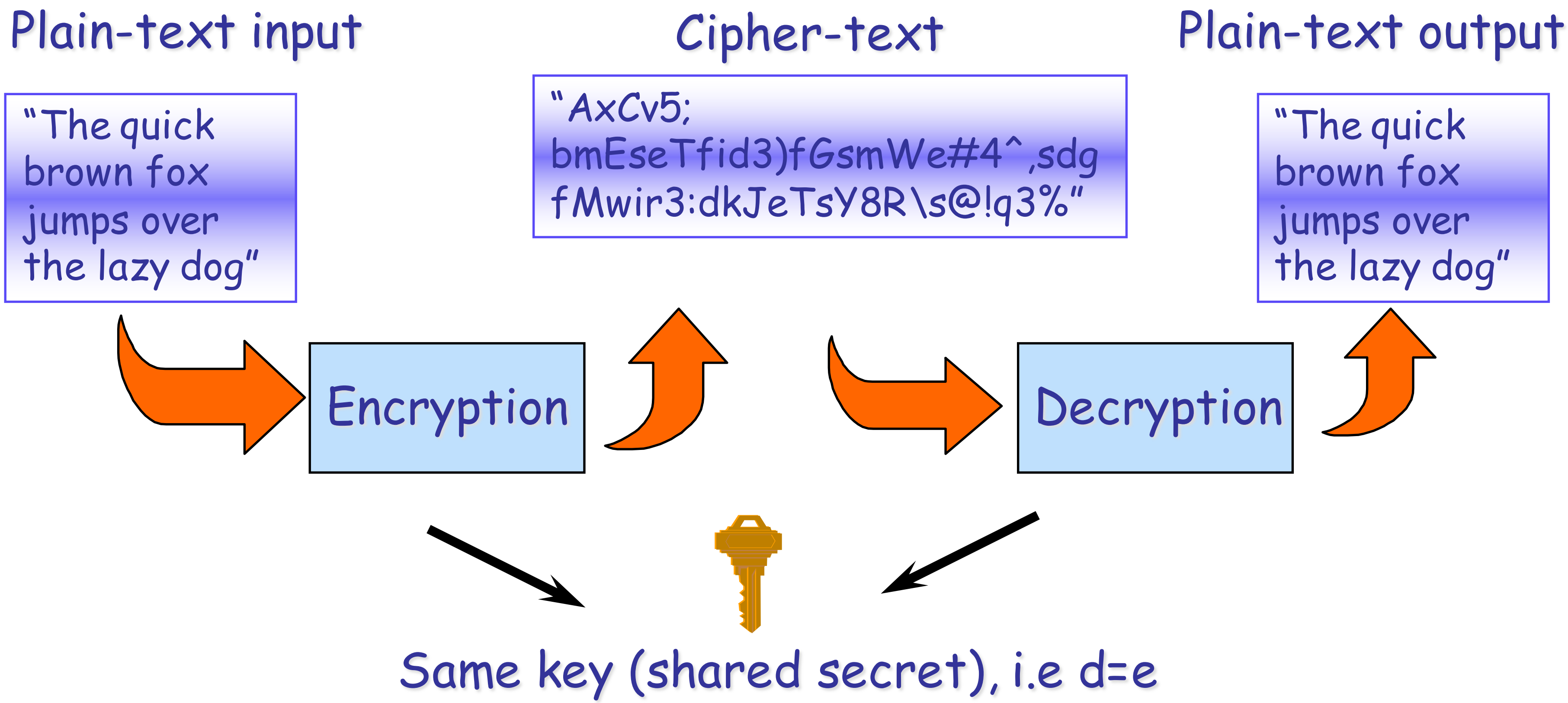
Cybersecurity

Symmetric Cryptography

Mauro Barni
University of Siena

Symmetric Cryptography



- *All classical encryption algorithms are private-key*
- It was the only type prior to invention of public-key in 1970's
- It implies a secure channel to distribute key

Symmetric Cryptography

There are two classes of symmetric-key schemes: *block ciphers* and *stream ciphers*.

- A *block cipher* is an encryption scheme which breaks up the plaintext message into strings (called *blocks*) of a fixed length *t* over an alphabet A, and encrypts one block at a time.
- A *stream cipher* is a very simple block ciphers having block length equal to one.

Symmetric Cryptography

- A *block cipher* breaks up the plaintext message into strings (called *blocks*) of a fixed length *t* over an alphabet A, and encrypts one block at a time.

THISMESSAGEISANEXAMPLEOFPLAINTETX

- A *stream cipher* is a very simple block ciphers having block length equal to one.

THISMESSAGEISANEXAMPLEOFPLAINTETX

Block ciphers

- Most well-known symmetric-key encryption techniques are block ciphers.
- Two important classes of block ciphers are **substitution ciphers** and **transposition ciphers**.
- Several such ciphers may be concatenated together to form a **product cipher**

Block ciphers

- **Substitution ciphers** are block ciphers which replace symbols (or groups of symbols) by other symbols or groups of symbols.
- **Transposition ciphers** simply permute the symbols in a block.
- These ciphers may be:
  - monoalphabetic: only one sub / transp rule used
  - polyalphabetic: more subs / transp rules used

Substitution ciphers

- ***A*** an alphabet of *q* symbols;
- ***M*** the set of all strings of length ***t*** over ***A***.
- ***K*** the set of all permutations on ***A***.
- Given $\mathbf{m} = (m_1 m_2 \ldots m_t) \in \boldsymbol{M}$,

  for each $e \in \boldsymbol{K}$ define an encryption E_e as:

$$E_e(\mathbf{m}) = (e(m_1)\ e(m_2) \ldots e(m_t)) = (c_1 c_2 \ldots c_t) = \mathbf{c}$$

- each element of m is encrypted with the same key

Substitution ciphers

- To decrypt **c** = ($c_1 c_2 \ldots c_t$) apply the inverse reordering $d = e^{-1}$

$$D_d(\mathbf{c}) = (d(c_1)\ d(c_2)\ \ldots\ d(c_t)) = (m_1 m_2 \ldots m_t) = \mathbf{m}$$

- E_e is a *mono-alphabetic substitution cipher*.
- The number of distinct substitution ciphers is $q!$ independent of the block size in the cipher.

Classical Substitution Ciphers

- We link each letter m to a number (position) i(m):

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

- A substitution cipher can be defined as:

  $c = E(m) = (i(m) + k) \bmod (26)$

  $m = D(c) = (i(c) - k) \bmod (26)$

  - k is an integer between 0 and 25
  - i(m) indicates the position of letter m
  - (i(m) + k) mod (26) is the remainder of (i(m) + k)/26

Caesar Cipher

- Earliest known substitution cipher
- Invented by Julius Caesar
- First attested use in military affairs
- Replaces each letter by 3rd letter on
- Example:

```
meet me after the toga party
PHHW PH DIWHU WKH WRJD SDUWB
```

Caesar Cipher

- Can define transformation as:

```
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
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
```

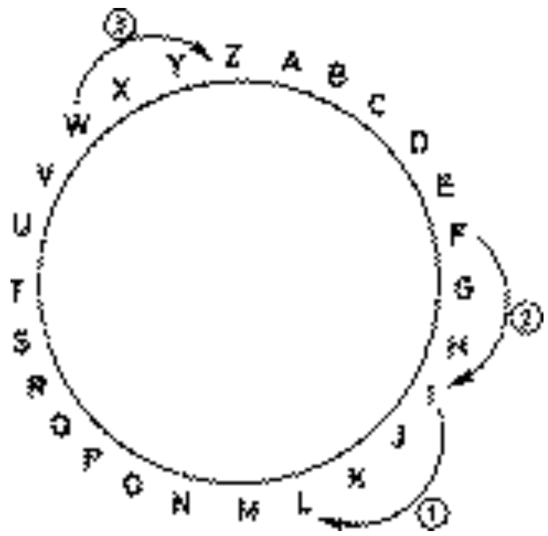
- Caesar cipher defined as:

  $c = E(m) = (i(m)+ 3) \bmod (26)$

  $m = D(c) = (i(c) - 3) \bmod (26)$

  e.g.

I CAME I SAW I CONQUERED
L FDPH L VDZ L FRQTXHUHG



Cryptanalysis of Caesar Cipher

- There exist only 26 possible ciphers
  - A maps to A, B ... Z
- Could simply try each in turn with a **brute force search**
- Given ciphertext, just try all shifts of letters
- Need to recognize a meaningful plaintext
  - This may be difficult, for instance, with zipped files (if you do not know that they are zipped)

Cryptanalysis of Caesar Cipher

As mentioned before, we do need to be able to recognise when a meaningful message is obtained

Usually easy for humans, hard for computers

Though if using say compressed data could be much harder or easier

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Figure 2.3 Brute-Force Cryptanalysis of Caesar Cipher

Monoalphabetic Cipher

- Rather than just shifting the alphabet we could shuffle (jumble) the letters arbitrarily

=> each plaintext letter maps to a random ciphertext letter

- plain alphabet mapped into a ciphered one:

```
Plain alphabet : abcdefghijklmnopqrstuvwxyz
Cipher alphabet: DKVQFIBJWPESCXHTMYAUOLRGZN
```

- Example

```
Plaintext:  ifwewishtoreplaceletters
Ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA
```

Monoalphabetic Cipher

- The key is the mapping between plain alphabet and ciphered alphabet
- If the alphabet is q=26 letters long:
  - **we have a total of 26! = 4 x 10^{26} possible keys**
- The simplicity and strength of the monoalphabetic substitution cipher made it the most used system for the first millenium.

Cryptanalysis

- With so many keys, you might think it is secure
- **WRONG !!** The problem are language characteristics, if the known language is used

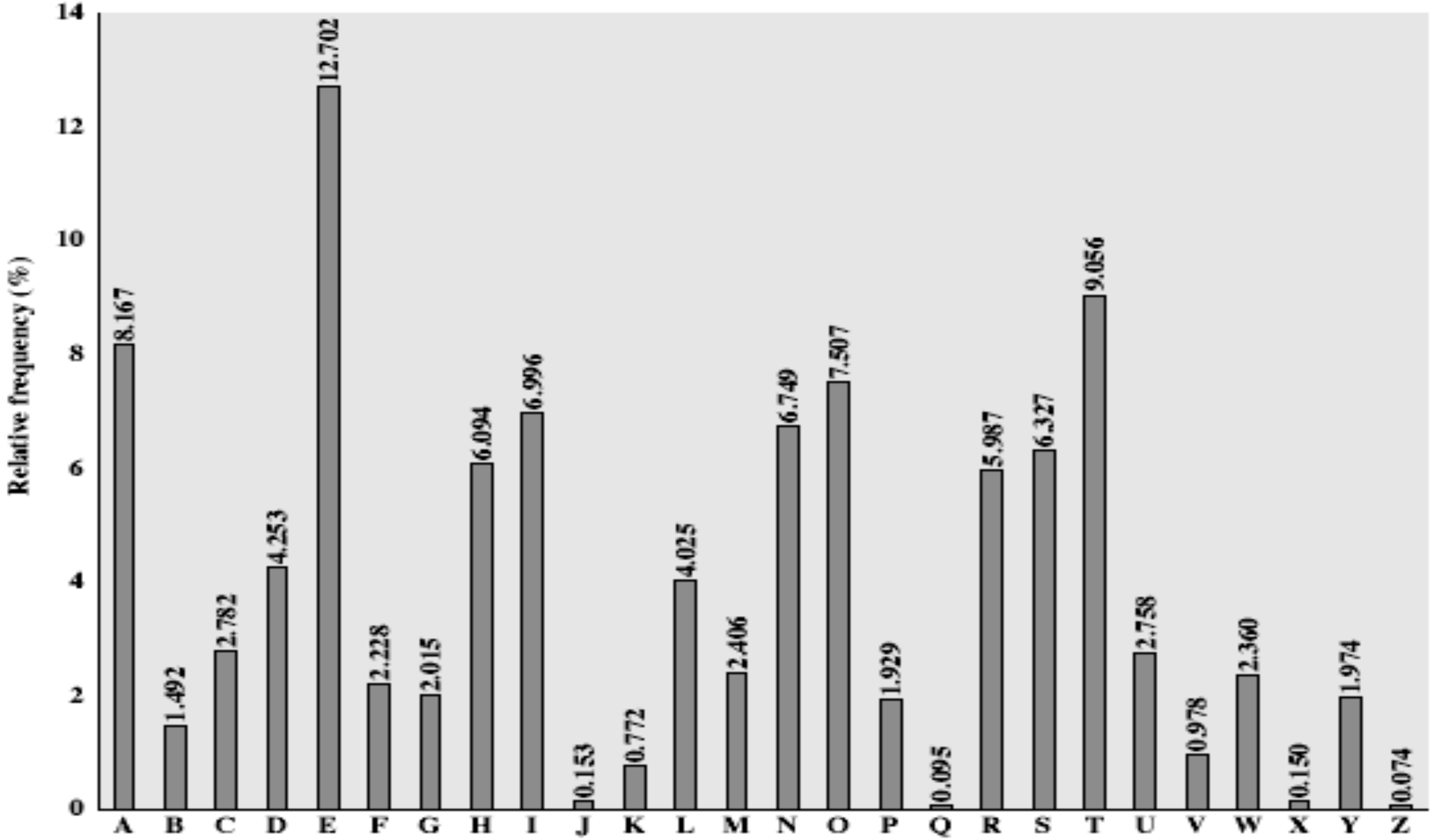
It was broken by Arabic scientists.

The earliest known description is in Abu al-Kindi's "A Manuscript on Deciphering Cryptographic Messages", published in the 9th century but only rediscovered in 1987 in Istanbul

Language Redundancy and Cryptanalysis

- Letters are not equally used:
  - in English E is by far the most common letter
  - then T,R,N,I,O,A,S
  - other letters are fairly rare (cf. Z,J,K,Q,X)
- Tables of single, double & triple letter frequencies are available

English Letter Frequencies



Italian Letter Frequencies

Cryptanalysis

- Key concept - monoalphabetic substitution ciphers do not change letter frequencies

- An attacker can:
  - calculate letter frequencies for ciphertext
  - compare counts/plots against known values
  - If needed, do that with digrams and trigrams

Cryptanalysis of substitution cipher

- The ***A*** alphabet is the Italian alphabet.
  - Three letters, a e i, have a frequency > 10%;
  - Four letters b, f, q, z have frequencies < 1%.
    - If the ciphertext does not exhibit these peaks is not Italian
    - If still can not decipher look at double letters: ll, tt, ss, cc, rr, gg, pp, nn.
  - If there are spaces between letters try to decrypt small words. In italian the only words of one letter are a, e, i, o

Other solutions

- To cope with the above vulnerabilities we can:
  - encrypt more letters together
    - Playfair cipher
  - use polyalphabetic ciphers
    - Vigenère cipher

Playfair Cipher

- Plaintext is encrypted two letters at a time
- Based on a 5x5 matrix dependent on a secret keyword
  - fill in, left to right, top to bottom, the letters of keyword (no duplicates), next all remaining letters in alphabetical order (i and j together)
- For example, if the key is MONARCHY

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Cipher - encryption

- Divide the plaintext in blocks of 2 letters;
  - if a pair is a repeated letter, insert a filler like ‘x',
    eg. "balloon" encrypts as "ba lx lo on"
- If both letters fall in the same row, replace each letter with the letter to its right (wrapping back to start from end)

eg. “**AR**” -> “**RM**”

M	O	N	**A**	**R**
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

→

M	O	N	A	**R**
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Cipher - encryption

- If both letters fall in the same column, replace them with the letter below them (again wrapping to top from bottom)

eg. "MU" -> "CM"

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Cipher - encryption

- Otherwise each plaintext letter of the pair is replaced by the letter in the same row, and in the column of the other letter of the pair.

"HS" -> "BP"

"EA" -> "IM"

M	O	N	A	R
C	**H**	Y	B	D
E	F	G	I	K
L	P	Q	**S**	T
U	V	W	X	Z

→

M	O	N	A	R
C	H	Y	**B**	D
E	F	G	I	K
L	**P**	Q	S	T
U	V	W	X	Z

- Decryption works in the reverse way

Security of Playfair Cipher

- Much better security
- Since we have $26^2 = 676$ digrams
  - much longer ciphertexts needed for cryptanalysis
- A 676 digram frequency table to analyse (vs. 26 for a monoalphabetic) is needed (but it exists!)

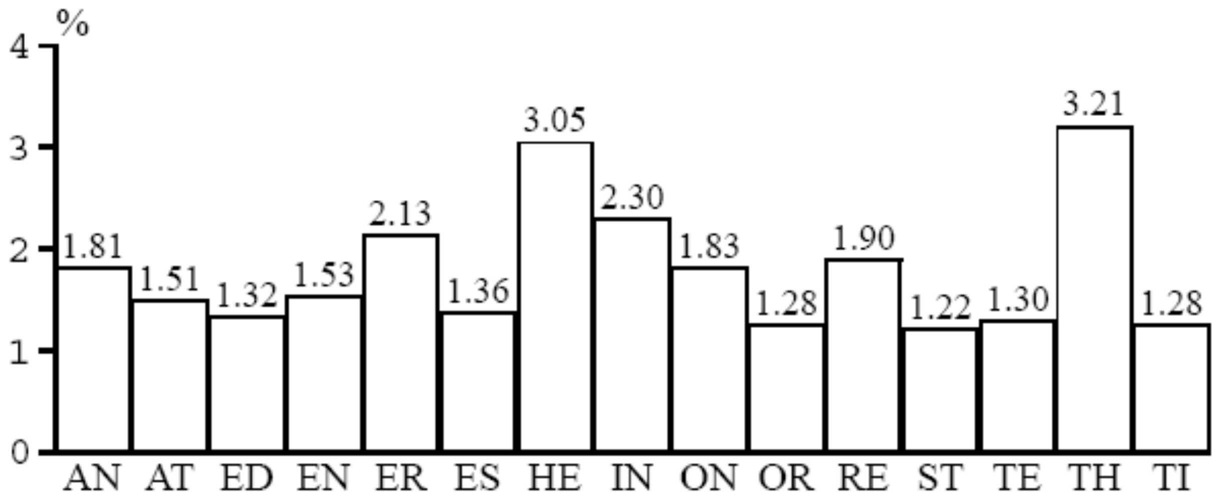


Figure 7.6: *Frequency of 15 common digrams in English text.*

Security of Playfair Cipher

- It would need correspondingly more ciphertext
- It was widely used for many years
  - eg. US & British army in WW1
- It **can** be broken, given a few hundred letters since still has much of plaintext structure

Polyalphabetic Ciphers

- Another approach to improve security is to use polyalphabetic substitution ciphers, based on:
  - set of monoalphabetic substitutions
  - a key to select which substitution rule is used for each letter of the message
- Makes cryptanalysis harder with more alphabets to guess and flatter frequency distribution
- A quite famous polyalphabetic substitution cipher is the Vigenère Cipher

Letter frequencies in ciphertexts

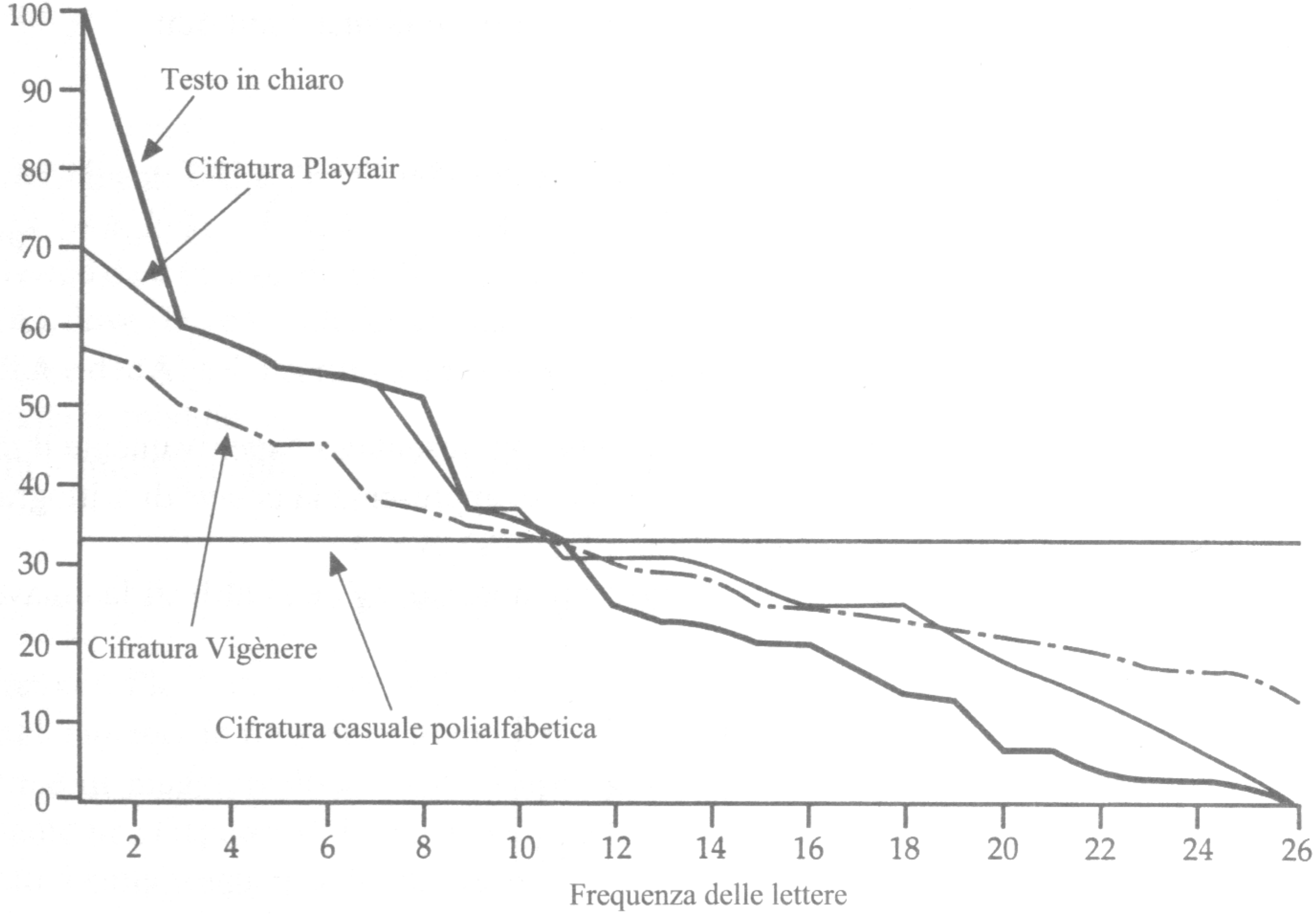


Figura 2.6 Frequenza relativa della presenza di lettere.

Vigenère Cipher

- It uses 26 Caesar ciphers (shifts from 0 to 25) chosen according to a key
- Key is a keyword $K = [k_1\ k_2\ ...\ k_d]$
- i^{th} letter specifies i^{th} alphabet to use
  - e.g. letter c indicates shift of 2

- Decryption works in reverse order

Example

- Write the plaintext
- Write the repeated keyword above it, so that the lenght of the key and of the plaintext are the same
- Use each key letter as a Caesar cipher key (e.g. the key letter b corresponds to a shift of 2 positions).
- Encrypt the corresponding plaintext letter
- E.g. using keyword *deceptive*

```
key:        deceptivedeceptivedeceptive
plaintext:  wearediscoveredsaveyourself
ciphertext:ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

Vigenère table

plaintext

key

	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

Vigenère table - encryption

```
key:          deceptivedeceptivedeceptive
plaintext:    wearediscoveredsaveyourself
ciphertext:   ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

plaintext

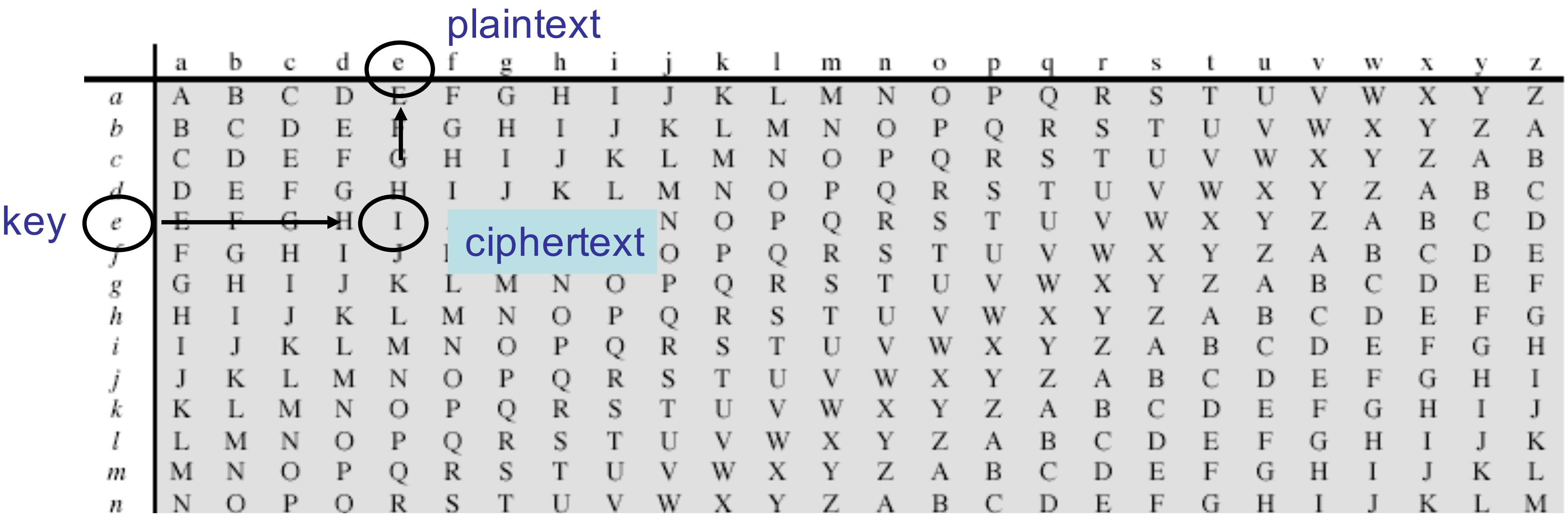
key

	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	[illegible]	[illegible]	[illegible]	[illegible]	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	[illegible]	[illegible]	[illegible]	[illegible]	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

ciphertext

Vigenère table - decryption

```
Key:        deceptivedeceptivedeceptive
Ciphert.:   ZICVTWQNGRZGVTWAVZHCQYGLMGJ
Plaint.:    wearediscoveredsaveyourself
```



	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	[illegible]	[illegible]	[illegible]	[illegible]	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	[illegible]	[illegible]	[illegible]	[illegible]	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

Security of Vigenère Cipher

- For each plaintext letter there are multiple ciphertext letters
- Letter frequencies are obscured, but not totally lost
- Attack starts with letter frequencies
  - see if looks monoalphabetic or not -> study the frequency histogram
  - if not, the cipher is a polyalphabetic one, then attack to determine the number of alphabets (key length)

Determine key-length

- By relying on letter frequency we can measure how english-like is a ciphertext (Index of coincidence: I.C.)
- We exhaustively try all key length until I.C. of all subsequences is large enough

Determine key-length

ciphertext

vptnvffuntshtarptymjwzirappljmhhqvsubwlzzygvtyitarptyiougxiuydtgzhhvv
mumshwkzgstfmekvmpkswdgbilvjljmglmjfqwioiivknulvvfemioiemojtywdsa
jtwmtcgluysdsumfbieugmvalvxkjduetukatymvkqzhvqvgvptytjwwldyeevqu
hlulwpkt

with keylength = 2:

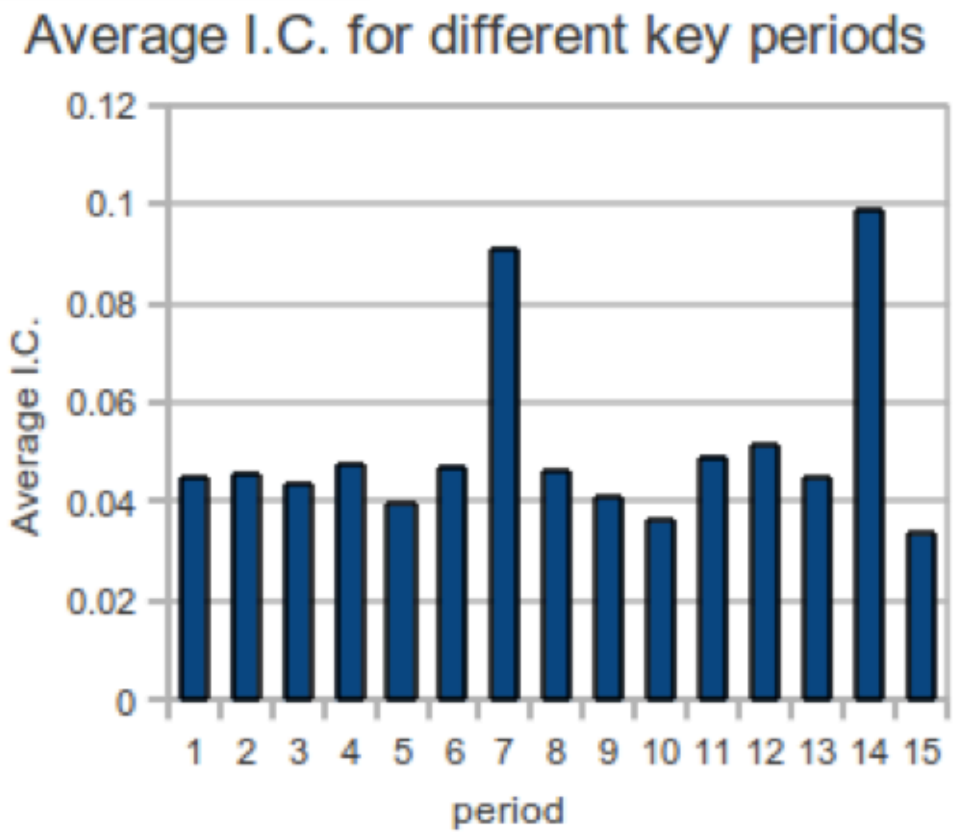
sequence 1:	v t v f n s t r t m w i a p j h q s b ...	0.049
sequence 2:	p n f u t h a p y j z r p l m h v u w...	0.046
		average: 0.048

with keylength = 3:

sequence 1:	v n f t t p m z a l h v b ...	0.049
sequence 2:	p v u s a t j i p j h s w...	0.046
sequence 3:	t f n h r y w r p m q u ...	0.046
		average: 0.047

Determine key-length

period	avg I.C.
1 :	0.0449443523561
2 :	0.0457833618884
3 :	0.0435885364312
4 :	0.0474962292609
5 :	0.0393612078978
6 :	0.0471437059672
7 :	0.0909922589726
8 :	0.0461858974359
9 :	0.0407804755631
10:	0.0361152882206
11:	0.0491603339901
12:	0.0512663398693
13:	0.0446886446886
14:	0.0988487702773
15:	0.0334554334554



Security of Vigenère Cipher

- Now that we know that the key-length is 7, we need to break 7 Ceasar’s cyphers
- We can do that by analyzing separately 7 subsequences extracted from the ciphertext
- Difficult if ciphertext is not long enough (w.r.t. key-length)

Security of Vigenère Cipher

- To remove the periodic nature of the key, it is possible to use a key sequence obtained through a concatenation of the key + the plaintext.

```
key:        deceptivewearediscoveredsav
Plaintext:  wearediscoveredsaveyourself
```

- Also this method, though more secure, is vulnerable to cryptoanalysis

Transposition ciphers

- Transposition ciphers permute the symbols present in a plaintext block (length **t**).
- ***A*** alphabet of *q* symbols,
- ***M*** set of all strings of length **t** over ***A***.
- ***K*** set of all permutations on the set {1,2,…,t}.
- Define for each e $\in$ ***K*** an encryption transformation E_e as:

$$E_e(m) = (m_{e(1)}\ m_{e(2)} \ldots m_{e(t)}) = (c_1 c_2 \ldots c_t) = c$$

- where $m = (m_1 m_2 \ldots m_t) \in M$.

Transposition ciphers

- The set of all such transformations is called a *transposition cipher*.
- To decrypt $c = (c_1c_2\ldots c_t)$ compute the inverse permutation $d = e^{-1}$ and $D_d(c) = (c_{d(1)}\ c_{d(2)} \ldots\ c_{d(t)}) = (m_1m_2\ldots m_t) = m$.
- A transposition cipher described as above preserves the number of symbols of a given type within a block, and thus is easily cryptanalyzed.

Example 1: Rail Fence Cipher

- Write message letters out diagonally (up and down) over a fixed number of rows
- Then read off cipher row by row
- E.g. write message `meet me after the toga party` with a rail fence of depth 2 as:

```
m e m a t r h t g p r y
 e t e f e t e o a a t
```

- giving the ciphertext

MEMATRHTGPRYETEFETEOAAT

Rail Fence Cipher

- Decrypting the message is easy if the row boundaries are known.
- Just write down the rows of `MEMATRHTGPRYETEFETEOAAT` in order:

```
MEMATRHTGPRY
 ETEFETEOAAT
```

- and reconstruct the "rails" of the fence:
- `meet me after the toga party`

Row Transposition Cipher

- Write the letters of the message row by row over a specified number of columns
- Reorder the columns according to some key before reading the matrix by column:

```
Key:         4 3 1 2 5 6 7

Plaintext:   a t t a c k p
             o s t p o n e
             d u n t i l t
             w o a m x y z

Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ
```

Exercise of cryptanalysis

- The following ciphertext is written with the English alphabet as **A**.
- Substitution or transposition cipher

- Try it!

Homework - ciphertext

TOTETHEOERNETIOFHIARSUGSTOTAANSFUSBPNDMISPONAESETEAAHONTLCHL
IITCUTDUTWDISPLPHERAETURHLOAHRSCWWVUEFSTOUISSETECAKAIFOIOOLAE
PDRFEOEROHOAOMTAOSEVESAEOCOCDSNATTITNTAWHMFHSEMWAEKHUALAG
TSTEELBHHEFENTETNCRUYMSUOVETSZTINKSHEHIWVAYTSTNOTCCCEKWSSAH
HTHFOISKDWTAAFUANPEGTHMNTIGTRRSNYLTAFIOOWFOLYIYSBLIEBOEROTBTA
ITEUSINHTETSOLRNHMNTSFETELNSNARWOOTAEUFRUEROAERSGISAEOTOBEA
DYPOIGNTETDEOLENMRADYSEPOAWEDHHATCENTEHUADAUASOKTAFEHSEROI
AOSMAINEOTYOEIHTDEOLETSEPECACTDEMYHRSHRBOITASEPFETWADEMMYOE
HNEAEHFLDFTIMRACIMSGVUPUEHRSHRSETHTAECLMTOSLNLFFRHWUDERHWIS
NSONOTMTEPRSOSRNTERUMNCNUEYHPNSFEPSDOEHLWDLYHISLNEFFIENTEPR
SHTAINMRTFHUWRHTKSHNEISLMGTIQITSAEIHBRBDIWOOLFRESERORNADWAUD
RWAYIEUTATERAOSMTIGFEDAHHUDSOEDONRFOWOEORNTAELREUNPZLSHWLA
DAEURTEBATOELSEAEHNLTOHRTAWKONTFHSOSINEOSAEOADOUALNTUTEAIEU
ORSLTOISCLEORIHHPLCSOTOGTNETRRSSFRAPTADOETIHHSEADHICRETTRARA
DOEHNMOATOSFYUOTEARPEINMHNHOIOSELMSNRMMEDBNOHSQTWHIBIEDURS
GDOFROONTKMATARLNOSEHOTEOEBLTYNERHDTSNRHSTSHTSNMOVLBSOTEO
ERNOATEEFNTEDHTAAMEHSFOHOLLTEATEEPTMSAYOGEWOBTHACSIHPSWGPD
STLEGDILTAETNEOFAHUTPEEOEOYEEHEIHUUKTAONWDDBTUNENARFTTDDOHA
RTEIVCTRHBNRLRRUEELMSARRSLHTFOEHEWOUNEDMCRFLDSNVEEUNIIETEET
HHDEIOEINMWTRREUNUWNSEECNTNHIHAPTRNAYSER

Rotor Machines

- Before modern ciphers, rotor machines were the most commonly used ciphers
- They were widely used in WW2
  - German Enigma, Allied Hagelin, Japanese Purple
- **They implemented a very complex, time varying substitution cipher**
- Used a series of cylinders, each giving one substitution, which independently rotated after each letter was encrypted.

Enigma machine

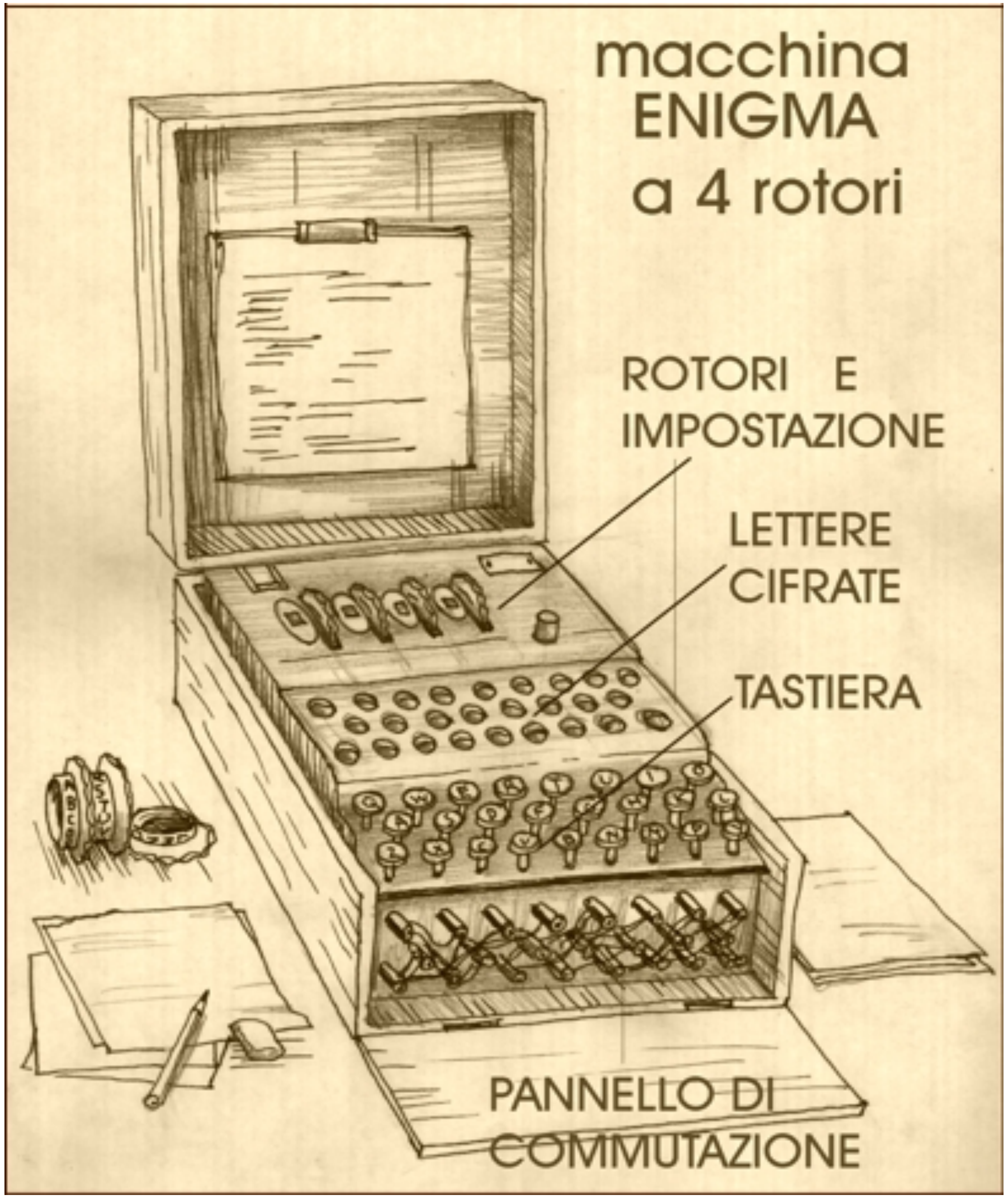
It was invented by Arthur Scherbius in 1918, to tackle with industrial espionage.

First sample was shown in 1923 at the International Postal Union Congress, raising the interest of German, Japanese and USA army.

Since 1929 was adopted by German army and extensively used until WW2.

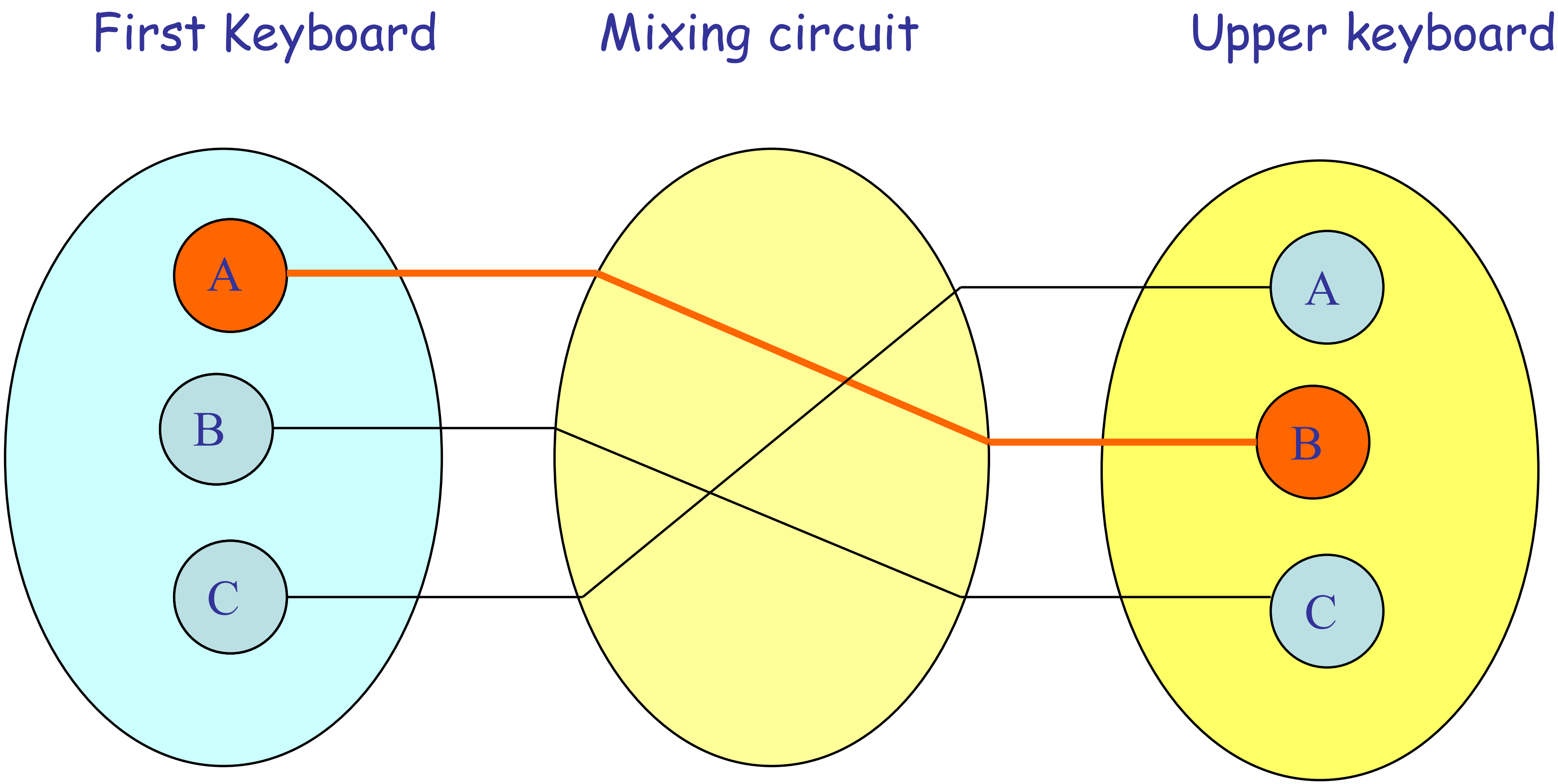
Enigma machine

- The Enigma machine provides an electro-mechanical implementation of a symmetric encryption and decryption algorithm
- It looks like a normal typewriter
- It has two keyboards: pressing a letter in the lower keyboard the letter highlighted in the upper keyboard is printed



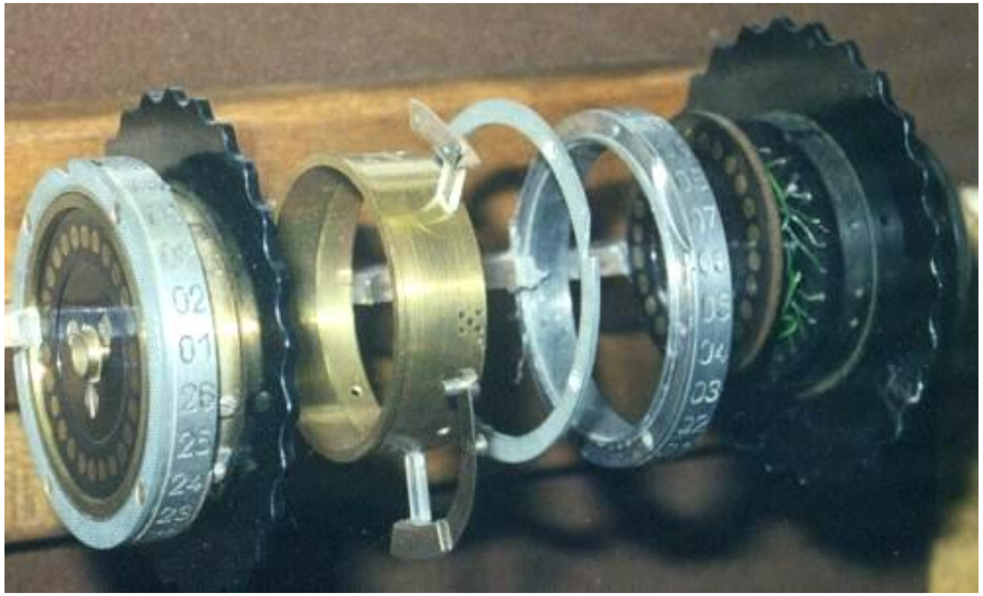
Enigma machine

- Basic idea:



The mixing circuit

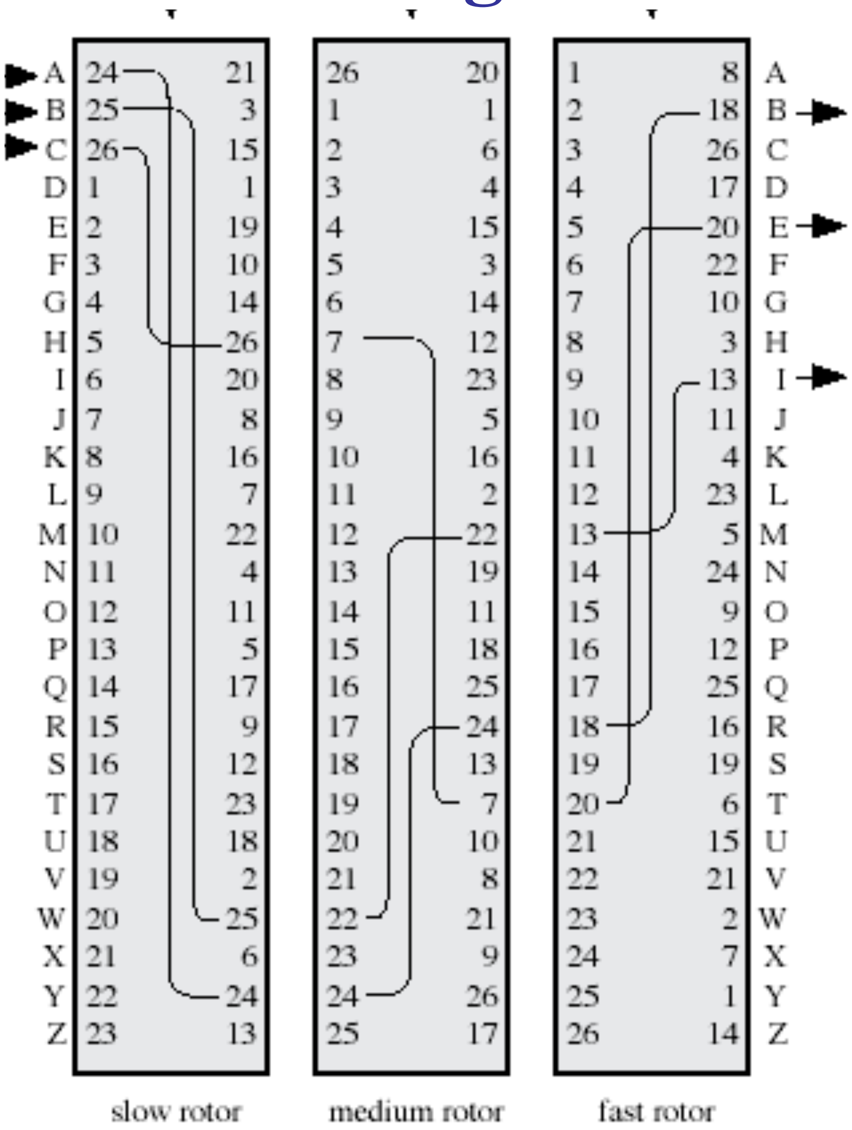
- The mixing circuit is obtained by means of electro-mechanical rotors -> thick rubber disks linked to the keyboard
- The rotor defined a simple mono-alphabetic substitution cipher



The mixing circuit

- After each stroke the rotor rotated by 1/26 thus defining a different substitution cipher
- In this way the Enigma machine implemented a multiple-alphabet substitution cipher

The mixing circuit



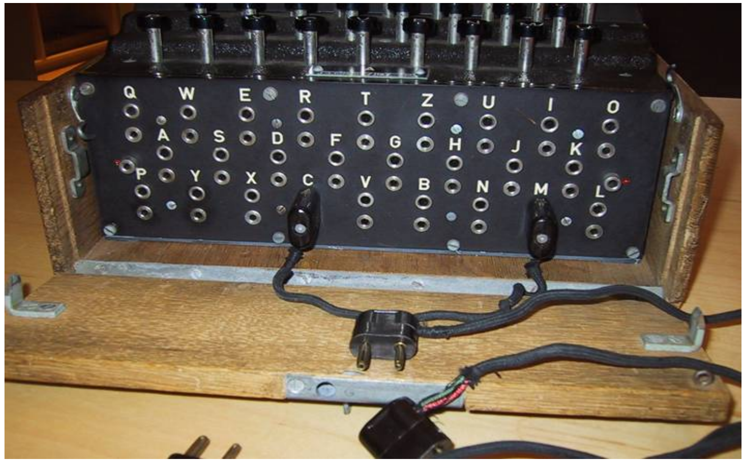
- To avoid cryptanalysis based on period analysis, Enigma adopted three cascaded rotors. The second rotated by one step only after that the first had completed its cycle, the third after a complete cycle of the second
- In this way the Enigma allowed for 26x26x26 = 17,756 different alphabet substitutions

Removable rotors

- By allowing the substitutions of rotors the number of possible substitutions increased further
- Given three rotors that can be mounted in 6 different ways, the key space increases by a factor 6
- Later on the number of rotors was increased from 3 to 5

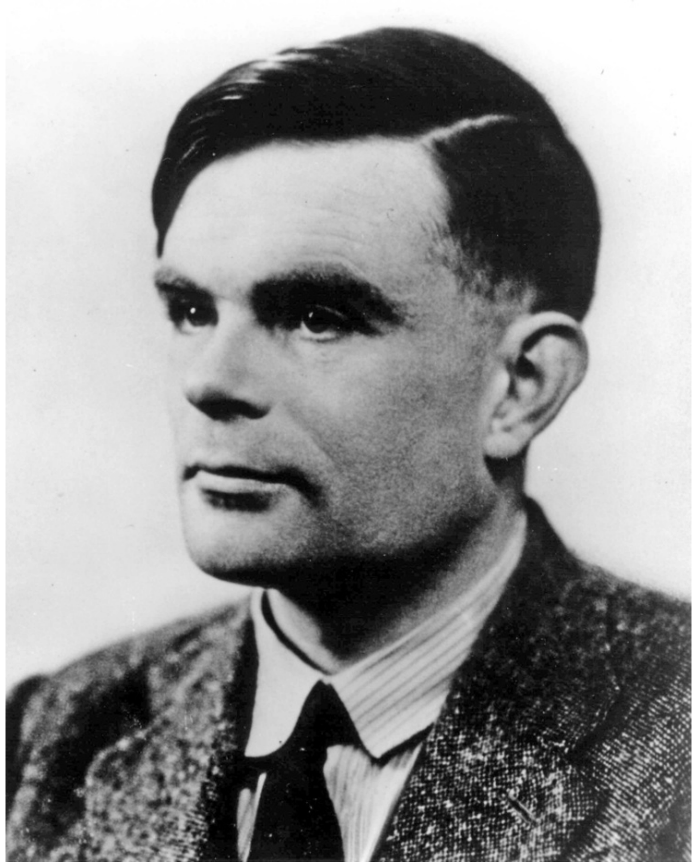
An additional plugboard

- A plugboard in the bottom part of the machine permitted to further enlarge the keyspace
- By connecting two letters with a patchcord the meaning of a letter was changed
- If the letters Q and R were connected, typing Q was equivalent to typing R, thus producing at the output the encryption of R



Breaking Enigma

- A group of English mathematicians guided by Alan Turing, gathered at Bletchley Park.
- After several efforts they managed to break the Enigma code, which was thought to be 100% secure by the Germans.

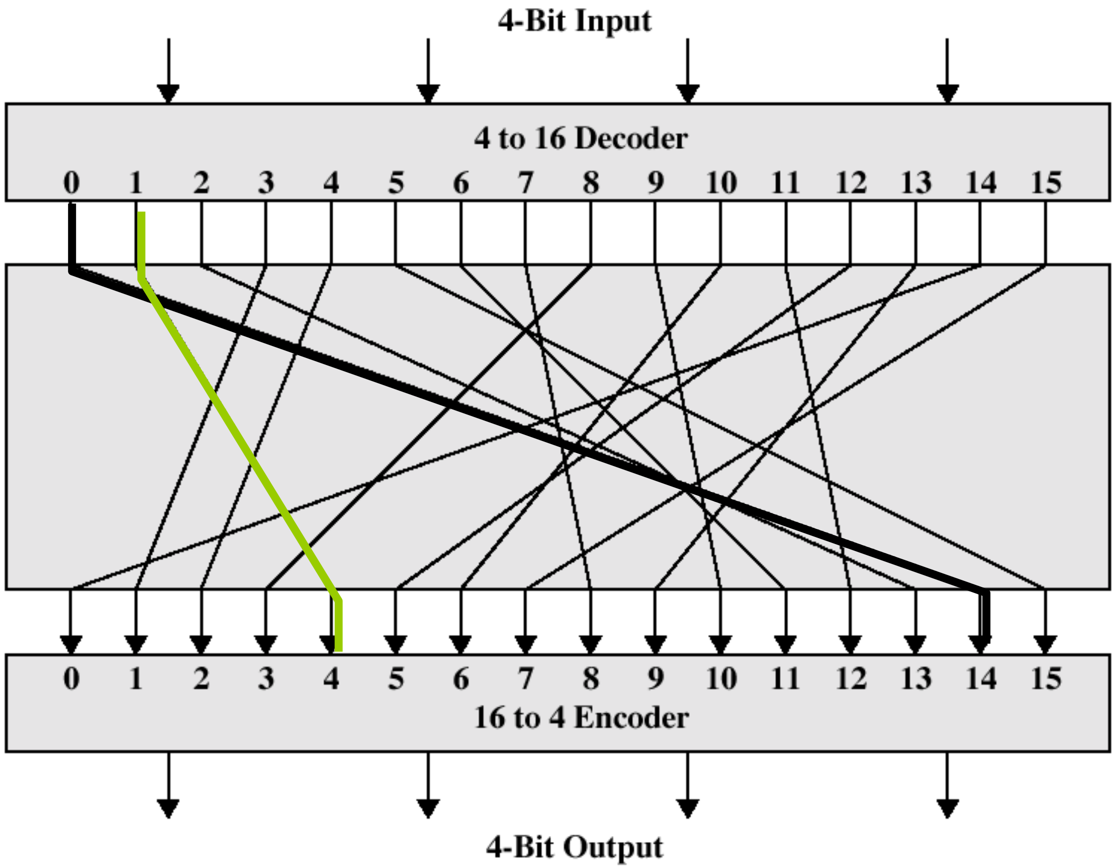


Modern Block Ciphers

- Will now look at modern block ciphers
- Most widely used types of cryptographic algorithms
- In particular: DES (Data Encryption Standard)
- DES is more and more often replaced by AES
  - Consider only DES for simplicity

Block Cipher Principles

Block ciphers look like an extremely large substitution.
Would need a table of 2^n entries for an n-bit block



Plaintext	ciphertext
0 = 0000	14 = 1110
1 = 0001	4 = 0100
2 = 0010	13 = 1101
3 = 0011	1 = 0001
4 = 0100	2 = 0010
5 = 0101	15 = 1111
6 = 0110	11 = 1011
7 = 0111	8 = 1000
8 = 1000	3 = 0011
9 = 1001	10 = 1010
10 = 1010	6 = 0110
11 = 1011	12 = 1100
12 = 1100	5 = 0101
13 = 1101	9 = 1001
14 = 1110	0 = 0000
15 = 1111	7 = 0111

Block Cipher Principles

- Not efficient: mapping is defined by a key, that is by the sequence of ciphertext bits => here 64 bits.
- 1110 010011010001…
- In general the key length would be equal to $\mathbf{n*2^n}$
- If n=64 => length= $64*2^{64}$= 10^{21} bits
- **We need an algorithmic definition**

Plaintext	ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Block Cipher Principles

- Most symmetric block ciphers are based on a **Feistel Cipher Structure**,
- Feistel (Horst Feistel, 1915-1990) proposed to decompose encryption into smaller transformations, using the idea of a product cipher (sequences of substitutions and permutations).
- Introduced in 1973, it applies Shannon's studies*

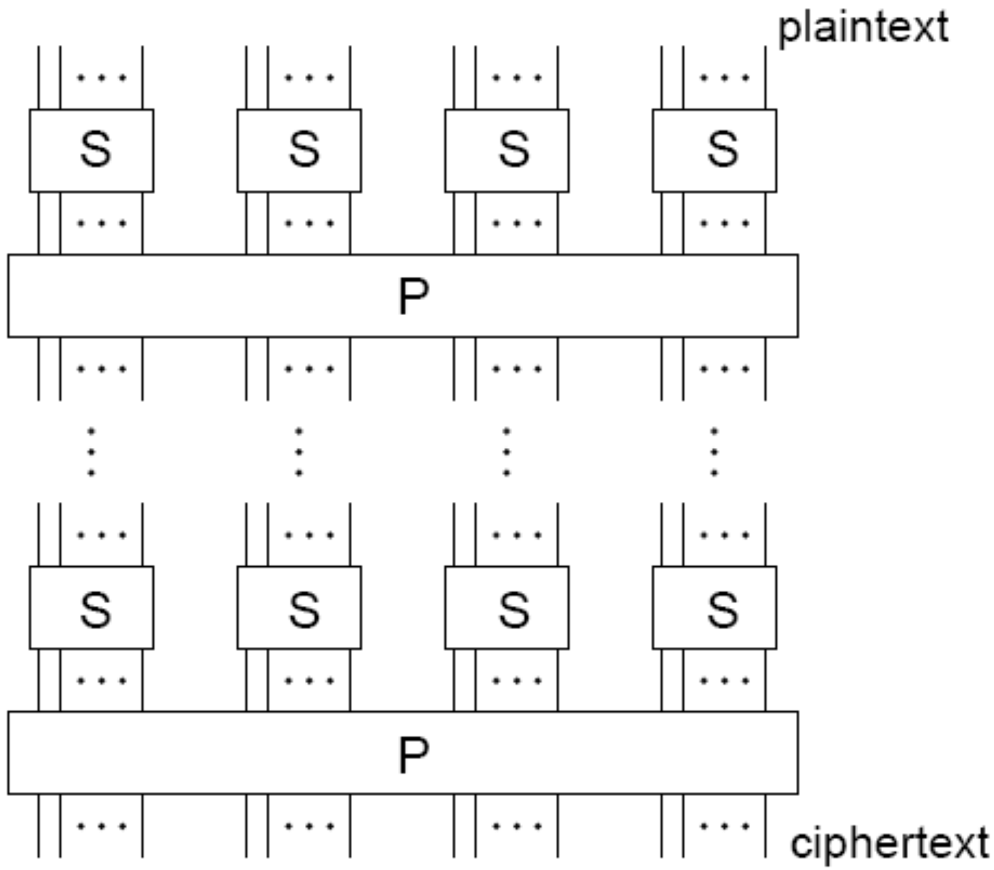
* Claude E. Shannon, "Communication Theory of Secrecy Systems", Bell System Technical Journal, vol.28-4, page 656--715, 1949.

Substitution-Permutation Ciphers

- In 1949 C. Shannon introduced the idea of substitution-permutation (S-P) networks, forming the basis of modern block ciphers
- S-P networks are based on the two cryptographic primitives we have seen before:
  - *Substitution* (S-box)
  - *Permutation* (P-box)

Substitution-Permutation Ciphers

A *substitution-permutation* (SP) *network* is a product cipher composed of a number of stages each involving substitutions and permutations



Confusion and Diffusion

- According to Shannon’s studies a secure cipher needs to completely obscure the statistical properties of the original message to avoid statistical attacks!
- In practice Shannon suggested that each cipher should aim at the following two goals:
  - **Confusion (substitution)**
  - **Diffusion (permutation)**

Confusion

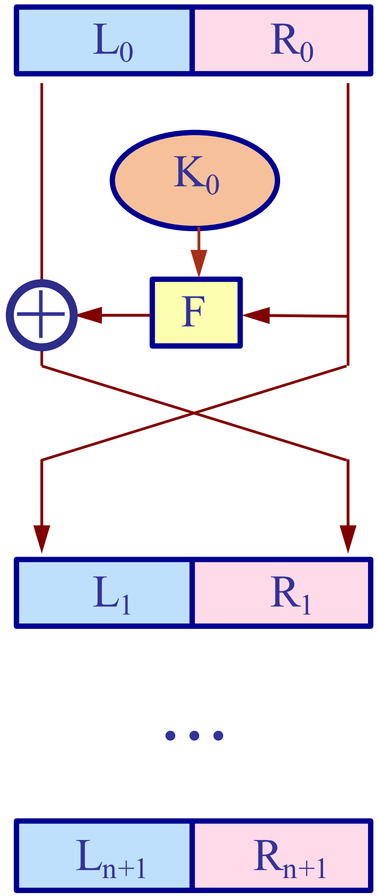
- **What is it ?**
  - It refers to making the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, to thwart attempts to discover the key
- **How to achieve it**
  - It is obtained through several substitutions;
  - The desired effect is that the knowledge of the statistics of the ciphertext does not help to find the key.

Diffusion

- **What is it ?**
  - Aims at making the statistical relationship between the plaintext and ciphertext as complex as possible.
  - It means spreading out of the influence of a single plaintext digit over many ciphertext digits.
    - flipping an input bit should change each output bit with a probability of one half (avalanche effect)
- **How to achieve it**
  - through several permutations + a function applied to the result, so that several bits of plaintext contribute to the creation of a single bit in the ciphertext and viceversa

Feistel Cipher Structure (encryption)

- Horst Feistel devised the **Feistel Cipher,** a structure which adapted Shannon's S-P network in an easily invertible structure
- Same h/w or s/w is used for both encryption and decryption, with just a change in how the keys are used. F function does need to be invertible but must be properly designed for security reasons



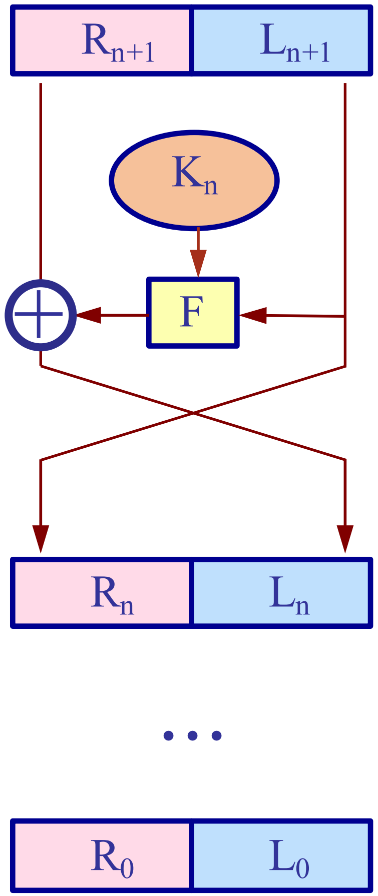
$L_{i+1} = R_i$

$R_{i+1} = L_i \text{ XOR } F(R_i,K_i)$

Key composed of n sub-keys derived from K of the same length of L_0 and R_0

Feistel Cipher Structure (decryption)

- Decryption is obtained by applying the same structure to the ciphertext
- In fact, each step can be reverted since half of the input is not changed and used to encrypt the second part
- We only need to swap the right and left parts and apply the same circuit with keys in reversed order
- At the end the two parts need to be swapped again



$$R'_{i-1} = L_i = R_{i-1}$$

$$\begin{aligned} L'_{i-1} &= R_i \text{ XOR } F(L_i, K_{i-1}) \\ &= R_i \text{ XOR } F(R_{i-1}, K_{i-1}) \\ &= L_{i-1} \text{ XOR } F(R_{i-1}, K_{i-1}) \text{ XOR } F(R_{i-1}, K_{i-1}) \\ &= L_{i-1} \end{aligned}$$

Feistel Design Principles

- Block size
  - increasing size improves security, slows cipher (64, 128)
- Key size
  - increasing size improves security, makes exhaustive key searching harder, but may slow cipher (64, now 128)
- Number of rounds
  - increasing number improves security, slows cipher (16)
- Subkey generation
  - greater complexity makes analysis harder, but slows cipher
- Round (F) function
  - greater complexity makes analysis harder, but slows cipher

Data Encryption Standard (DES)

- The most widely used block cipher in world
- Adopted in 1977 by National Bureau of Standards
  - now Nat. Inst. of Standards and Technology (NIST)
- It encrypts 64-bits of data by using a 56-bit key
- Now replaced by AES

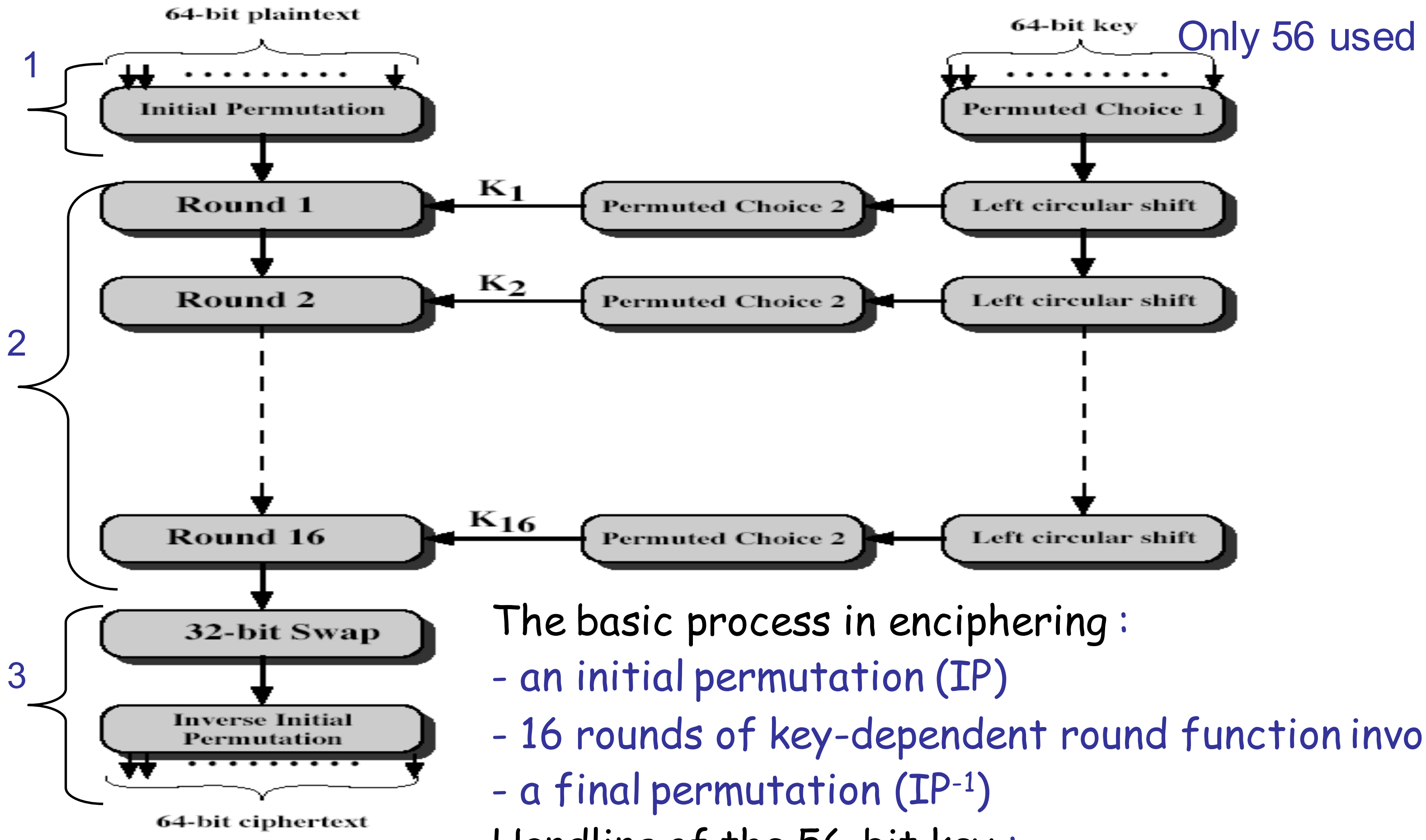
DES History

- IBM designed the Lucifer cipher
  - team led by Feistel
  - used 64-bit data blocks with 128-bit key
- then redeveloped as a commercial cipher implemented in a chip => keys 56 bits
- in 1973 NBS issued request for proposals for a national cipher standard
- IBM submitted their revised Lucifer which was eventually accepted as the DES.

DES Design Controversy

- DES standard is public
- There was considerable controversy over design
  - choice of 56-bit key (vs Lucifer 128-bit)
  - and because design criteria (S-box) were classified
- Subsequent public analysis showed the design was appropriate
- DES was widely used (financial applications)
- After DES, Triple DES was introduced, but also this has been abandoned in favor of AES

DES Encryption



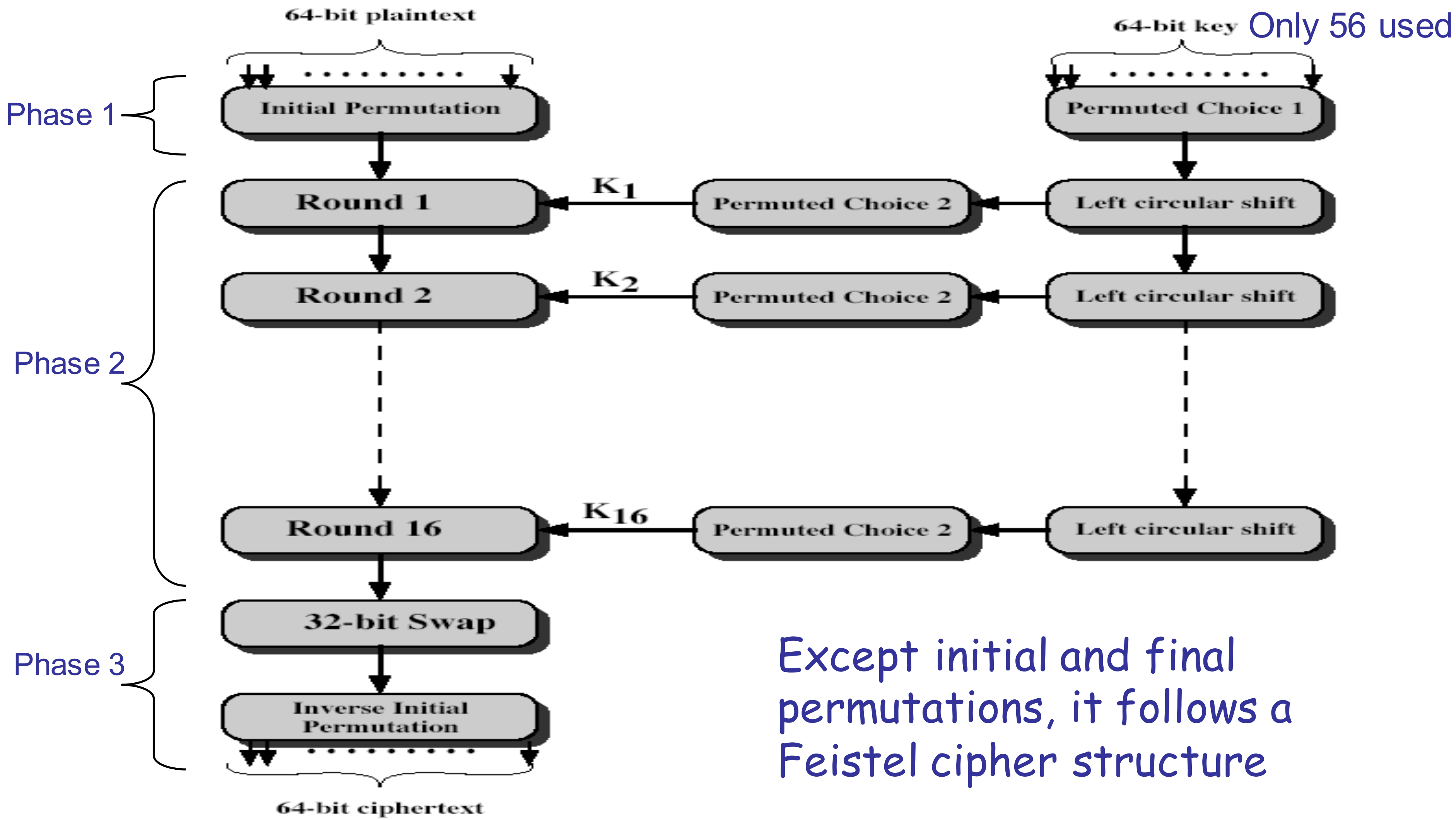
The basic process in enciphering :

- an initial permutation (IP)
- 16 rounds of key-dependent round function involving Ss and Ps
- a final permutation (IP^{-1})

Handling of the 56-bit key :

- initial key permutation which selects 56-bits
- 16 stages to generate subkeys using circular shift + permutation

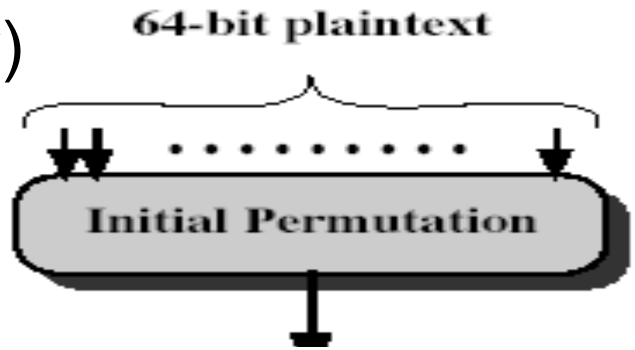
DES Encryption



Except initial and final permutations, it follows a Feistel cipher structure

Initial Permutation IP

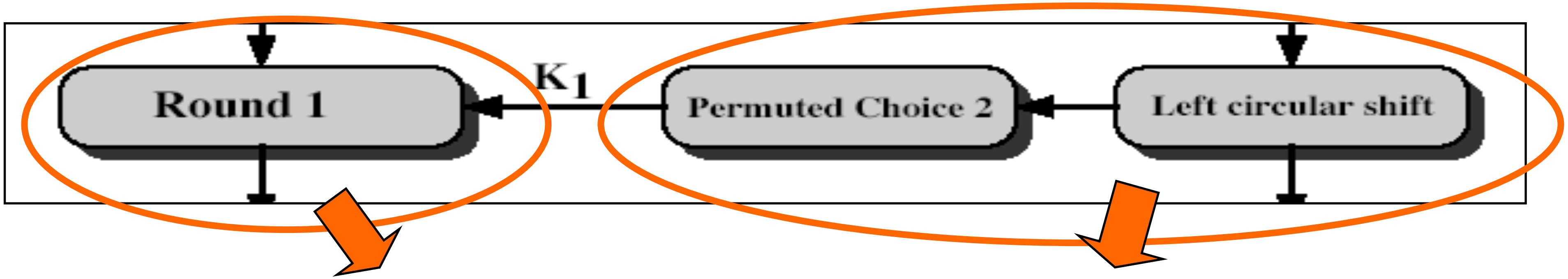
- first step reorders the input data bits
- even bits to left half LH, odd bits to RH
- quite regular in structure (easy in hw)
- $IP^{-1}(IP(M)) = M$



M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}
M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}
M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}
M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

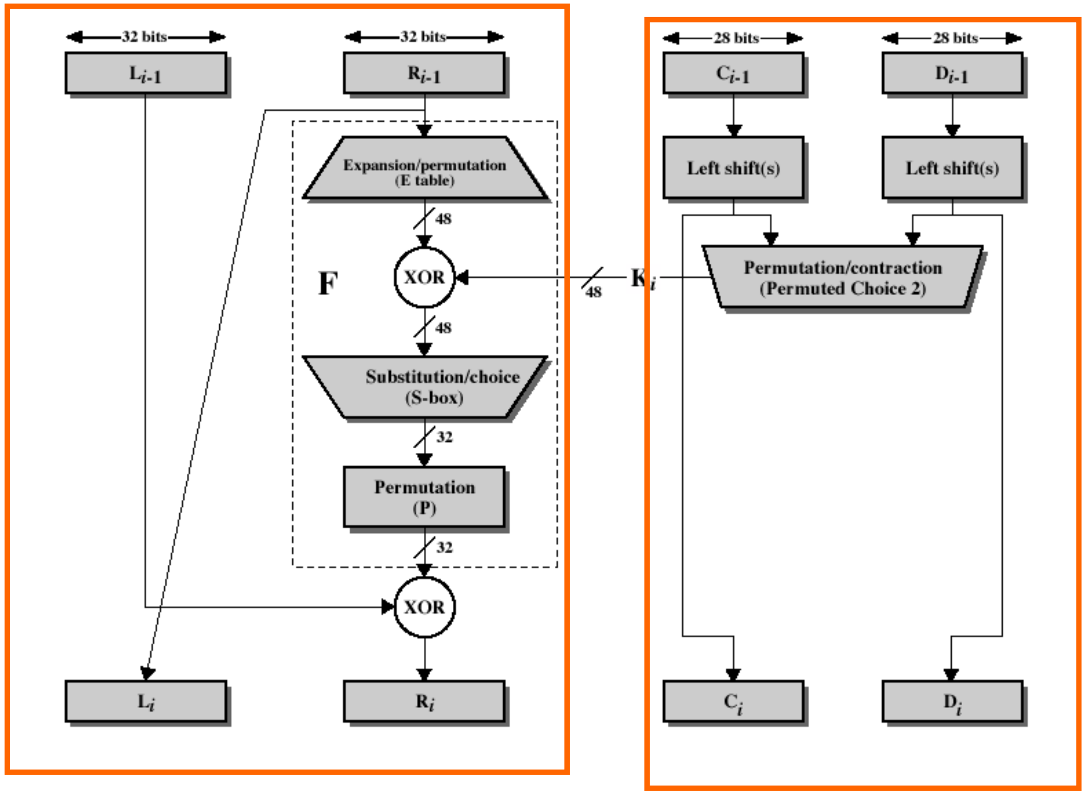
M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

DES Round Structure



The S-boxes provide "confusion"

The permutation P then spreads this as widely as possible so each S-box output affects as many S-box inputs in the next round as possible, thus providing "diffusion"

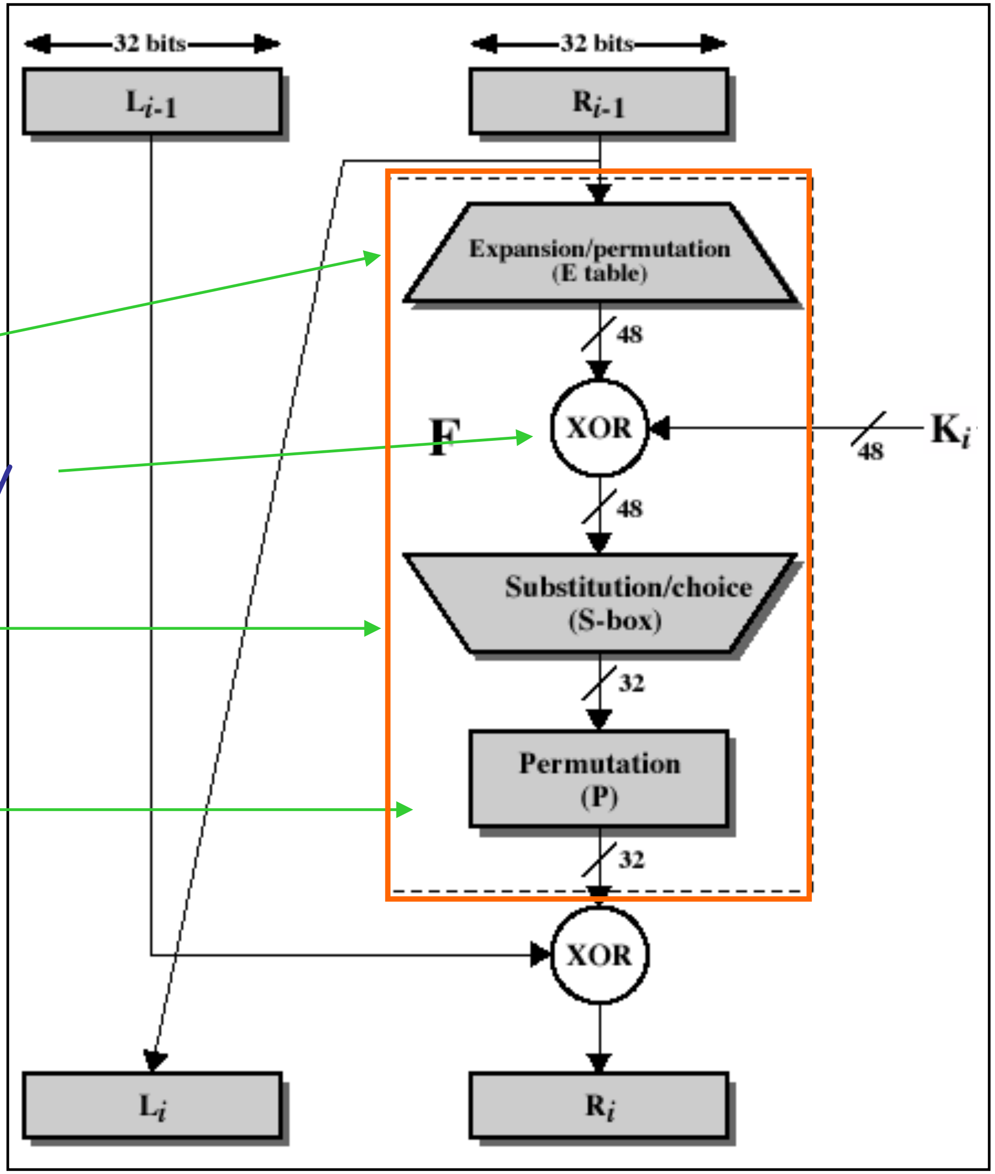


Expands R to 48-bits using perm E

xor with subkey

passes through 8 S-boxes to get 32-bit

permutes this using 32-bit perm P



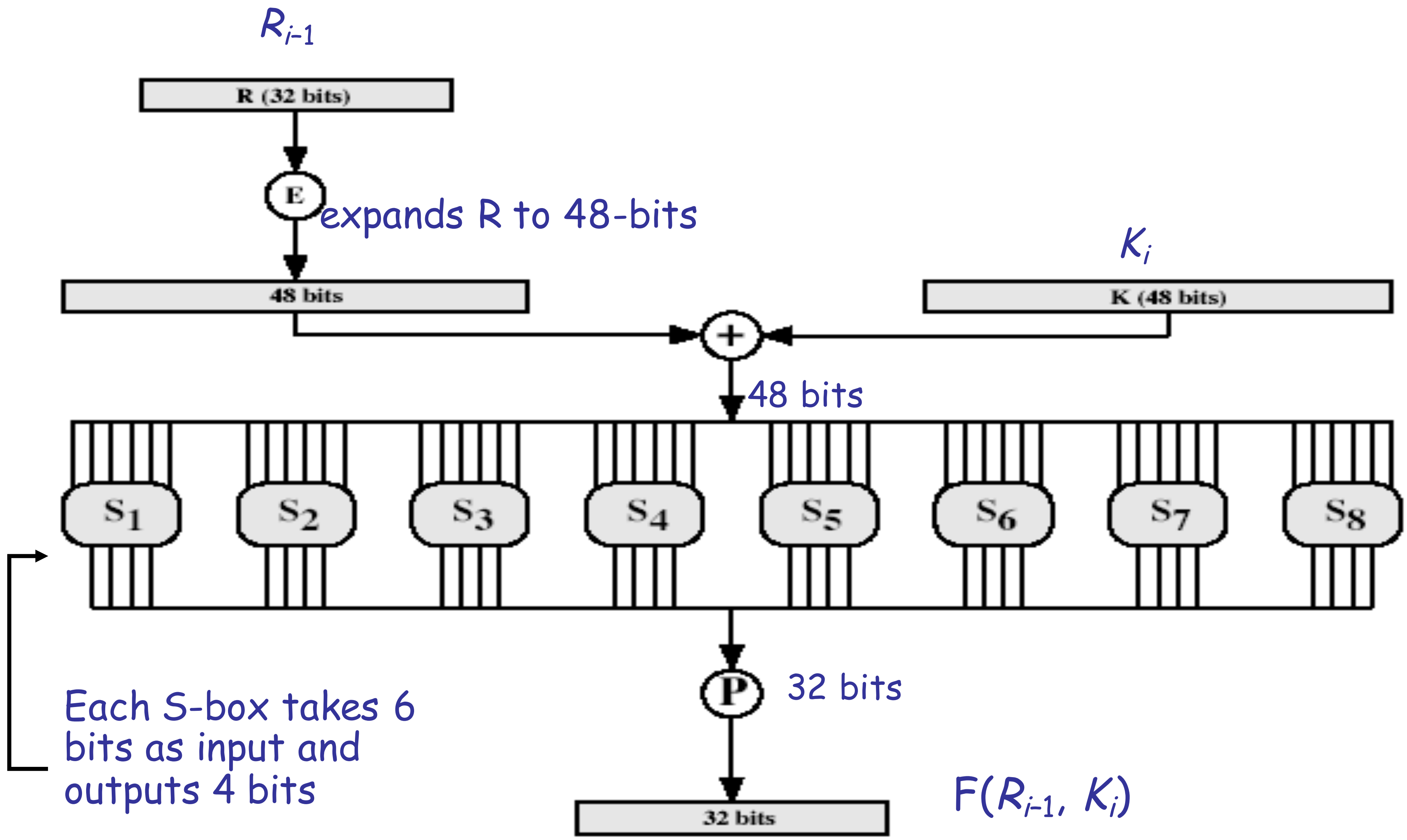
It uses two 32-bit L & R halves

As any Feistel cipher it can be described as:

$L_i = R_{i-1}$

$R_i = L_{i-1}$ xor $F(R_{i-1}, K_i)$

DES Function F



Substitution Boxes S

- Each S-box is defined by a 4x16 table

	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	[illegible]	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

  - outer input bits 1 & 6 (**row** bits) select one row
  - inner input bits 2-5 (**col** bits) select one column
  - the number in selected cell converted in binary gives the 4 output bits
    - E.g. input 110000 => row 2, column 8 =>
    - output 1111

Substitution Boxes S

- Final result is 8 sets of 4 bits, or 32 bits
- row selection depends on both data & key

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

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

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

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

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

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

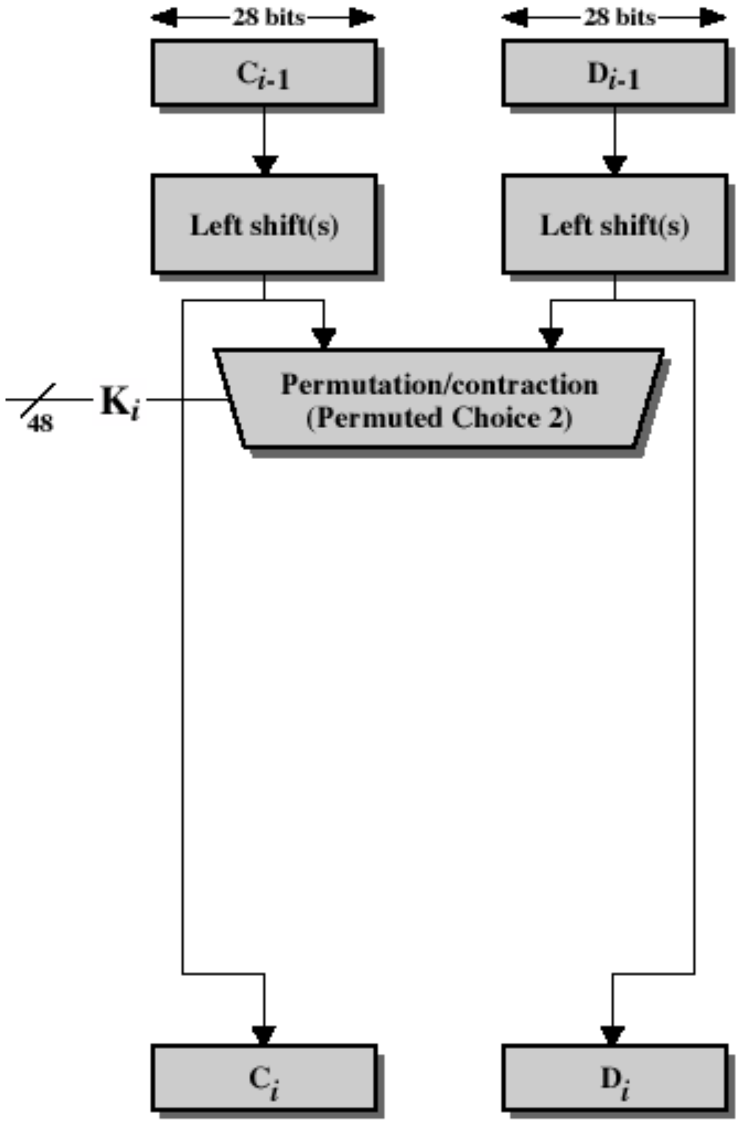
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

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

DES Key Generation

Subkeys used in each round:

- initial permutation of the key (PC1) which selects 56-bits divided into two 28-bit halves
- 16 stages consisting of :
  - rotating **each half** separately either of 1 or 2 places depending on the **key rotation schedule**
  - permuting them by PC2 with 48 bits as output for use in F



DES Decryption

- According to Feistel design, decryption is equivalent to encryption using subkeys in reverse order (SK_{16}, …, SK_1)
- 1st round with SK_{16} undoes 16th encrypt round
- ….
- 16th round with SK_1 undoes 1st encrypt round

DES and avalanche Effect

- It is a desirable property of encryption algorithms
- Changing **one** input or key bit results in changing approx **half** output bits
  - No more, no less than 0.5 !!!
- It makes difficult attempts to reduce the size of key space research
- DES exhibits a strong avalanche effect

Strength of DES – Key Size

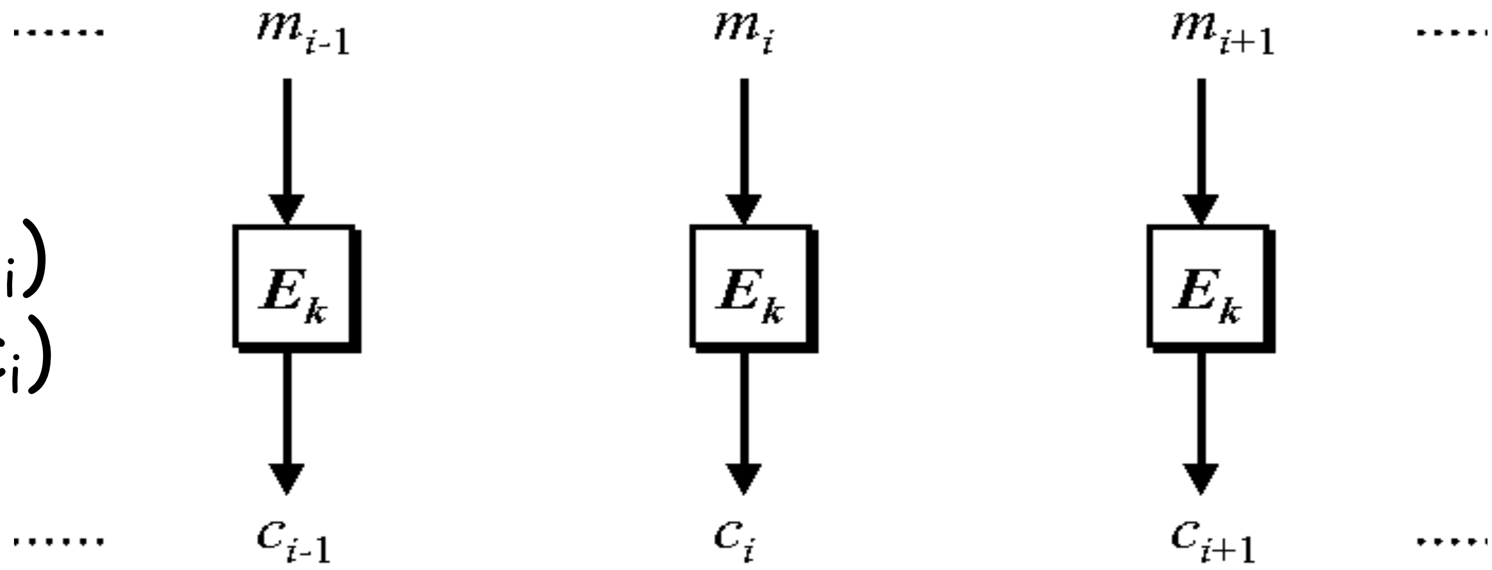
- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- Brute force search is hard !!!!
- With the availability of more powerful computers the time to break DES has decreased considerably
  - in 1998 on dedicated hw in a few days
  - in 1999 in 22hrs!
- Still must be able to recognize plaintext in automatic way
- Alternatives to DES: Triple DES, today AES

Modes of Operation

- A block cipher encrypts plaintext in fixed-size n-bit blocks (often n=64).
- For messages exceeding *n* bits, the simplest approach is to partition the message into blocks and encrypt each separately.
- Other approaches have been designed.
- We have currently 4 different methods of employing block ciphers (*Modes of Operation*)
  - m_i = plaintext block c_i = ciphertext block
  - E_k = Encryption D_k = Decryption

Electronic Code Book mode

$$c_i = E_k(m_i)$$
$$m_i = D_k(c_i)$$



- *Chaining dependencies*: each plaintext block is encrypted independently.
- *Identical plaintext blocks*: identical blocks in plaintext give identical blocks in ciphertext, so that block ciphers do not hide data patterns !!!!

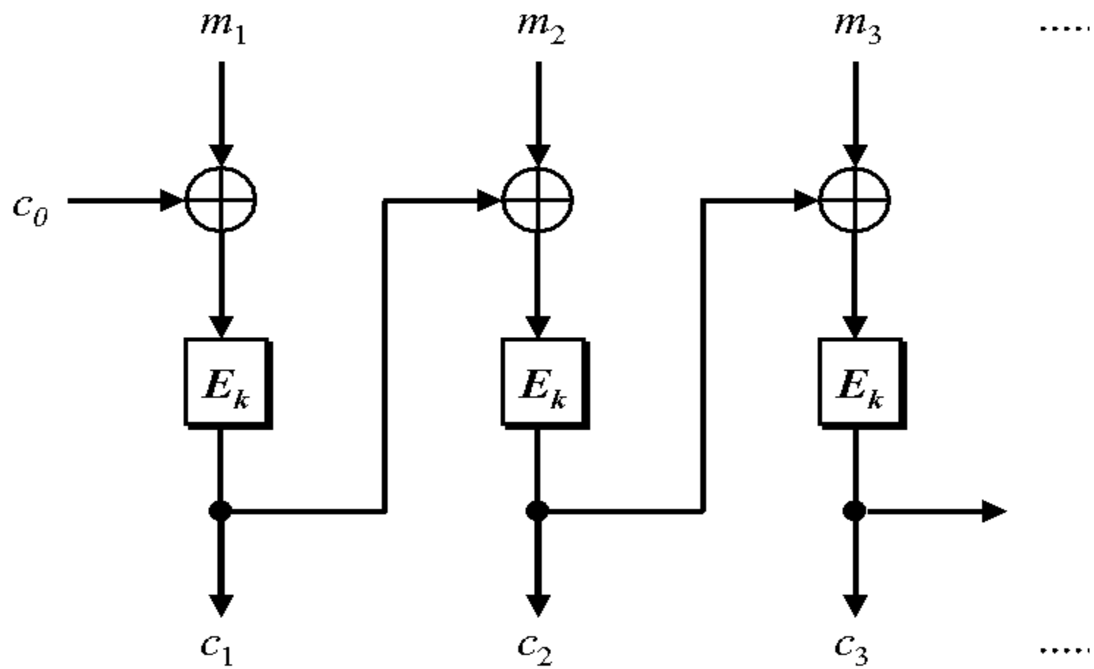
Electronic Code Book mode

- Not recommended for messages longer than a few blocks
- *Error propagation*: one or more bit errors in a single ciphertext block affect decryption of that block only.
- Since ciphertext blocks are independent, malicious substitution of ECB blocks does not affect the decryption of adjacent blocks.
- ECB allows easy parallelization => speed.

Cipher Block Chaining mode

A vector c_0 is used as a "seed" for the process:

$$c_i = E_k(m_i \oplus c_{i-1})$$
$$m_i = D_k(c_i) \oplus c_{i-1}$$



- *Identical plaintext blocks*: identical ciphertext blocks result only if same plaintext is enciphered under same key & seed.
- *Chaining dependencies*: ciphertext c_i depends on m_i and all preceding plaintext blocks: rearranging order of c_i blocks affects decryption.

Cipher Block Chaining mode

- c_0 should be different for any two messages encrypted with same key and is randomly chosen. It can be transmitted with the key or the ciphertext.
- Encryption process is difficult to parallelize. Decryption can be easily parallelized.
- *Error propagation*: a single bit error in ciphertext block c_i affects decryption of blocks c_i and c_{i+1} (since m_i depends on c_i and c_{i-1} being $m_i = D_k(c_i) \oplus c_{i-1}$).
- *Error recovery*: the CBC mode is *self-synchronizing*: if an error occurs in block c_i or if an entire block is lost, c_{i+2} is correctly decrypted

Cipher FeedBack mode

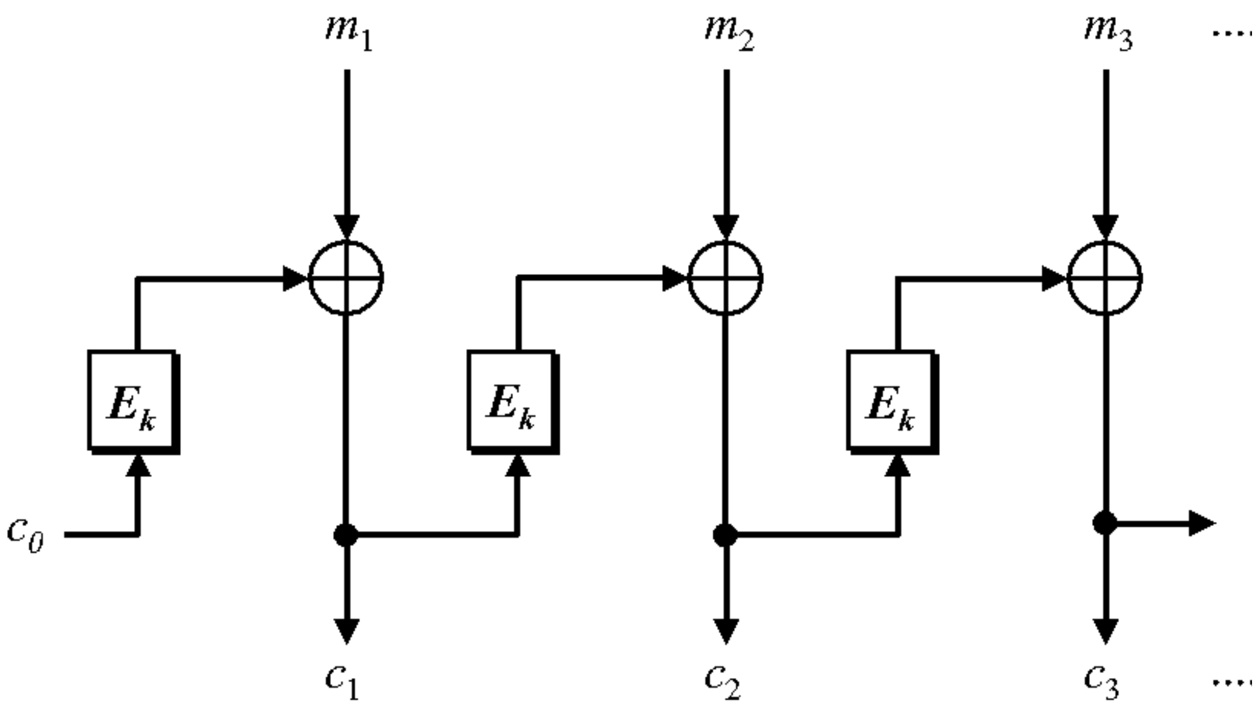
- CBC decryption re-synchronizes after transmission losses only if a whole block is lost
- Additionally, zero padding is needed if the size of the message length is not a multiple of 64 (for DES)
- CFB overcomes the above limitations

Cipher FeedBack mode

A vector c_0 is used as a "seed" for the process.

$c_i = m_i \oplus E_k(c_{i-1})$

$m_i = c_i \oplus D_k(c_{i-1})$



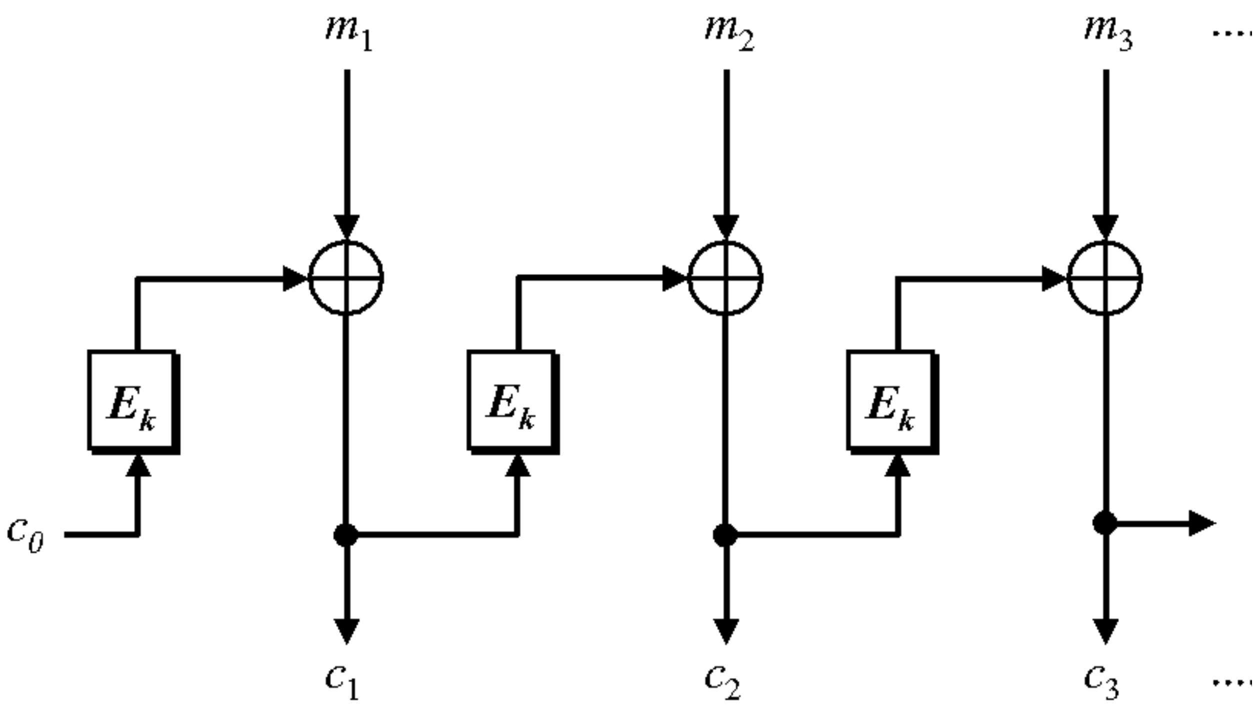
- As for CBC, the encryption process can not be parallelized, but the decryption process can.
- To improve self-synchronizing properties, CFB must be used in a slightly different way.

Cipher FeedBack mode

A vector c_0 is used as a "seed" for the process.

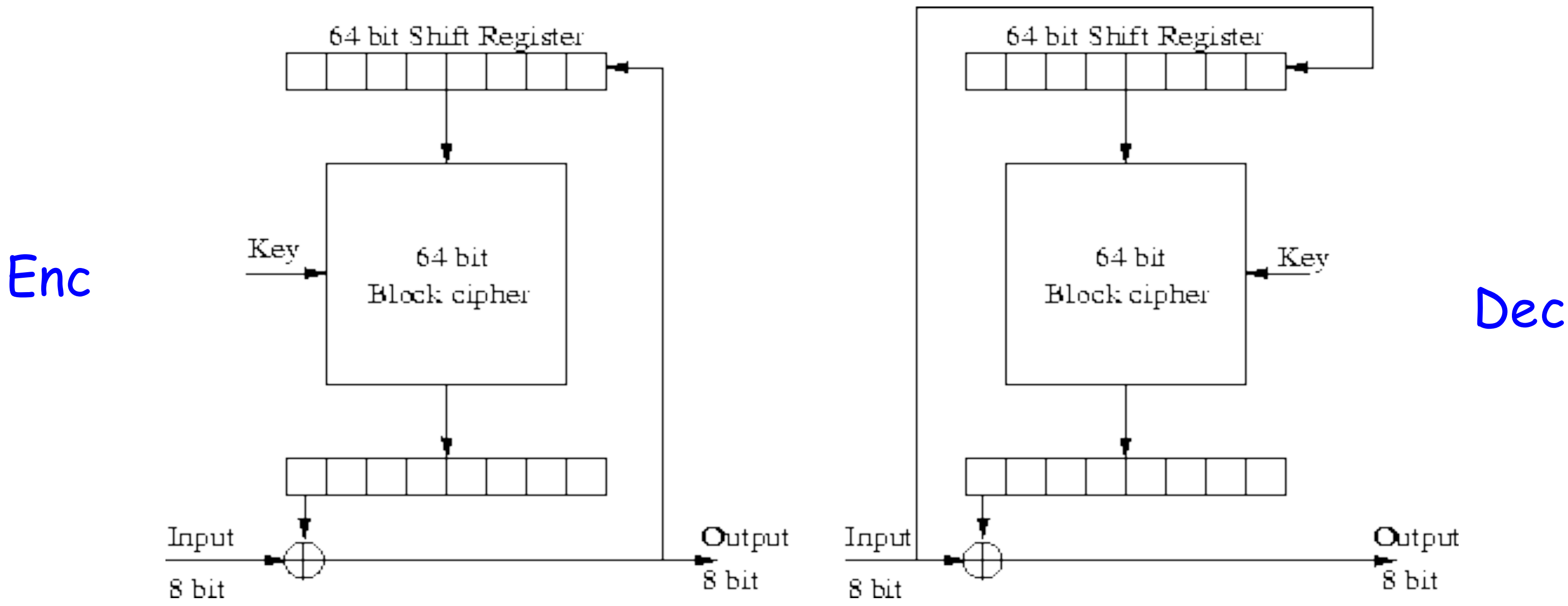
$c_i = m_i \oplus E_k(c_{i-1})$

$m_i = c_i \oplus D_k(c_{i-1})$



- As for CBC, the encryption process can not be parallelized, but the decryption process can.

Cipher FeedBack mode



- A shift register is filled with c_0, and encryption is run to produce 64 output bits.
- The left-most 8 bits of the output are XOR'ed with the byte to be transmitted.
- The result of the XOR is sent over the network and also fed back to the 64 bit shift register, shifting the left-most 8 bits out.
- Then, the encryption algorithm is run again and the next 8 bits are encrypted in the same manner.

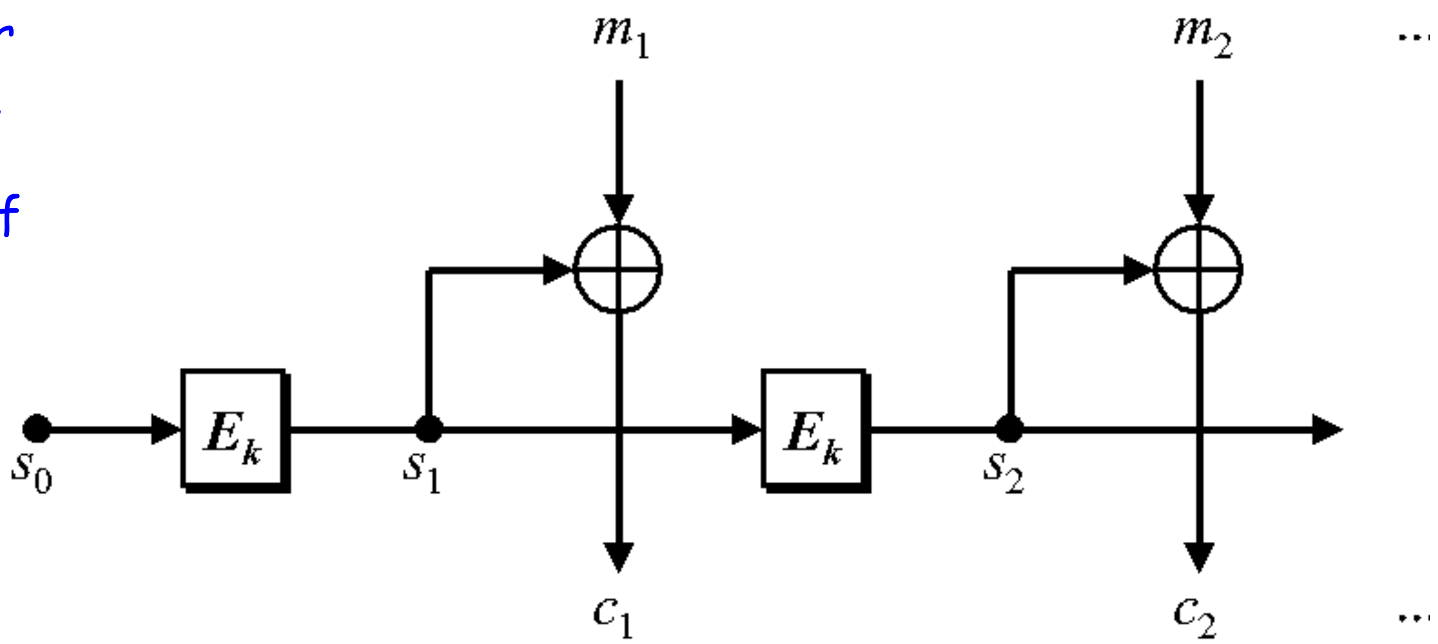
Output FeedBack mode

An initialization vector s_0 is used as a "seed" , and each s_i is derived from the encryption of the previous block s_{i-1}.

$$c_i = m_i \oplus s_i$$

$$m_i = c_i \oplus s_i$$

$$s_i = E_k(s_{i-1})$$



- OFB may be used for applications where error propagation must be avoided.
- It is similar to CFB, but differs in that the quantity XORed with each plaintext block is generated independently of plaintext & ciphertext. In this way parallelization can still be obtained.

Output FeedBack mode

- *Identical plaintexts*: as per CBC and CFB modes, changing the seed results in the same plaintext enciphered to a different output.
- *Chaining dependencies*: the keystream is plaintext-independent
- *Error propagation*: one or more bit errors in any c_i affects the decipherment of only that block.

Stream Ciphers

- Stream ciphers typically operate on smaller units of plaintext, usually bits.
- They are, in one sense, block ciphers having block length equal to one.
- If transmission errors are highly probable, stream ciphers are advantageous since they have no error propagation.
- They can also be used when the data must be processed one symbol at a time (e.g. in devices with no memory or limited data buffering).
- Can be designed to be much faster than any block cipher

Stream Ciphers

- There are relatively few fully-specified stream ciphers in the open literature.
  - Most practical stream ciphers are proprietary and confidential.
  - However some block ciphers have been published, some of which standardized or in public domain.
- Nevertheless, stream ciphers are widely used today, and one can expect increasingly more concrete proposals in the coming years.

Stream Ciphers

- In block ciphers, each plaintext element is encrypted with the same key e:

$$E_e(\mathbf{m}) = (e(m_1)e(m_2)\ldots e(m_t)) = (c_1c_2\ldots c_t) = c;$$

  - encryption of any plaintext with a block cipher will result in the same ciphertext when same key is used (we neglect the operating modes described previously, which sometimes make block ciphers similar to stream ciphers).
- In stream ciphers, the idea is to generate a *keystream*, sequence of key values, so that each plaintext element is encrypted with a different key.

Stream Ciphers

- In stream ciphers, the idea is to generate a *keystream*, sequence of key values, so that each plaintext element is encrypted with a different key.
- ***A*** alphabet of *q* symbols, E_e a simple substitution cipher with block length 1 where $e \in \boldsymbol{K}$.
- $m_1m_2m_3\ldots$ is the plaintext string, and $e_1e_2e_3 \ldots$ is a *keystream* from ***K.***
- A *stream cipher* takes the plaintext string and produces a ciphertext string $c_1c_2c_3 \ldots$ where $c_i = E_{ei}(m_i)$.
- If d_i denotes the inverse of e_i, then $D_{di}(c_i) = m_i$ decrypts the ciphertext string.

Stream Ciphers

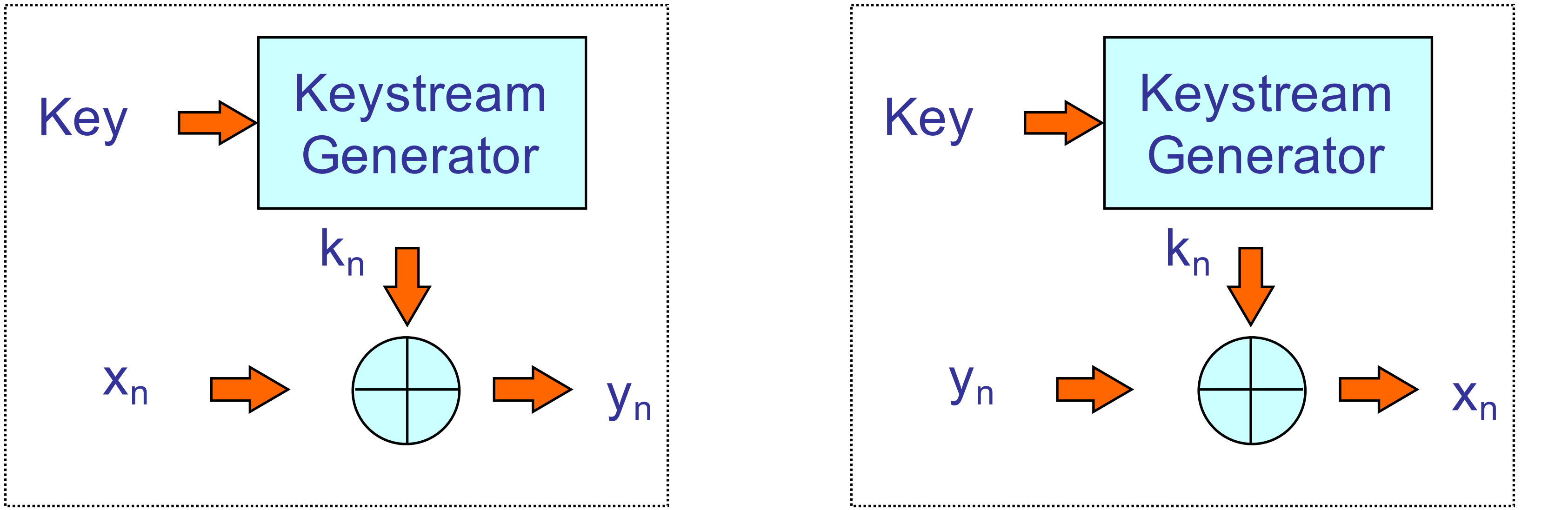
- A stream cipher generates what is called a *keystream* (a sequence of bits used as a key).
- The keystream could be generated at random, or by an algorithm, called *keystream generator*, which generates the keystream from an initial key (called a *seed*).

Stream Ciphers

- The generation of keystream can be:
  - independent of the plaintext and ciphertext (*synchronous* stream cipher);
  - it can depend on the data and its encryption (*self-synchronizing*).
- Most designs are for synchronous stream ciphers.

Additive Stream Ciphers

- Most of the stream ciphers proposed to date are additive stream ciphers.
  - Encryption is accomplished by combining keystream and plaintext, with the XOR.



Encryption

Decryption

Vernam Cipher

- The *Vernam Cipher* is a stream cipher defined on the alphabet ***A*** = {0,1}.
- A binary message $m_1m_2m_3\ldots m_t$ is operated on by a binary key string $k_1k_2k_3 \ldots k_t$ of the same length to produce a ciphertext string $c_1c_2c_3 \ldots c_t$ where

$$c_i = m_i \oplus k_i; \quad 1 \le i \le t$$

- Decryption is simply obtaind by

$$m_i = c_i \oplus k_i; \quad 1 \le i \le t$$

One-Time Pad

- If key string is randomly chosen and never used again, the Vernam cipher is called ***one-time pad***.
- It is unbreakable since ciphertext bears no statistical relationship to the plaintext
  - randomness of stream key completely destroys statistical properties in the message
  - **Unconditional security**
- For **any ciphertext** C and any plaintext M, a key K always exists mapping M into C, in addition all keys are equiprobable => it is not possible to discover the key which was really used.

One-Time Pad

Given the ciphertext:

- ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

suppose that with a brute force attack we find:

- key: px7mvmsydoftyrvzwc tniebnecvgdupahfzzlmnyih
- plaintext: mr mustard with the candlestick in the hall

and

- key : mfugpmiydyaxgoufhkll7mhsqdqogtewbqfgyovuhwt
- plaintext : miss scarlet with the knife in the library

How can we decide which is the correct plaintext?

One-Time Pad

- It is (provably) unconditionally secure !!!
- Problems:
  - We use only once the random key
  - problem of safe distribution of key (same length of the plaintext)
- This reduces the practicality of the system in all but a few specialized situations.
  - Until very recently the red line Moscow - Washington was secured by a one-time pad. Key transport by trusted courier.

Periodic Stream Ciphers

- A stream cipher is periodic with period *d* if $k_{i+d} = k_i$ for all i > 0
- The Vigenère cipher with keyword length *m* can be thought of as a periodic stream cipher with period *m*.
  - In this case, the key is $K=(k_1, .., k_m)$;
  - K itself provides the first m elements of the keystream, then the keystream just repeats itself from that point on.

Stream Cipher Properties

Some design considerations are:

- Generate keystreams with long period (same considerations of Vigenère)
- Statistically random (same number of 0’s and 1’s)
- Use long key (seed) to generate the keystream (against brute force attacks). Today at least 128 bits.

RC4 Cipher

- It is a proprietary cipher invented by Ron Rivest in 1987 for the company “RSA Security”.
- It was revealed in 1994 on the Internet by an anonymous.
- Simple but effective, very fast in hw and sw.
- The period of keystream is > 10^{100}
- Most widely used stream cipher (e.g. in web SSL/TLS, wireless WEP) .

RC4 Cipher

- A key with variable length 1÷256 bytes
- The key is used to initialize a byte array **S** of 256 elements with a random permutation of 8-bit values (0-255).
- At each iteration, **S** contains a different permutation.
- To encrypt:
  - Byte *k* of the keystream is generated selecting one of the elements of **S**
  - A XOR between plaintext byte and *k* is computed.

RC4: initialization

- **S**: **internal state** of the cipher, filled in with 0 ... 255
- **key is used only here**
- Given a key K of length *keylen* bytes (<=256)

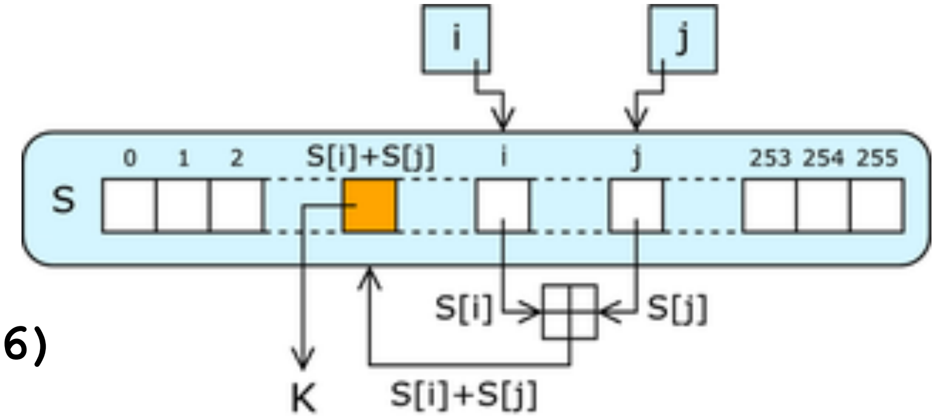
```
for i = 0 to 255 do
   S[i] = i
j = 0
for i = 0 to 255 do
   j = (j + S[i] + k[i mod keylen]) (mod 256)
   swap (S[i], S[j])
```

- The result is a well and truly shuffled array of numbers in [0, 255].
- Total number of possible states is 256!

RC4: PRGA

- Uses the permuted S to encipher input data one byte at a time
- Encryption continues shuffling array values
- Sum of shuffled pair selects "stream key" value

```
i = j = 0
for each message byte Mi
  i = (i + 1) (mod 256)
  j = (j + S[i]) (mod 256)
  swap(S[i], S[j])
  t = (S[i] + S[j]) (mod 256)
  K = S[t]
```



RC4 Encryption

- The "keystream" value is XORed with next byte of message to en/decrypt

```
i = j = 0
for each message byte Mi
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    K = S[t]
    Ci = Mi XOR K
```

RC4 Example

- Simple 4-byte example
- S = {0, 1, 2, 3}
- K = {1, 7, 1, 7}
- Set i = j = 0

RC4: initialization

First Iteration (i = 0, j = 0, S = {0, 1, 2, 3}):

j = (j + S[i] + K[i]) = (0 + 0 + 1) = 1

Swap S[i] with S[j]: S = {1, 0, 2, 3}

Second Iteration (i = 1, j = 1, S = {1, 0, 2, 3}):

j = (j + S[i] + K[i]) = (1 + 0 + 7) = 0 (mod 4)

Swap S[i] with S[j]: S = {0, 1, 2, 3}

RC4: initialization

Third Iteration (i = 2, j = 0, S = {0, 1, 2, 3}):

j = (j + S[i] + K[i]) = (0 + 2 + 1) = 3

Swap S[i] with S[j]: S = {0, 1, 3, 2}

Fourth Iteration (i = 3, j = 3, S = {0, 1, 3, 2}):

j = (j + S[i] + K[i]) = (3 + 2 + 7) = 0 (mod 4)

Swap S[i] with S[j]: S = {2, 1, 3, 0}

PRGA and encryption

Reset i = j = 0, Recall S = {2, 1, 3, 0}

i = i + 1 = 1

j = j + S[i] = 0 + 1 = 1

Swap S[i] and S[j]: S = {2, 1, 3, 0}

Output t = [S[i] + S[j] = 1+1 = 2

Output K = S[2] = 3

$$C_i = M_i \text{ XOR } K$$

RC4 Security

To: **cypherpunks@toad.com**

Subject: **Thank you Bob Anderson**

From: **nobody@jpunix.com**

Date: **Fri, 9 Sep 1994 22:11:49 -0500**

SUBJECT: RC4 Source Code I've tested this. It is compatible with the RC4 object module that comes in the various RSA toolkits.

```
/* rc4.h */ typedef struct rc4_key { unsigned char state[256]; unsigned char x; unsigned char y; } rc4_key; void
    prepare_key(unsigned char *key_data_ptr,int key_data_len, rc4_key *key); void rc4(unsigned char
    *buffer_ptr,int buffer_len,rc4_key * key);
/*rc4.c */ #include "rc4.h" static void swap_byte(unsigned char *a, unsigned char *b); void
    prepare_key(unsigned char *key_data_ptr, int key_data_len, rc4_key *key) { unsigned char swapByte;
    unsigned char index1; unsigned char index2; unsigned char* state; short counter; state = &key->state[0];
    for(counter = 0; counter < 256; counter++) state[counter] = counter; key->x = 0; key->y = 0; index1 = 0;
    index2 = 0; for(counter = 0; counter < 256; counter++) { index2 = (key_data_ptr[index1] + state[counter] +
    index2) % 256; swap_byte(&state[counter], &state[index2]); index1 = (index1 + 1) % key_data_len; } }
    void rc4(unsigned char *buffer_ptr, int buffer_len, rc4_key *key) { unsigned char x; unsigned char y;
    unsigned char* state; unsigned char xorIndex; short counter; x = key->x; y = key->y; state = &key->state[0];
    for(counter = 0; counter < buffer_len; counter ++) { x = (x + 1) % 256; y = (state[x] + y) % 256;
    swap_byte(&state[x], &state[y]); xorIndex = state[x] + (state[y]) % 256; buffer_ptr[counter] ^=
    state[xorIndex]; } key->x = x; key->y = y; } static void swap_byte(unsigned char *a, unsigned char *b) {
    unsigned char swapByte; swapByte = *a; *a = *b; *b = swapByte; }
```

Key Distribution in symmetric schemes

- Symmetric schemes require both parties to share a common secret key
- An important and difficult issue is how to securely distribute this key
- Often secure system failure is due to a break in the key distribution scheme
- Remember that for a system with n users the number of keys grows as $O(n^2)$

Key Distribution

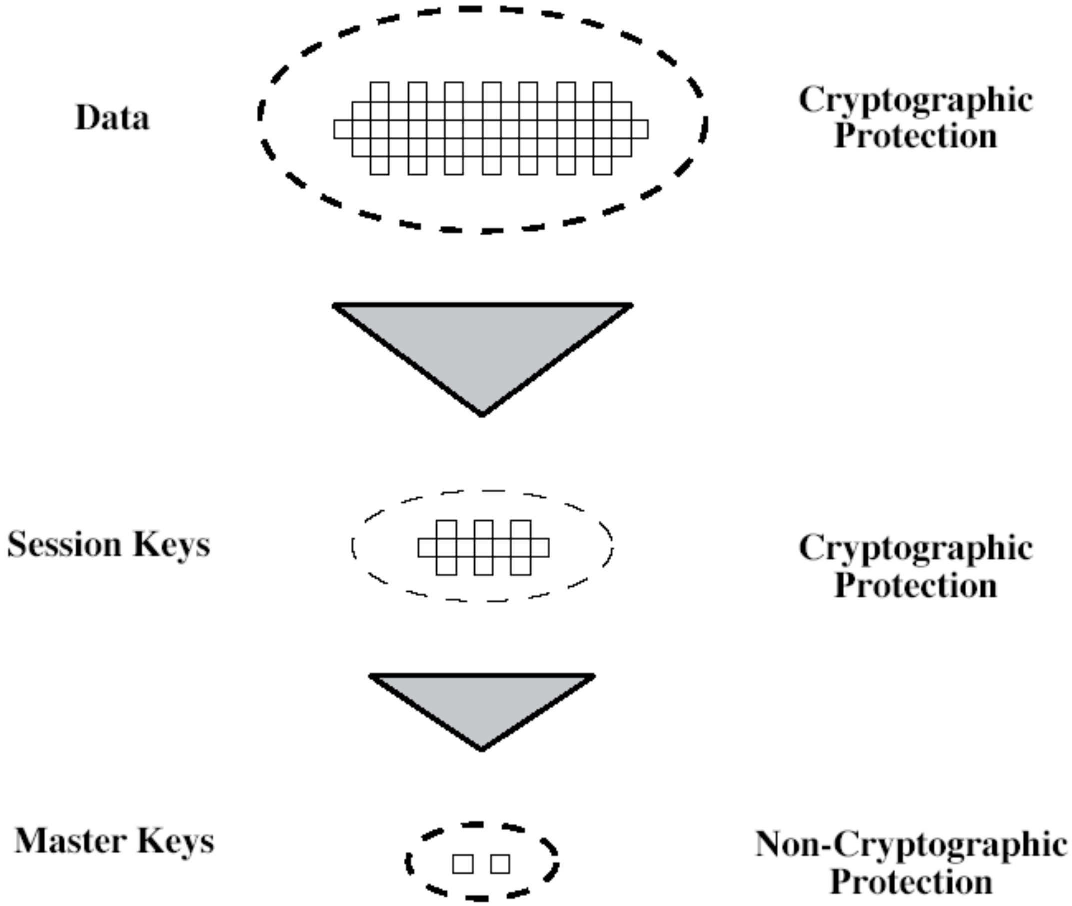
- Users A & B have various **key distribution** alternatives:
- **Manual distribution (only for very few users)**
  - A can select key and physically deliver it to B
  - Third party can select & deliver key to A & B
- **Based on secured channel**
  - If A & B have communicated previously they can use previous key to encrypt a new key
  - If A & B can communicate securely with a third party C, C can distribute keys to A & B
  - *Solution based on public-key cryptography*

KDC – based Key Distribution

- For point-to-point connections variations of the last option are used.
- A Key Distribution Center (KDC) shares (e.g. by means of manual distribution) with each user a master key used to grant the security of the connections with the user.
- For n users, only n master keys are sent manually.
  - **$n << n(n-1)/2$!!**

KDC – based Key Distribution

- Transmitted data are protected through a temporary session key, generated for each connection
- KDC sends the session key to the host by protecting it with the master key shared with the user



Key Distribution Issues

- Hierarchies of KDC's required for large networks, but mutual trust becomes a problem
- Session key lifetimes should be limited for greater security

References

- W. Stallings, “Crittografia e Sicurezza delle Reti”, Mc Graw Hill
  - Chapter 2, chapter 3 (sec. 3.2, 3.3, 3.4, 3.7), chapter 6 (sec. 6.5), chapter 7 (sec. 7.3)
- RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1
  - Chapter 2
- A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press
  - Chapter 6,7