, a tabela de chaves corre ao contrário, isto é a primeira subchave utilizada é a K_{16}, depois a K_{15}, etc., acabando na K_1

Importância do DES

- Ubiquo, norma federal dos EUA
- Quando foi normalizado, com 56 bits de chave, tera uma cifra computacionalmente segura
  - Já não é o caso actualmente
  - O DES foi quebrado utilizando ataques de força bruta em 56 horas, usando computadores reciclados com um custo inferior a $250.000 (15 de Julho de 1998)
- Remédio imediato: Algoritmo *Triple Data Encryption* (ou *Triple* DES, 3DES)

Vale apena ler o artigo
The DES Algorithm Illustrated, por *J. Orlin Grabbe*
para se perceber melhor o DES
http://www.aci.net/kalliste/des.htm

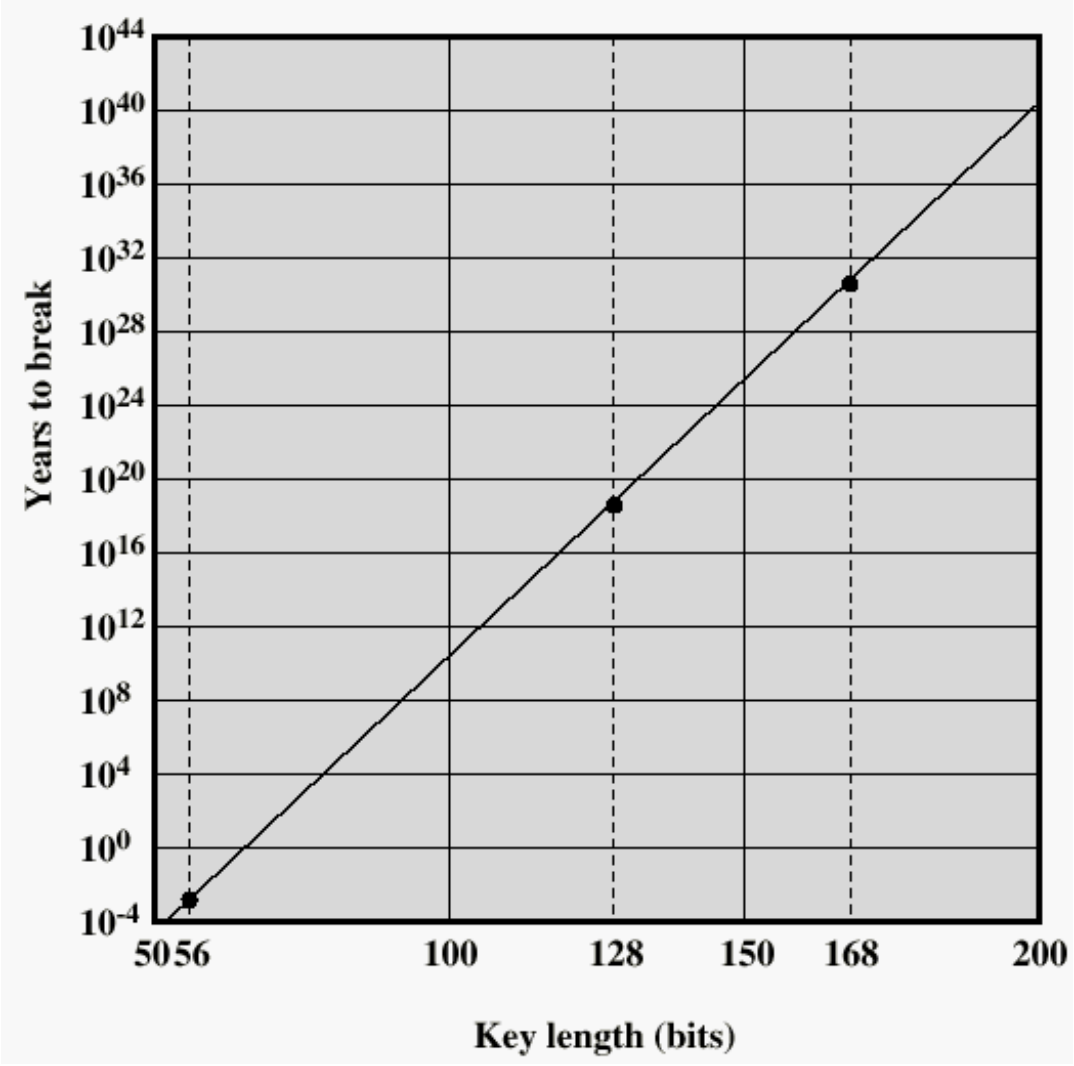
Tempo para quebrar um código (10^6 decifrações/μs)

- Existem 2^{56} chaves possíveis de 56 bits (~ $7{,}2 \times 10^{16}$)
- Em 1993 foi feito um estudo de custo de uma máquina paralela para quebrar o DES:

Key Search Machine Unit Cost	Expected Search Time
$100,000	35 hours
$1,000,000	3.5 hours
$10,000,000	21 minutes

- Desafio
  - Em 29 de janeiro de 1997, *RSA Laboratories* publicou um desafio para ver se alguém era capaz de quebrar uma mensagem cifrada com DES
  - Um consultor desenvolveu um programa de força bruta e distribuiu-o pela Internet
  - 96 dias depois a mensagem foi quebrada
  - Mais de 70.000 máquinas foram usadas

Tempo para quebrar um código (10^6 decifrações/μs)



Broken DES

DES encryption was broken in 1999 by Electronics Frontiers Foundation (EFF, www.eff.org). This resulted in NIST issuing a new directive that year that required organizations to use **Triple DES**, that is, three consecutive applications of DES. (That DES was found to be not as strong as originally believed also prompted NIST to initiate the development of new standards for data encryption. **The result is AES**.)

Triplo DEA (TDEA/3DES)

- Uso de três chaves e três execuções do algoritmo DES (cifra-decifra-cifra)

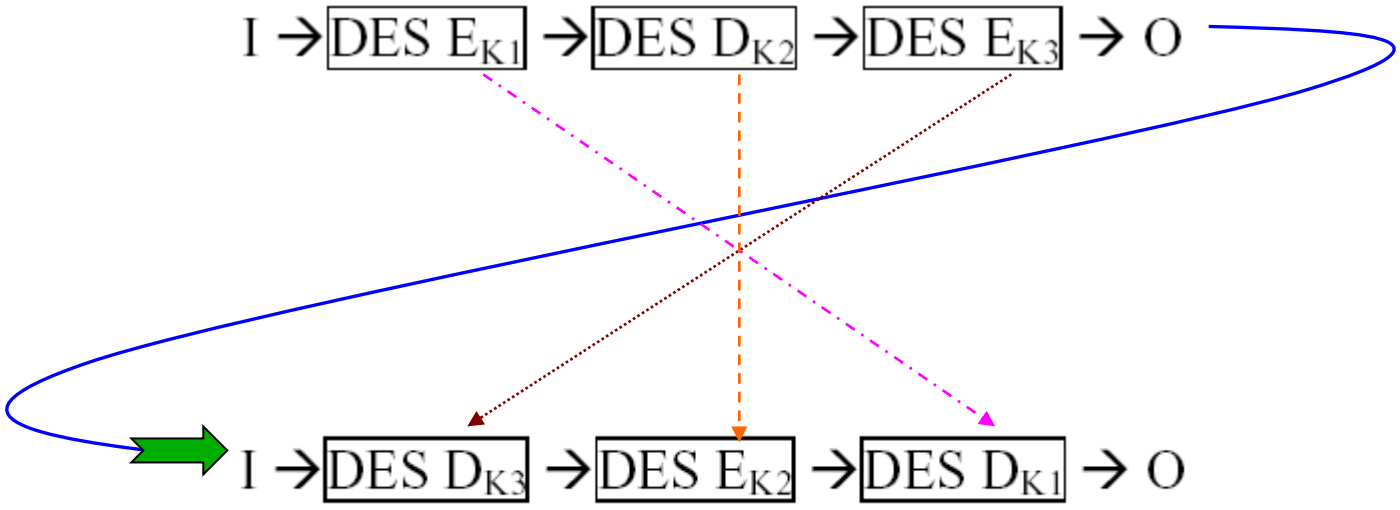
$$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$

  - C = cifra
  - P = Texto em claro
  - $E_{K[X]}$ = cifra de X usando a chave K
  - $D_{K[Y]}$ = decifra de Y usando a chave K

- Comprimento efectivo da chave: 168 bits (3 x 56 bits)

TDEA

Encriptação



Decriptação

Realidades do TDEA

- Duas opções de chave
  - Três chaves separadas (como mostrado no *slide* anterior)
  - Duas chaves; $E_{K1} = E_{K3}$
  - Comprimento resultante das chaves de 168 ou 112 bits
- O *3-key* TDEA é apenas cerca de duas vezes mais seguro que o DES, não três vezes mais seguro
- Implementado em hardware o 3-key TDEA atinge débitos de 1 Gbps

Vantagens do TDEA

- Já muito analisado, não parece ter vulnerabilidades escondidas
- Muito menos vulnerável a ataques de força bruta que o DES
- Pode ser implementado em silicio suportando débitos muito elevados

Desvantagens do TDEA

- A implementação em software é lenta
- Limitado a blocos com a dimensão de 64 bit
- “Triplica” o problema de distribuição de chaves do DES

AES: Evolução face ao DES

- ***Advanced Encryption Standard*** (FIPS PUB 197)
  - Criado para tornear a fraqueza do DES e do TDEA
  - Baseado no algoritmo de Rijndael
    - Joan Daemen e Vincent Rijmen, Belgas, autores
  - Norma dos EUA adoptada em 26 de Novembro de 2001
  - Tornou-se efectiva em 26 de Maio de 2002
  - Comprimento das chaves de 128, 192, e 256 bits
  - Blocos de dimensão 128 bits
    - No AES, Rijndael permite outros comprimentos
  - Menos testado que o DES
  - Provavelmente uma longa coexistência entre o TDEA e o AES

AES: Próxima geração (actual) em relação ao DES

Seleção do *Advanced Encryption Standard* (AES)

Processo selectivo do algoritmo desde 1997

Etapa 1: 15 candidatos selecionados em Agosto 1998

Etapa 2: 5 finalistas anunciados em Agosto 1999

Escolha do vencedor em Outubro 2000

Normalizado em 2001

Vencedor: **Rijndael** – Vincent Rijmen, Joan Daemen

Outros algoritmos finalistas:

- MARS – IBM
- RC6 – RSA Laboratories
- Serpent – Ross Anderson, Eli Biham, Lars Knudsen
- Twofish – Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

AES: Próxima geração (actual) em relação ao DES

Futuro NIST: *Advanced Encryption Standard* (AES)

- **Requisitos mínimos definidos para o AES**:
  - Algoritmo publicamente definido
  - Ser uma cifra simétrica de bloco
  - Projectado para que o tamanho da chave possa aumentar
  - Implementável tanto em hardware quanto em software
  - Disponibilizado livremente ou de acordo com termos ANSI
- **Factores de julgamento**:
  - Segurança (esforço requerido para criptoanálise)
  - Eficiência computacional
  - Requisitos de memória
  - Adequação a hardware e software
  - Simplicidade
  - Flexibilidade

Estrutura de Rinjdael

- Rijndael não é uma cifra de Feistel; utiliza caixas de substitição
- "...tipicamente parte dos bits do estado intermédio são apenas transpostos para outras posições não sendo modificados"
- "...[cada] transformação num "*round*" é composta por três transformações unifomes distintas e inversíveis

Outros algoritmos de cifra em bloco simétricas

- ***International Data Encryption Algorithm* (IDEA)**
  - Chave de 128 bits
  - Usado no PGP

- ***Blowfish***
  - Fácil de implementar
  - Velocidade de execução elevada
  - Corre em menos de 5K bytes de memória

Outras cifras em bloco simétricas

- **RC5**
  - Apropriada para ser implementada em hardware e software
  - Rápido e simples
  - Adaptável a processadores com diferentes comprimentos de palavras
  - Número variável de "*rounds*"
  - Chaves de comprimento variável
  - Necessidade baixa de memória
  - Segurança elevada
  - Rotações dependentes dos dados

- **Cast-128**
  - Dimensão da chave de 40 a 128 bits
  - A função dos "*round*" difere de "*round*" para "*round*"

Problema da distribuição de chaves

- As chaves secretas devem ser preposicionadas em todos os locais antes de poderem ocorrer comunicações seguras

- Como fazer isso?
  - Transporte físico seguro
  - Transporte electrónico seguro

A procura de uma forma de conseguir a distribuição de chaves secretas levou ao desenvolvimento da criptografia de chave pública (***public key cryptography)***

Distribuição de chaves

- **Chaves de sessão:**
  - Dados cifrados com uma chave de utilização única. No fim da sessão a chave é destruída

- **Chaves permanentes:**
  - Utilizada entre entidades com a finalidade de distribuir as chaves de sessão

Distribuição de chaves

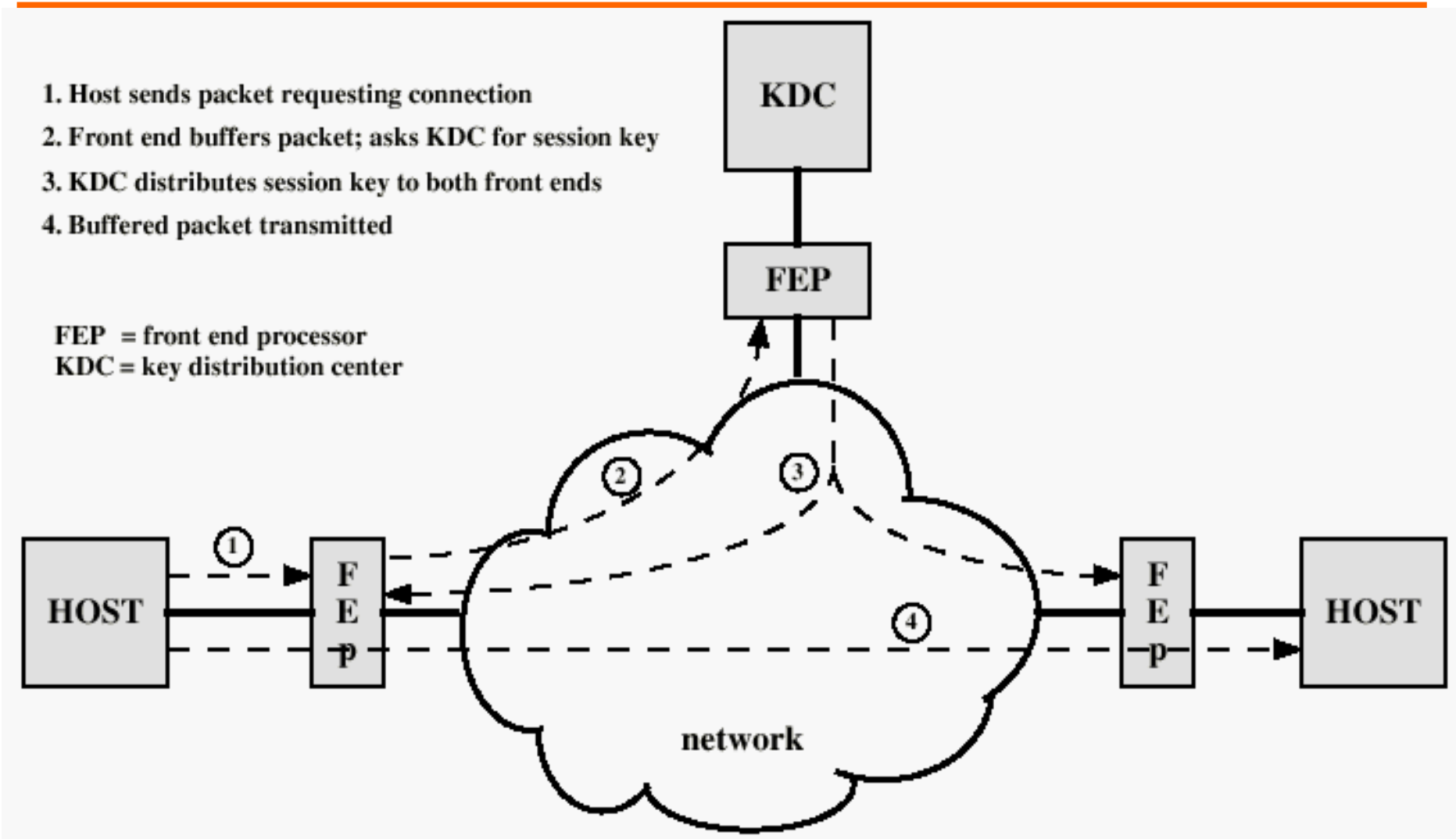
1. Uma chave pode ser seleccionada por A e entregue, fisicamente, a B

2. Uma terceira entidade pode seleccionar uma chave entregá-la fisicamente a A e a B

3. Se A e B tiverem utilizado previamente uma chave, a nova chave pode ser entregue utilizando a chave anterior

4. Se A e B tiverem cada um uma ligação cifrada a uma terceira entidade C, C pode entregar a nova chave, através da ligação cifrada, a A e a B

Distribuição automática de chaves para protocolos orientados à ligação



Leitura recomendada

- A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*
- Stallings, W. *Cryptography and Network Security: Principles and Practice,* Prentice Hall
- Schneier, B. *Applied Cryptography*, New York: Wiley
- Mel, H.X. Baker, D. *Cryptography Decrypted*. Addison Wesley
- Henric Johnson, Blekinge Institute of Technology, Sweden, http://www.cs.ru.nl/J.Bruijning/

Random Numbers

The random numbers are a must in cryptography, but the generation of random number has been a recursive problem.

- **Pure random numbers** are usually based on natural phenomenas'.
- Usually the type of random numbers we use are **pseudorandom numbers**. These are generated by deterministic systems and their "randomness" sometimes is problematic when used in cryptography.

A pseudorandom sequence of numbers is **cryptographically secure** if it is difficult for an attacker to predict the next number from the numbers already in his/her possession.

Random Numbers

To be considered **truly random**, a sequence of numbers must exhibit the following two properties:

- **Uniform Distribution:** This means that all the numbers in a designated range must occur equally often.
- **Independence:** This means that if we know some or all the number up to a certain point in a random sequence, we should not be able to predict the next one (or any of the future ones).

Truly random numbers can only be generated by physical phenomena (microscopic phenomena such as thermal noise, and macroscopic phenomena such as cards, dice, and the roulette wheel).

Modern computers try to approximate truly random numbers through a variety of approaches.

Algorithmically generated random numbers are called pseudorandom numbers.

Características de um bom gerador

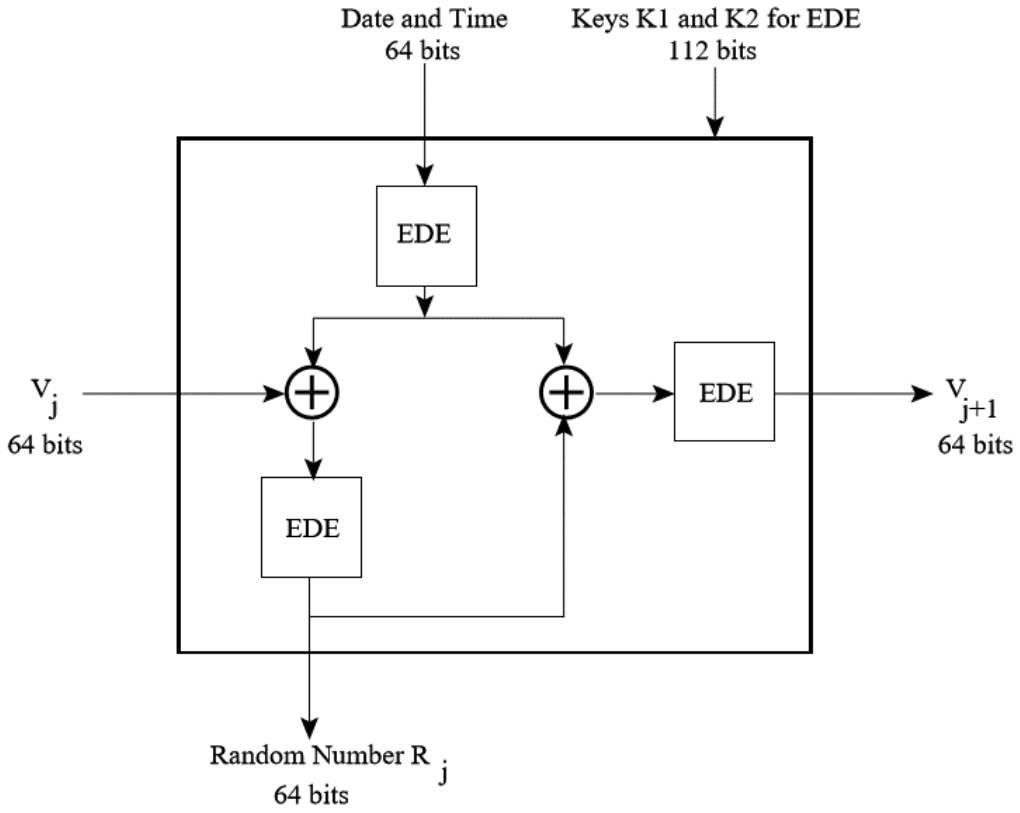
- Densidade máxima
  - Tal que os valores assumidos por R_i, $i=1,2,\ldots$, não incluam grandes intervalos iguais em *[0, 1]*
  - Problema : Em vez de continuo, cada R_i é discreto
  - Solução: Um inteiro muito grande para o módulo *m*
- Período máximo
  - Para conseguir a máxima densidade e evitar ciclos
  - Consegue-se fazendo para um gerador LCM (***Linear Congruential Method***) uma boa escolha de *a, c, m*, e X_0

$$(X_{i+1}=(a * X_i+c) \bmod m,\ \ i=0, 1, 2, \ldots \quad R_i=X_i/m,\ \ i=1, 2, \ldots)$$

- A maioria dos computadores usa uma representação binária dos números
  - A velocidade e eficiência são melhoradas escolhendo o valor para o módulo, *m*, igual ou próximo de uma potência de 2.

ANSI X9.17/X9.31 Pseudorandom Number Generator

Cryptographically secure pseudorandom number generator (CSPRNG)

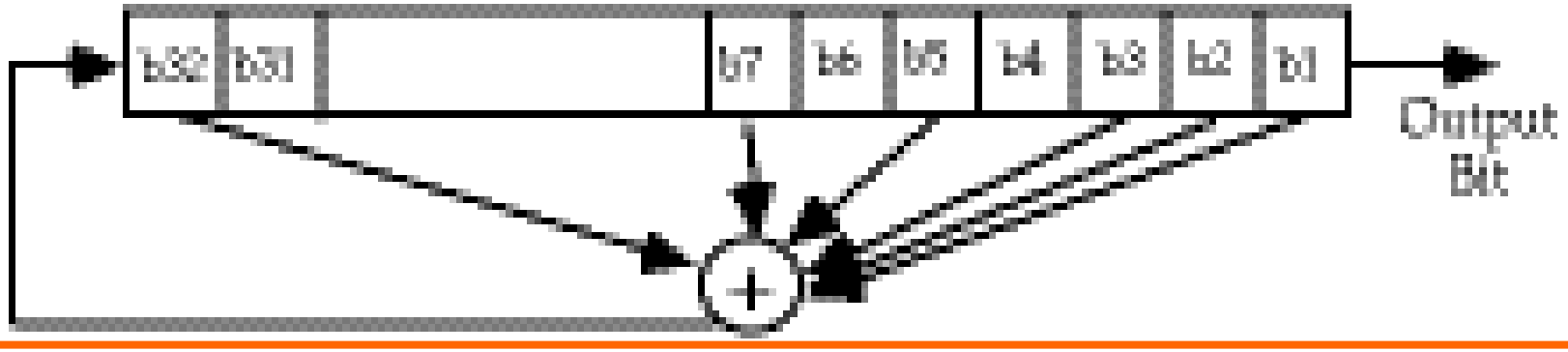


EDE - Encrypt/decrypt/encrypt (triple DES), versões mais recentes usam o AES

Linear Feedback Shift Registers

- Os *linear feedback shift registers* (LFSRs) são quase sempre propostos para serem usados como PRG (*pseudo-random generator*) em cifras sequenciais
  - Usam um *shift register* de algum comprimento
  - Cada vez que um bit é necessário todos os bits são deslocados de uma casa
  - O bit que sai é o utilizado
  - Um novo bit é criado como função linear dos outros

Exemplo



Linear Feedback Shift Registers

- A chave é o estado inicial (valor) do *shift register* e a função de realimentação usada permite que, com um *shift register* de n bits, se possam obter sequências de 2^n-1 necessárias à utilização como função polinomial primitiva linear, a qual é fácil de codificar quer em hardware quer em software mas INSEGURA pois, embora estatisticamente aleatória, é predizível apenas com 2n bits da sequência.

- Os LFSR formam a base da maioria das cifras sequenciais, apesar destas tentarem esconder a estrutura LFSR que utilizam.

Entropy of a random number generator

If we organize the bit stream produced by an entropy source into words, which could be bytes, and if we consider each such word as a random variable that can take the i^{th} value with probability p_i, we can associate the following entropy with the bit stream:

$$H = -\sum_i \; p_i \log_2 pi$$

Let's say the bit stream is organized into bytes. A byte takes on 256 numeric values, 0 through 255. If each of these values is equally probable, then $p_i = 1/256$, and the entropy associated with the entropy source would be 8 bits. This is the **highest entropy possible** for 8-bit words. If the probability distribution of the values taken by 8-bit words were to become nonuniform, the entropy will become less than its maximum value. For the deterministic case, when all the next 8-bit patterns are known apriori, **the entropy is zero**.