Computer security

Symmetric encryption

Olivier Markowitch

Definition

$a, b, m \in \mathbb{Z}$ and $m > 0$:

$$a \equiv b \pmod{m} \text{ iff } m \mid b - a$$

Definitions

- $f : X \to Y$ is a one-way function if $y = f(x)$ is easy to compute for all $x \in X$, but for a y randomly chosen in Y, it is computationally difficult to find $x \in X$ such that $f(x) = y$

- $f : X \to Y$ is a one-way trapdoor function if f is a one-way function for which inverses can easily be computed with an additional information, called trapdoor

Symmetric ciphers

- a cipher is used to encrypt plaintexts, the results are called ciphertexts

- ciphertexts are decrypted to recover the plaintexts

- keys are used to encrypt and to decrypt

Symmetric ciphers

- The keys must be secretly shared by the communicating entities (symmetric ciphers use shared secret keys)

- (P, C, K, E, D) is cryptosystem where E is the encryption algorithm, D is the decryption algorithm, P, C and K are respectively the space of the plaintexts, the space of the ciphertexts and the space of the keys, and where $\exists k, k' \in K$ such that, $\forall x \in M : D_{k'}(E_k(x)) = x$

- in symmetric cryptosystems, usually $k = k'$ (or at least, knowing k it is easy to compute k')

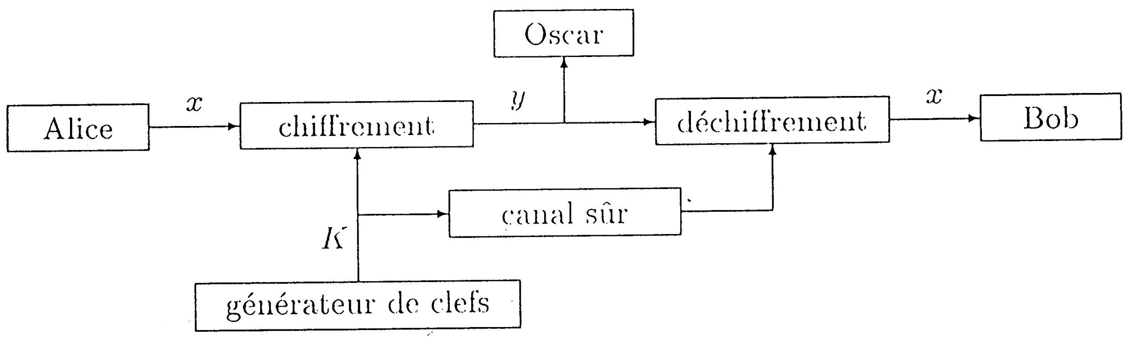


Figure 1.1 : Le canal de communication

Cryptanalysis

Kerckhoffs (1883): encryption and decryption algorithms are known

Auguste Kerkhoffs

Cryptanalysis

- *known ciphertext attack*
- *known plaintext attack*
- *choosen plaintext attack*
- *choosen ciphertext attack*

Historical encryption schemes

- substitutions
- transpositions

Shift encryption scheme

$M = C = K = \mathbb{Z}_{26}$, $0 \leq k \leq 25$ and $x, y \in \mathbb{Z}_{26}$

$$E_k(x) = x + k \mod 26$$

$$D_k(y) = y - k \mod 26$$

Example: with $k = 3$, the plaintext CAESAR is ciphered in FDHVDU

Cryptanalysis?

Mono-alphabetic substitution

$M = C = \mathbb{Z}_{26}$, K is the set of permutations on $\{0, \ldots, 25\}$

For each permutation $k \in K$ we have:

$$E_k(x) = k(x)$$

$$D_{k'}(y) = k^{-1}(y)$$

where $x, y \in \mathbb{Z}_{26}$ and k^{-1} being the inverse permutation of k

Cryptanalysis? (brute force: $26! > 4 \cdot 10^{26}$ possible keys)

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T

n	o	p	q	r	s	t	u	v	w	x	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

Chiffrement

A	B	C	D	E	F	G	H	I	J	K	L	M
d	l	r	y	v	o	h	e	z	x	w	p	t

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	g	f	j	q	n	m	u	s	k	a	c	i

Déchiffrement

Mono-alphabetic substitution

plaintext: “chiffrementparpermutation”

ciphertext: “YGZPPCHTHSMLXCLHCTUMXMZFS”

Attack: letters frequencies in the ciphertexts are the same than in the plaintexts $\rightarrow$ use of frequencies tables based on the language of the plaintext (letter, digrams, trigrams, ...)

lettre	probabilité	lettre	probabilité
A	0,082	N	0,067
B	0,015	O	0,075
C	0,028	P	0,019
D	0,043	Q	0,001
E	0,127	R	0,060
F	0,022	S	0,063
G	0,020	T	0,091
H	0,061	U	0,028
I	0,070	V	0,010
J	0,002	W	0,023
K	0,008	X	0,001
L	0,040	Y	0,020
M	0,024	Z	0,001

Table 1-1 : Probabilité d'occurrence des lettres de l'alphabet

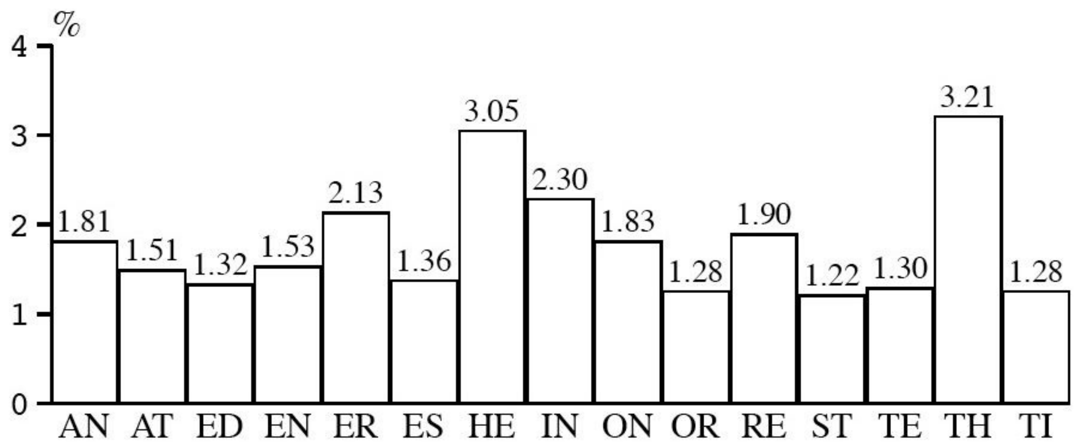


Figure 7.6: *Frequency of 15 common digrams in English text.*

Poly-alphabetic substitution

Encryption of blocs composed of t symbols

- E consists in all the sets of t permutations of the symbols

- each key $k \in K$ define a set of t permutations $(p_1, \ldots, p_t)$

- the plaintext $x = x_1 \ldots x_t$ is encrypted on the basis of the key k:

$$E_k(x) = p_1(x_1) \ldots p_t(x_t)$$

- the decryption key k' define the set of the t corresponding inverse permutations: $(p_1^{-1}, \ldots p_t^{-1})$

Vigenere cipher

Blaise de Vigenere 16^{th} century

Blaise de Vigenere

Encryption scheme described by Blaise de Vigenere but probably first designed by Giovan Battista Bellaso

Vigenere cipher

$m > 0$ and $M = C = K = (\mathbb{Z}_{26})^m$

For a key $k = (k_1, \ldots, k_m)$:

$$E_k(x) = E_k(x_1, \ldots, x_m) = (x_1 + k_1, \ldots, x_m + k_m)$$

$$D_k(y) = D_k(y_1, \ldots, y_m) = (y_1 - k_1, \ldots, y_m - k_m)$$

with $x_i, y_i \in \mathbb{Z}_{26}$ and all the operations are computed in $\mathbb{Z}_{26}$

Vigenere cipher

Example:

key: “hello” 7 4 11 11 14”

plaintext: “rendezvousahuitheure”

```
17 04 13 03 04 25 21 14 20 18
07 04 11 11 14 07 04 11 11 14
-- -- -- -- -- -- -- -- -- --
24 08 24 14 18 06 25 25 05 06

00 07 20 08 19 07 04 20 17 04
07 04 11 11 14 07 04 11 11 14
-- -- -- -- -- -- -- -- -- --
07 11 05 19 07 14 08 05 02 18
```

ciphertext: “YIYOSGZZFGHLFTHOIFCS”

Vigenere cipher

In practice the key can be a string of m characters (converted in numbers within the ciphering process)

There are 26^m possible keys

Cryptanalysis: how to determine the key length? (Kasisky)

Vigenere cipher

Exercise: try to cryptanalyse the following ciphertext

iwiegiqjaexgotcijxaehogsfpxzifrqkskghxtrmqkmvrwvaje
xyszhllfzglrehyrtsvplsaxmqkrrmw gugvhsivvfuughrkicg
hycrvditvvhycfqpkcyefangsxxrrmwreuiyonvvigczphsee
xiurdiqzuegkofw vhiejxdjiiitaicwsxejiqzeexxtsvrvsazwg
gpiiviehyhtolwvgvfrvjebmgjjvrhjemelyprwoksltsusvvfgp
rfojdvjdhrzuxkrlrhihrrwolcsqjetvbvtfkugpymhhivrdhskvx
yeaimagveljoegwuukhdhoihtaet ioaitmhzazxganvivveti
vomgphzecghveehdtthydriexhrlzkhtcvkuusjmhxeuypgr
zrlrdlxsgrrmwxerfvullqhttzrvullfoksrrvratphl

Transposition ciphers

$m > 0$, $M = C = \{0, \ldots, 25\}^m$

K is the set of the permutations $\{1, \ldots, m\}$

A key $k \in K$ is a permutation and:

$$E_k(x) = E_k(x_1, \ldots, x_m) = (x_{k(1)}, \ldots, x_{k(m)})$$

$$D_{k'}(y) = D_{k'}(y_1, \ldots, y_m) = (y_{k^{-1}(1)}, \ldots, y_{k^{-1}(m)})$$

where $x_i, y_i \in \mathbb{Z}_{26}$ and k^{-1} is the corresponding inverse permutation

Transposition ciphers

suppose the following permutation:

$1 \to 3, 2 \to 5, 3 \to 1, 4 \to 6, 5 \to 4, 6 \to 2$

and the corresponding inverse permutation:

$1 \to 3, 2 \to 6, 3 \to 1, 4 \to 5, 5 \to 2, 6 \to 4$

plaintext: "annulerlelancement" = "annule rlelan cement"

ciphertext: "NEALNU ENRALL MTCNEE"

Cryptanalysis: plaintext symbols occurences are preserved in the ciphertext.

Scytale

Strip of leather around a stick (around 400 BC)

The plaintext is written by writing a letter on each convolution

To decrypt, we must use a stick of the same diameter as the original stick

Enigma

Electromecanical device (with a keyboard, a lightboard, three different rotors and a reflector)

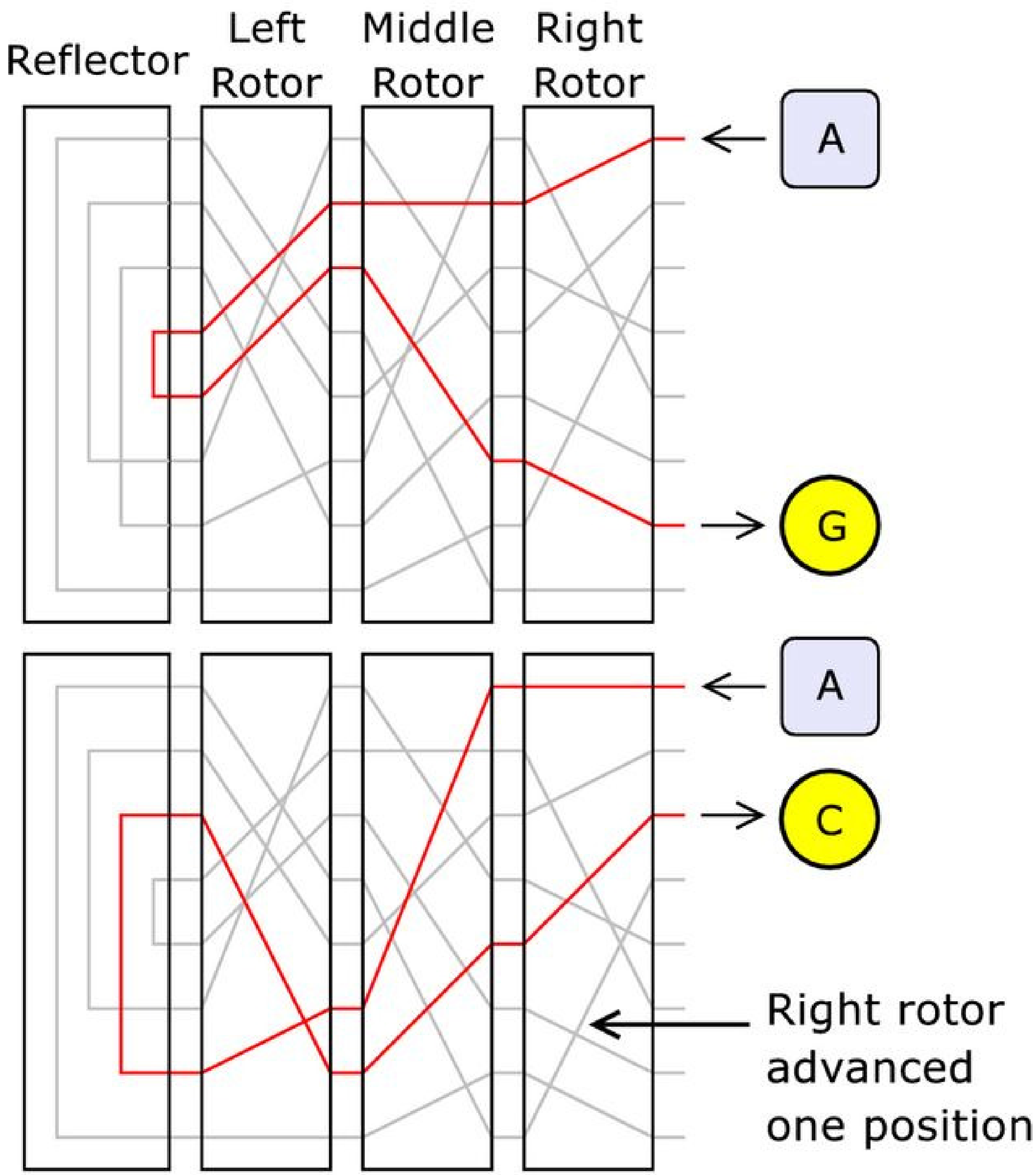
To encrypt: press the plaintext letter on the keyboard and the corresponding ciphertext lights up

To decrypt: press the ciphertext letter on the keyboard and the corresponding clairtext lights up

The rotors turn after that a letter is pressed on the keyboard

The sender and the recipient have to configure in the same way their respective device (choice, positions and order of the rotors, configuration of the reflector, ...) $\rightarrow$ this is the secret key


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


Reflector
Left Rotor
Middle Rotor
Right Rotor
A
G
A
C
Right rotor advanced one position

Alan Turing

TURING AT BLETCHLEY PARK

Alan Turing joined Bletchley Park in September 1939, aged 27. Bletchley Park was home to the Government Code and Cypher School, responsible for studying and breaking secret communications.

Turing was one of several high-flying Cambridge and Oxford academics identified for recruitment to Bletchley Park. Not all were mathematicians like Turing. One of the candidates was the fantasy writer and linguist J R R Tolkien, though he did not take up the offer.

Turing became head of the naval Enigma team at Bletchley Park. Building on the work of Polish specialists he designed systems and machines to break Enigma messages.

CODEBREAKING ON AN INDUSTRIAL SCALE

Every day, the settings used by German Enigma operators changed, meaning Bletchley Park needed to test millions of settings to find the new ones. This was codebreaking on an industrial scale – and it needed industrial-sized machines to do it.

These huge devices were invented by Alan Turing and his colleague Gordon Welchman, and named 'bombes' after a Polish predecessor. Dozens of wheels in each bombe rotated continuously, electrically testing up to a million different settings until a potential Enigma match had been found.

The bombes made a noise 'like a thousand knitting needles' according to one writer, and over 200 were manufactured by a nearby punched-card equipment firm. Some were installed at Bletchley Park but most operated elsewhere. All 200 operated around the clock.

The secret bombe manufacturing facility at the British Tabulating Machines factory, Letchworth, and one of the bombe machines in Bletchley Park's bombe hut.

Images: GCHQ

Wheels from a bombe checking machine, 1940s

The bombes were manufactured on a secret production line at the British Tabulating Machines factory in Letchworth, near Bletchley. Each one contained dozens of wheels similar to these.

Different wiring configurations were tested by the bombes at high speed until a possible match to that day's Enigma settings had been found. The device would then stop, and the settings were tested on a smaller 'checking machine'.

After the war ended the bombes were broken up and destroyed. However, these wheels from a checking machine survived, leaving us a tangible reminder of Turing's wartime achievements.

Source: GCHQ
Inv. No: E2012.41.1

One-time pad

One-time pad also known as Vernam scheme (1917)

Gilbert Vernam　　Joseph Mauborgne

Perhaps already initially designed around 1880 by Frank Miller

One-time pad

The plaintext $x = x_1 \dots x_t$ where each $x_i \in 0, 1$

The key $k = k_1 \dots k_t$ where each $k_i \in 0, 1$ is randomly chosen

$$E_k(x) = y = x_i \oplus k_i, 1 \leq i \leq t$$

$$D_k(m) = x = y_i \oplus k_i, 1 \leq i \leq t$$

The one-time pad is proven to be secure iff each key is used once

Stream ciphers

One-time pad generalization to be more practical

Short key expanded into a keystream

Keystream used in the same way than the one-time pad (xor)

Stream ciphers

Examples of stream ciphers: SEAL and RC4

Phillip Rogaway et Don Coppersmith *A Software-Optimized Encryption Algorithm*, Fast Software Encryption, Lecture Notes in Computer Science 809, Springer-Verlag, 1994

Phillip Rogaway et Don Coppersmith *A Software-Optimized Encryption Algorithm*, Journal of Cryptology, volume 11, number 4, Springer International 1998

Ron Rivest *The RC4 Encryption Algorithm*, RSA Data Security, 1992

Product ciphers and bloc ciphers

A *product cipher* combines (at least) two transformations (substitution, transposition) in order to produce a cipher that is more secure than the individual transformations

A *bloc cipher* is an encryption scheme that cuts the plaintext in fixed size blocs and encrypts each bloc separately

Bloc ciphers are usually *iterative*: a set of transformations is repeated. Each iteration is called a *round*. A different sub-key is derivated from the key for each round.

Feistel cipher

Horst Feistel

Feistel cipher

Iterative process (1970)

Plaintext ($2t$ bits with L_0 = t at the left and R_0 = t bits at the rigth)

r rounds where k_i is the corresponding sub-key:

$$L_i = R_{i-1}$$

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

Same process to decrypt (using the sub-keys in the opposite order)

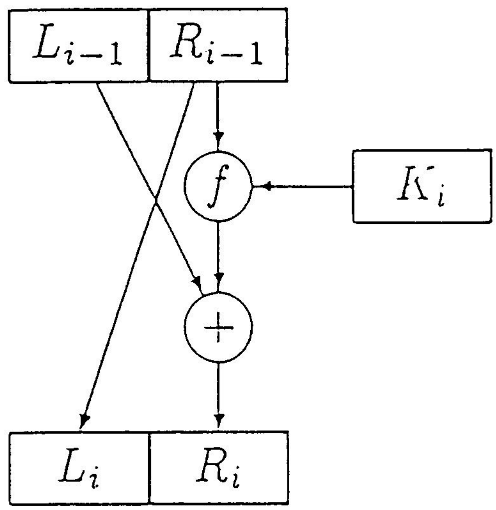


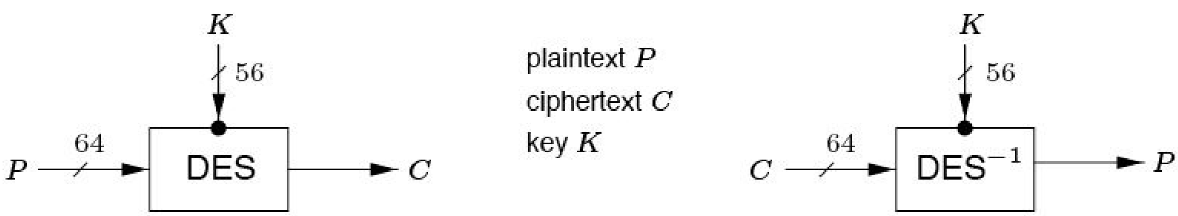
Figure 3.1 : Un tour de DES

Data Encryption Standard

Horst Feistel

Don Coppersmith

Data Encryption Standard



DES input-output.

Lucifer in 1973 - DES is a standard in 1976

Data Encryption Standard

DES encrypts a 64-bits plaintext into a 64-bits ciphertext using a 56-bits key

1. an initial permutation (IP) is realized on the plaintext (x): $x_0 = IP(x) = L_0 R_0$ (L_0 is the 32-bits at the left and R_0 is the 32-bits at the right)

2. 16 iterations of the function f: $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$ where $k_1, \ldots k_{16}$ are the 16 sub-keys (48-bits)

3. final permutation (inverse of the initial permutation): $IP^{-1}((R_{16} L_{16})$. The result of this last permutation is the 64-bits ciphertext

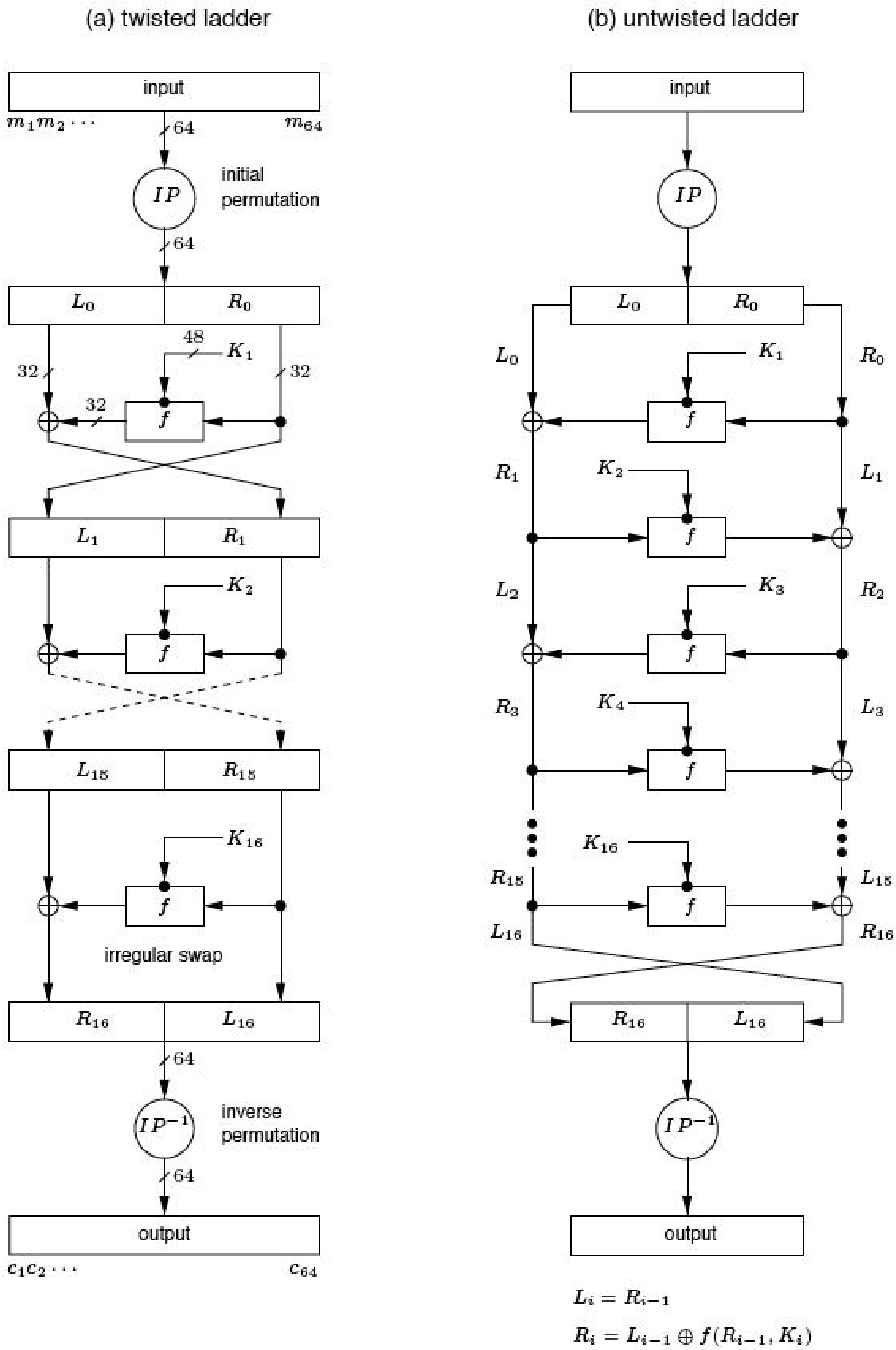


Figure 7.9: *DES computation path.*

Data Encryption Standard

The function f takes a 32-bits value (A) as input as well as a 48-bits sub-key (J), and produces a 32-bits result

1. A is *expanded* from 32 to 48 bits (thanks to an expansion function (E) that realizes a permutation and duplicates some bits)

2. $B = E(A) \oplus J$

3. B is cut in 8 blocs B_i of 6 bits

DES: f et SBoxes (suite)

4. the 8 B_i are modified by a different “sbox” (substitution box): $C_i = S_i(B_i), 1 \leq i \leq 8$ where C_i is 4 bits. Each sbox is a different 4×16 table of hexadecimal numbers. The bits $b_1 b_6$ of B_i composes the row index in the table, the bits $b_2 b_3 b_4 b_5$ composes the column index in the table. At this coordinate is the 4-bits result of the sbox

5. The 8 C_i composes C that goes trough a permutation P

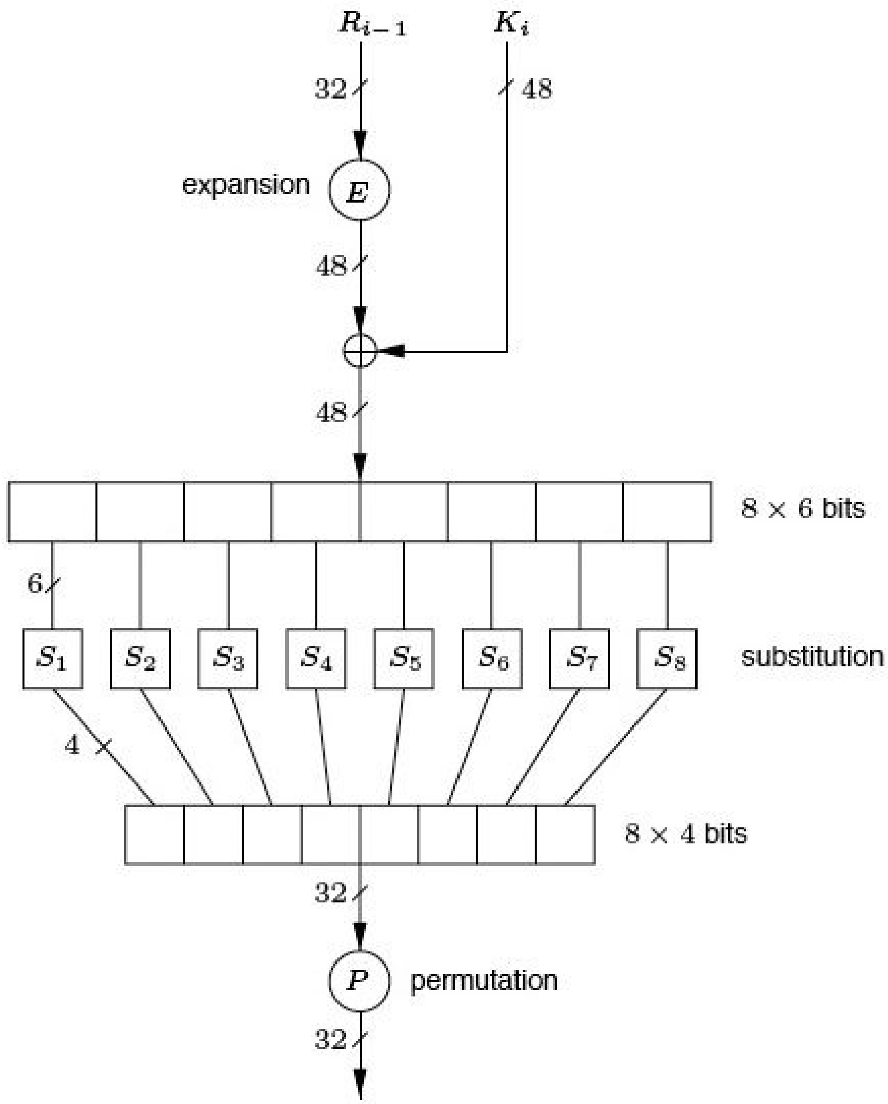


Figure 7.10: *DES inner function f.*

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

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

Table 7.2: *DES initial permutation and inverse (IP and IP^{-1}).*

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

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

Table 7.3: *DES per-round functions: expansion E and permutation P.*

row	column number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 7.8: *DES S-boxes.*

Data Encryption Standard

The key k can be 64 bits, but only 56 bits are significative (the bits 8, 16, 24, 32, 40, 48, 56 et 64 are parity bits)

These 8 bits are not considered anymore

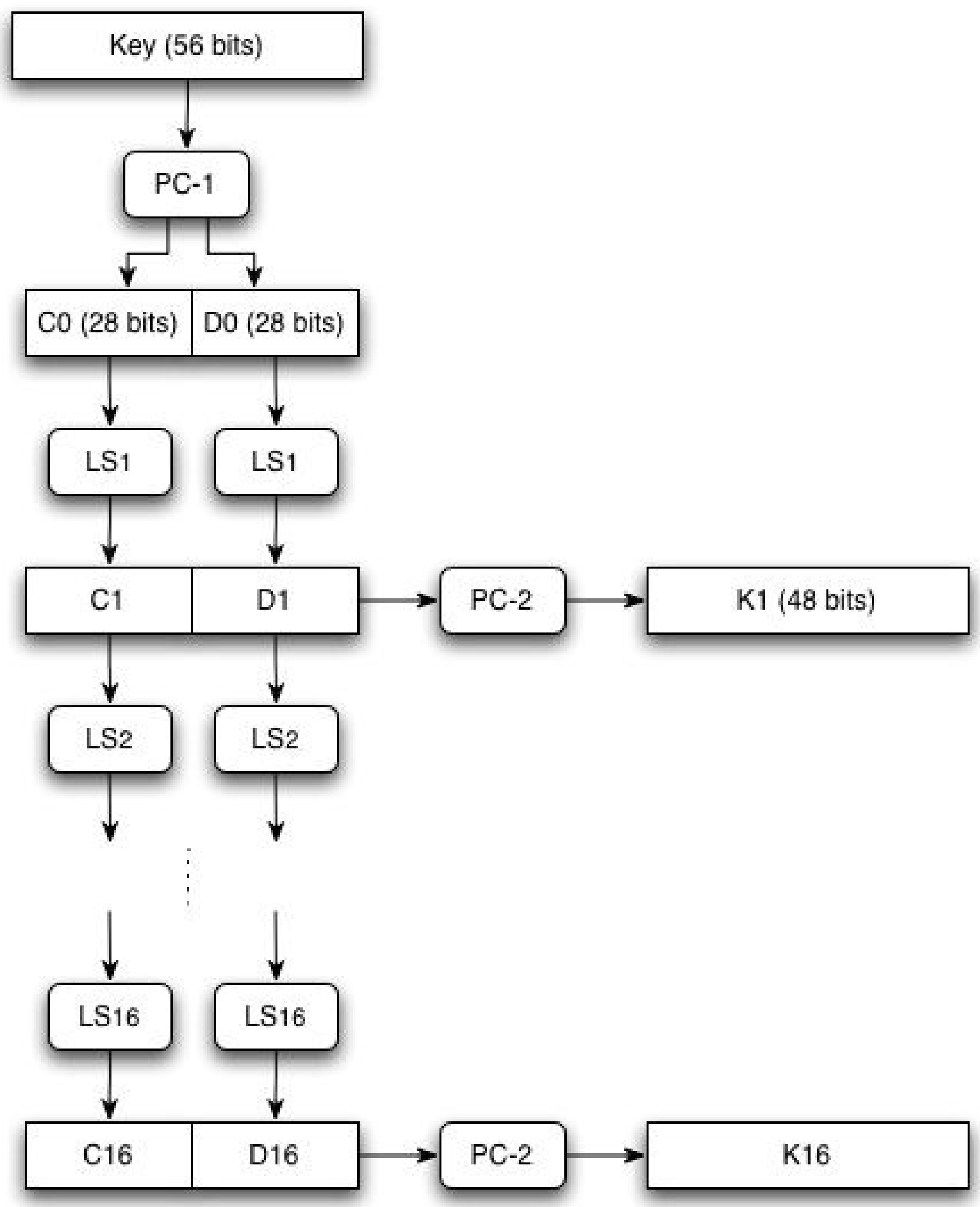
Data Encryption Standard

1. 56-bits key goes through a permutation $PC1$: $PC1(k) = C_0D_0$ where C_0 is the 28 first bits and D_0 the 28 following bits

2. forall $i \in [1, 16]$:

   - $C_i = LS_i(C_{i-1})$ and $D_i = LS_i(D_{i-1})$ where LS_i is a circular rotation to the left of one position when $i = 1, 2, 9, 16$ and a circular rotation to the left of two positions otherwise

   - $k_i = PC2(C_iD_i)$ where$PC2$ is a permutation that outpouts 48 bits



Key (56 bits)
PC-1
C0 (28 bits)
D0 (28 bits)
LS1
LS1
C1
D1
PC-2
K1 (48 bits)
LS2
LS2
LS16
LS16
C16
D16
PC-2
K16

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
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

Table 7.4: *DES key schedule bit selections (PC1 and PC2).*

Data Encryption Standard

DES has 4 weak keys = key k such that $\forall x \in M$: $E_k(E_k(x)) = x$

DES has 6 pairs of semi-weak keys = (k_1, k_2) such that $\forall x \in M$: $E_{k_1}(E_{k_2}(x)) = x$

Moreover, for the four weak keys, it exists 2^{32} messages $x \in M$ such that $E_k(x) = x$

weak key (hexadecimal)	C_0	D_0
0101 0101 0101 0101	$\{0\}^{28}$	$\{0\}^{28}$
FEFE FEFE FEFE FEFE	$\{1\}^{28}$	$\{1\}^{28}$
1F1F 1F1F 0E0E 0E0E	$\{0\}^{28}$	$\{1\}^{28}$
E0E0 E0E0 F1F1 F1F1	$\{1\}^{28}$	$\{0\}^{28}$

Table 7.5: *Four DES weak keys.*

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

Table 7.6: *Six pairs of DES semi-weak keys (one pair per line).*

$\boxed{\text{Chiffrem}^{t}\text{ de DES}}$

$$x \to IP \to L_0\,R_0 \to L_1|R_1 = R_0|L_0 \oplus f(R_0, k_1)$$

$$\to L_2|R_2 = R_1|L_1 \oplus f(R_1, k_2) \to \cdots$$

$$\to L_{15}|R_{15} = R_{14}|L_{14} \oplus f(R_{14}, k_{15})$$

$$\to R_{16}|L_{16} = L_{15} \oplus f(R_{15}, k_{16})|R_{15} \to IP^{-1} \to y$$

$\boxed{\text{Déchiffrem}^{t}\text{ de DES}}$: m̂ algorithme mais on inverse l'ordre des clés.

$$y \to IP \to \overbrace{R_{16}}^{L'_0}|\overbrace{L_{16}}^{R'_0} = \overbrace{L_{15} \oplus f(R_{15}, k_{16})}^{L'_0}|\overbrace{R_{15}}^{R'_0}$$

$$\to \overbrace{R_{15}}^{L'_1 = R'_0}|\overbrace{L_{15} \oplus f(R_{15}, k_{16}) \oplus f(R_{15}, k_{16})}^{R'_1 = L'_0 \oplus f(R'_0, \underline{\underline{k_{16}}})} = \overbrace{R_{15}}^{L'_1}|\overbrace{L_{15}}^{R'_1}$$

$$\to \overset{L'_2 = R'_1}{L_{15}}\;|\overset{R'_2 = L'_1 \oplus f(R'_1, \underline{\underline{k_{15}}})}{R_{15} \oplus f(\underbrace{R'_1}_{= L_{15} = R_{14}}, k_{15})} = R_{14}|L_{14} \oplus f(R_{14}, k_{15}) \oplus f(R_{14}, k_{15})$$

$$= \overbrace{R_{14}}^{L'_2}|\overbrace{L_{14}}^{R'_2}$$

$$\to \cdots \to \overbrace{R_1}^{L'_{15}}|\overbrace{L_1}^{R'_{15}} \to \overbrace{R_1 \oplus f(\underset{R_0}{L_1}, k_1)}^{R'_{16} = L'_{15} \oplus f(R'_{15}, \underline{\underline{k_1}})}\;|\;\overbrace{\underset{R_0}{L_1}}^{L'_{16} = R'_{15}}$$

$$= L_0 \oplus f(R_0, k_1) \oplus f(R_0, k_1)|R_0 = \overbrace{L_0}^{R'_{16}}|\overbrace{R_0}^{L'_{16}}$$

$$\to IP^{-1} \to x$$

Bloc ciphers caracteristics

An alteration of one bit of a plaintext bloc results (after encryption of the bloc) in the alteration of each bit of the corresponding ciphertext with a probability $\frac{1}{2}$

An alteration of one bit of a ciphertext bloc results (after decryption of the bloc) in the alteration of each bit of the corresponding plaintext with a probability $\frac{1}{2}$

An alteration of one bit of a key results (after encryption of a plaintext) in the alteration of each bit of the corresponding ciphertext with a probability $\frac{1}{2}$

Cryptanalysis

Differential cryptanalysis (Eli Biham and Adi Shamir, 1993): properties (caracteristics) deduced from two xored plaintexts that are encrypted

Eli Biham and Adi Shamir

Cryptanalysis

Linear cryptanalysis (Mitsuru Matsui, 1994): linear approximation of the sboxes

Mitsuru Matsui

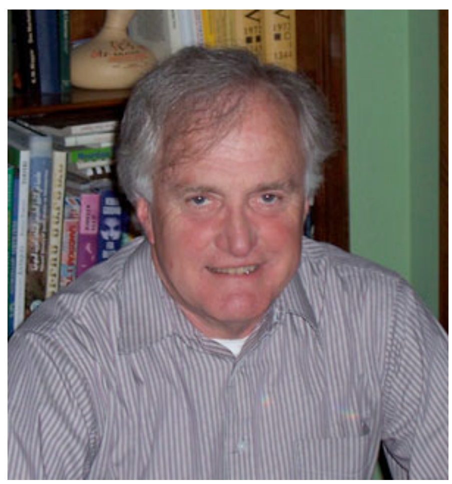
attack method	data complexity		storage complexity	processing complexity
	known	chosen		
exhaustive precomputation	—	1	2^{56}	1 (table lookup)
exhaustive search	1	—	negligible	2^{55}
linear cryptanalysis	2^{43} (85%)	—	for texts	2^{43}
	2^{38} (10%)	—	for texts	2^{50}
differential cryptanalysis	—	2^{47}	for texts	2^{47}
	2^{55}	—	for texts	2^{55}

Table 7.7: *DES strength against various attacks.*

Other bloc ciphers

IDEA

X. Lai et J. Massey *A proposal for a new block encryption standard*, Eurocrypt'90, Lecture notes in computer science, volume 473, Springer-Verlag 1991

Xuejia Lai and James Massey

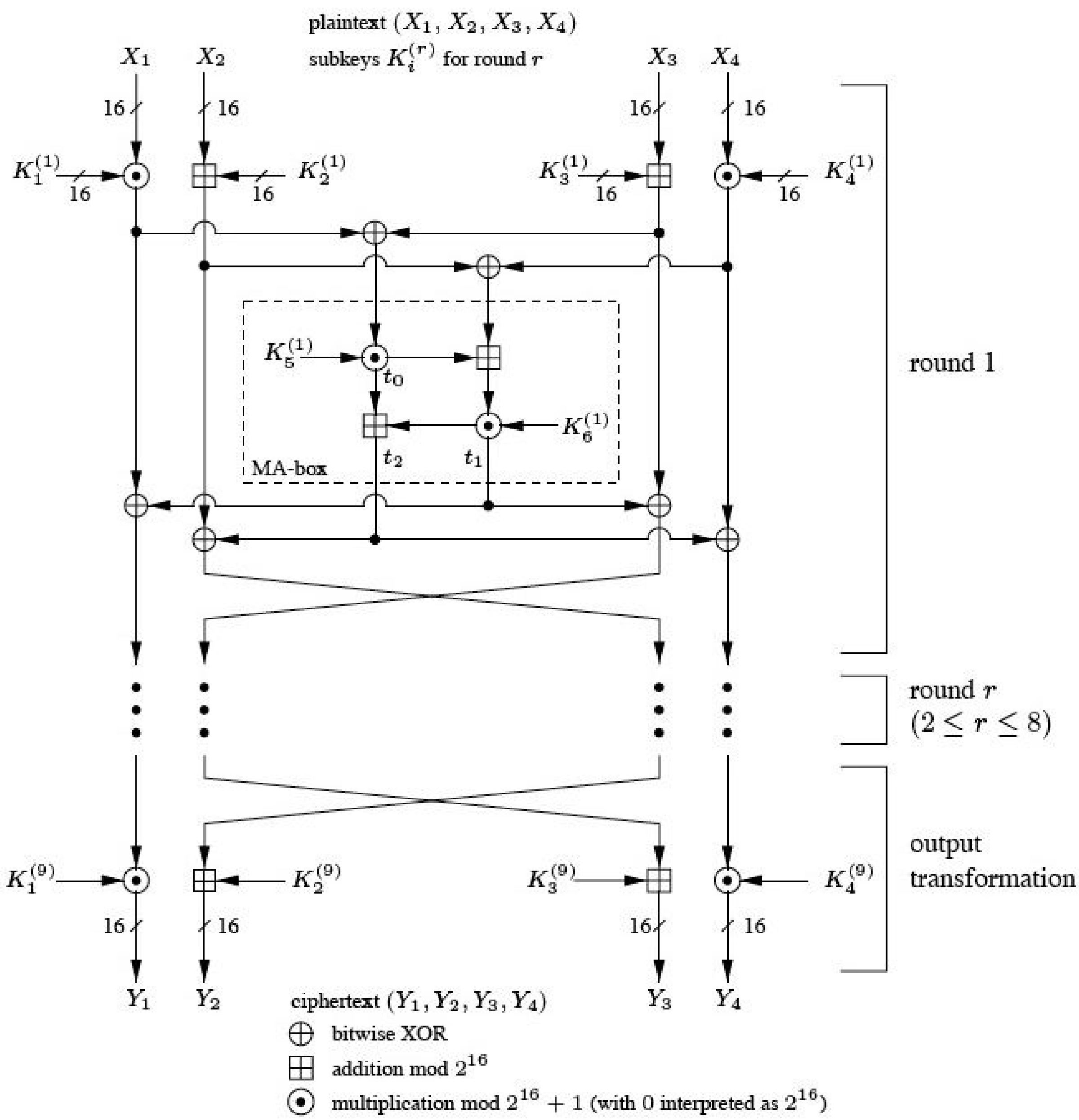


Figure 7.11: *IDEA computation path.*

Algorithm IDEA encryption

INPUT: 64-bit plaintext $M = m_1 \dots m_{64}$; 128-bit key $K = k_1 \dots k_{128}$.
OUTPUT: 64-bit ciphertext block $Y = (Y_1, Y_2, Y_3, Y_4)$. (For decryption, see Note 7.103.)

1. (key schedule) Compute 16-bit subkeys $K_1^{(r)}, \dots, K_6^{(r)}$ for rounds $1 \le r \le 8$, and $K_1^{(9)}, \dots, K_4^{(9)}$ for the output transformation, using Algorithm 7.102.
2. $(X_1, X_2, X_3, X_4) \leftarrow (m_1 \dots m_{16}, m_{17} \dots m_{32}, m_{33} \dots m_{48}, m_{49} \dots m_{64})$, where X_i is a 16-bit data store.
3. For round r from 1 to 8 do:
   (a) $X_1 \leftarrow X_1 \odot K_1^{(r)}$, $X_4 \leftarrow X_4 \odot K_4^{(r)}$, $X_2 \leftarrow X_2 \boxplus K_2^{(r)}$, $X_3 \leftarrow X_3 \boxplus K_3^{(r)}$.
   (b) $t_0 \leftarrow K_5^{(r)} \odot (X_1 \oplus X_3)$, $t_1 \leftarrow K_6^{(r)} \odot (t_0 \boxplus (X_2 \oplus X_4))$, $t_2 \leftarrow t_0 \boxplus t_1$.
   (c) $X_1 \leftarrow X_1 \oplus t_1$, $X_4 \leftarrow X_4 \oplus t_2$, $a \leftarrow X_2 \oplus t_2$, $X_2 \leftarrow X_3 \oplus t_1$, $X_3 \leftarrow a$.
4. (output transformation) $Y_1 \leftarrow X_1 \odot K_1^{(9)}$, $Y_4 \leftarrow X_4 \odot K_4^{(9)}$, $Y_2 \leftarrow X_3 \boxplus K_2^{(9)}$, $Y_3 \leftarrow X_2 \boxplus K_3^{(9)}$.

Algorithm IDEA key schedule (encryption)

INPUT: 128-bit key $K = k_1 \dots k_{128}$.
OUTPUT: 52 16-bit key sub-blocks $K_i^{(r)}$ for 8 rounds r and the output transformation.

1. Order the subkeys $K_1^{(1)} \dots K_6^{(1)}, K_1^{(2)} \dots K_6^{(2)}, \dots, K_1^{(8)} \dots K_6^{(8)}, K_1^{(9)} \dots K_4^{(9)}$.
2. Partition K into eight 16-bit blocks; assign these directly to the first 8 subkeys.
3. Do the following until all 52 subkeys are assigned: cyclic shift K left 25 bits; partition the result into 8 blocks; assign these blocks to the next 8 subkeys.

Other bloc ciphers

RC5

R. Rivest *The RC5 encryption algorithm*, Fast Software Encryption, Lecture note in computer science, volume 1008, Springer-Verlag 1995

Ronald Rivest

Algorithm RC5 encryption (w-bit wordsize, r rounds, b-byte key)

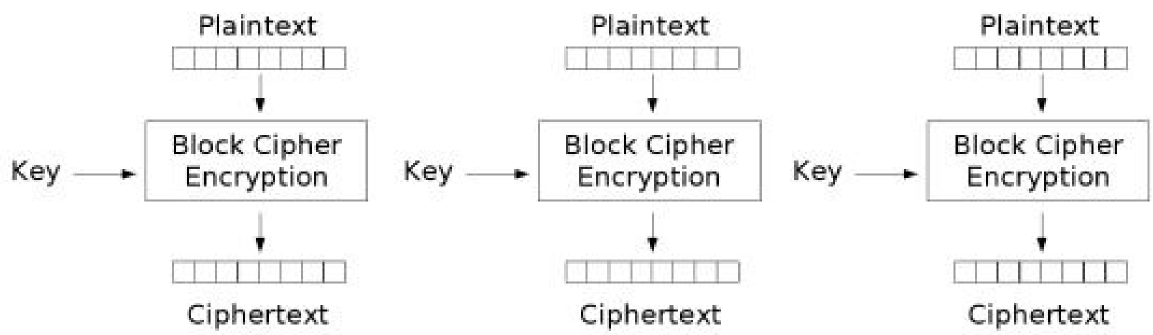
INPUT: $2w$-bit plaintext $M = (A, B)$; r; key $K = K[0] \dots K[b-1]$.
OUTPUT: $2w$-bit ciphertext C. (For decryption, see Note 7.117.)

1. Compute $2r + 2$ subkeys $K_0, \dots, K_{2r+1}$ by Algorithm 7.116 from inputs K and r.
2. $A \leftarrow A \boxplus K_0$, $B \leftarrow B \boxplus K_1$. (Use addition modulo 2^w.)
3. For i from 1 to r do: $A \leftarrow ((A \oplus B) \hookleftarrow B) \boxplus K_{2i}$, $B \leftarrow ((B \oplus A) \hookleftarrow A) \boxplus K_{2i+1}$.
4. The output is $C \leftarrow (A, B)$.

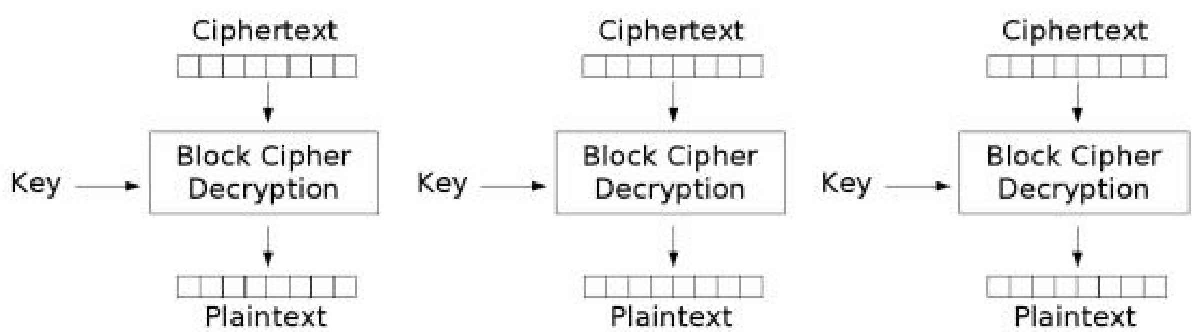
Algorithm RC5 key schedule

INPUT: word bitsize w; number of rounds r; b-byte key $K[0] \dots K[b-1]$.
OUTPUT: subkeys $K_0, \dots, K_{2r+1}$ (where K_i is w bits).

1. Let $u = w/8$ (number of bytes per word) and $c = \lceil b/u \rceil$ (number of words K fills). Pad K on the right with zero-bytes if necessary to achieve a byte-count divisible by u (i.e., $K[j] \leftarrow 0$ for $b \le j \le c \cdot u - 1$). For i from 0 to $c - 1$ do: $L_i \leftarrow \sum_{j=0}^{u-1} 2^{8j} K[i \cdot u + j]$ (i.e., fill L_i low-order to high-order byte using each byte of $K[\cdot]$ once).
2. $K_0 \leftarrow P_w$; for i from 1 to $2r + 1$ do: $K_i \leftarrow K_{i-1} \boxplus Q_w$. (Use Table 7.14.)
3. $i \leftarrow 0, j \leftarrow 0, A \leftarrow 0, B \leftarrow 0, t \leftarrow \max(c, 2r + 2)$. For s from 1 to $3t$ do:
   (a) $K_i \leftarrow (K_i \boxplus A \boxplus B) \hookleftarrow 3$, $A \leftarrow K_i$, $i \leftarrow i + 1 \bmod (2r + 2)$.
   (b) $L_j \leftarrow (L_j \boxplus A \boxplus B) \hookleftarrow (A \boxplus B)$, $B \leftarrow L_j$, $j \leftarrow j + 1 \bmod c$.
4. The output is $K_0, K_1, \dots, K_{2r+1}$. (The L_i are not used.)

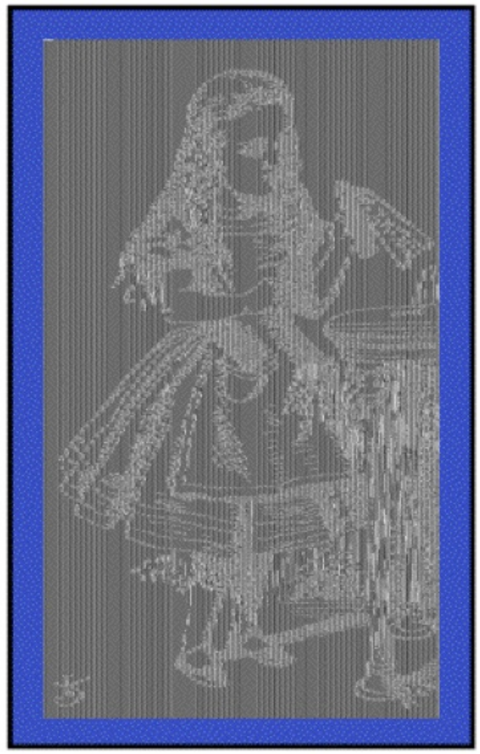


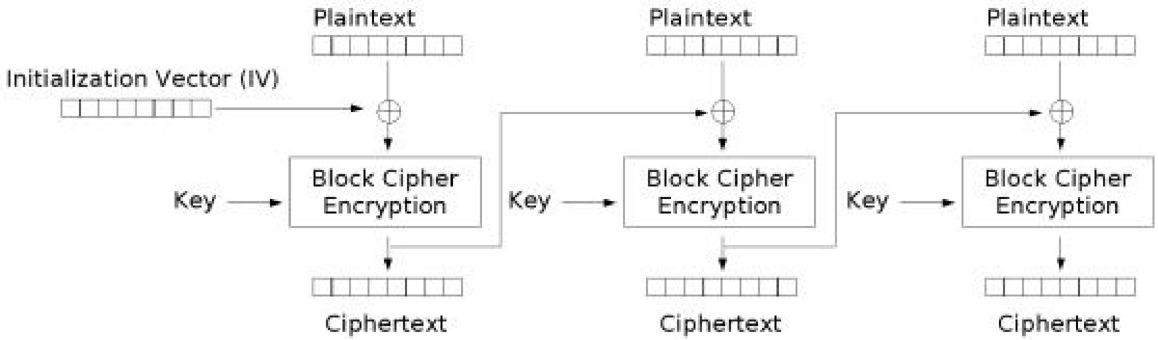
Electronic Codebook (ECB) mode encryption



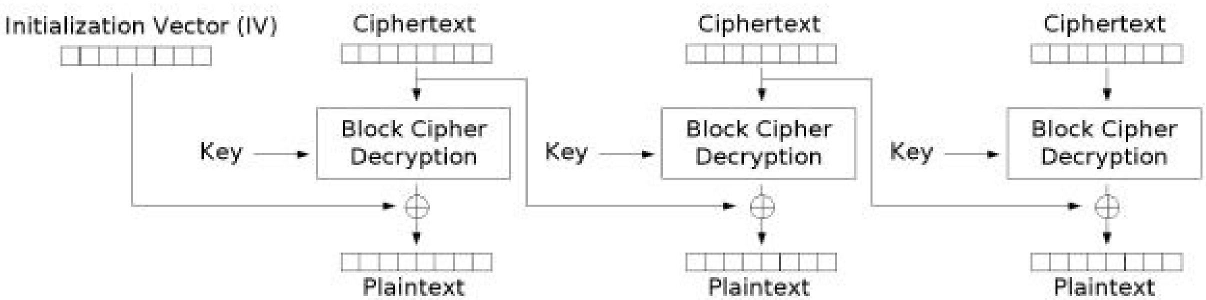
Electronic Codebook (ECB) mode decryption


DRINK ME

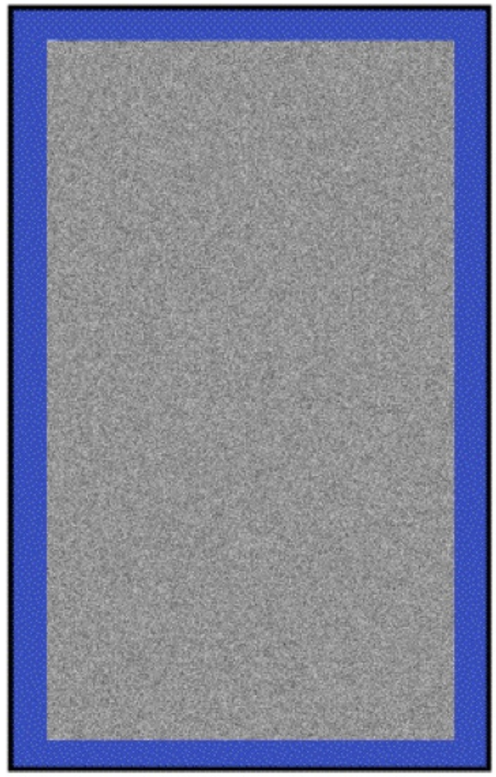


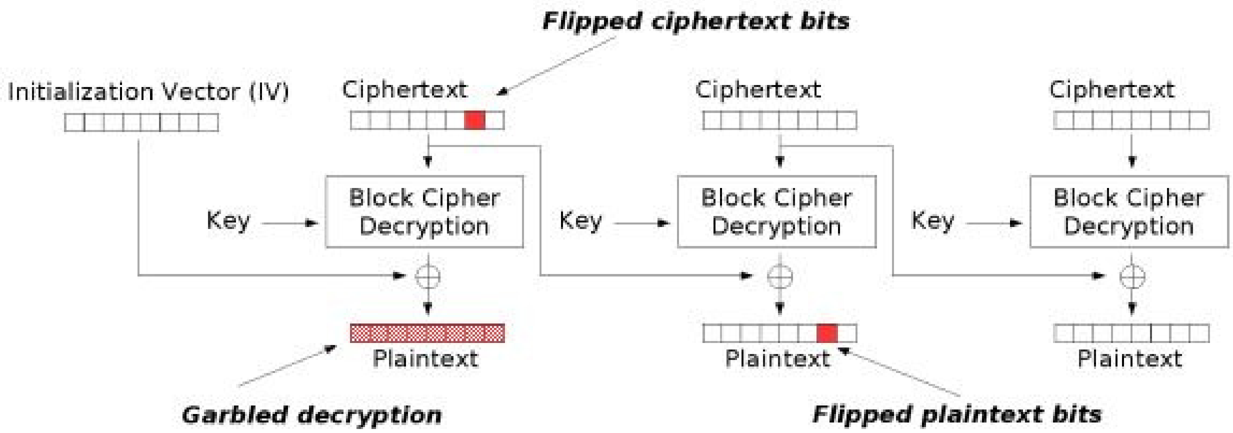


Cipher Block Chaining (CBC) mode encryption

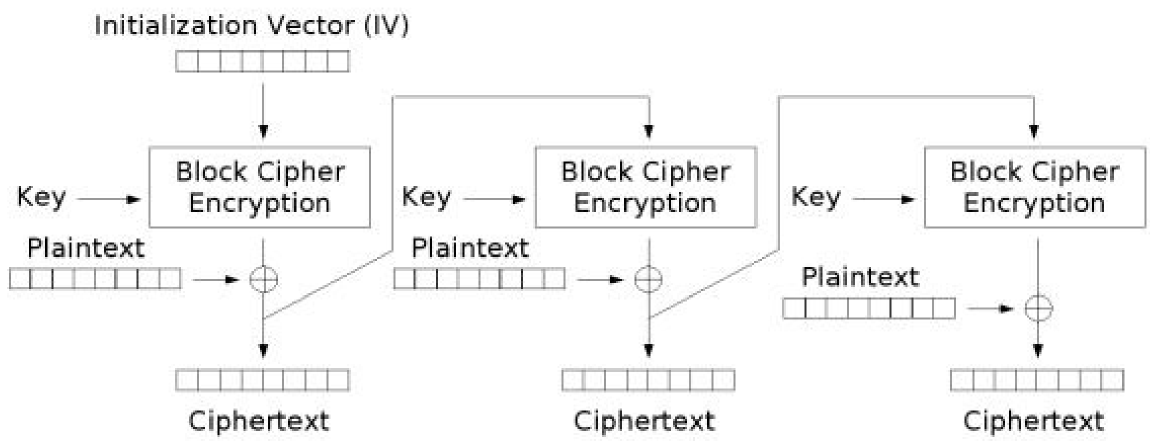


Cipher Block Chaining (CBC) mode decryption

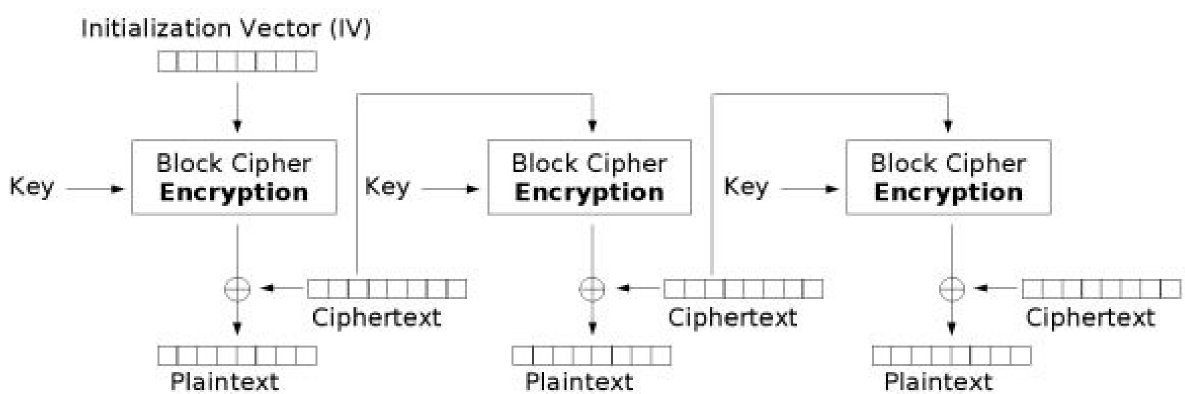




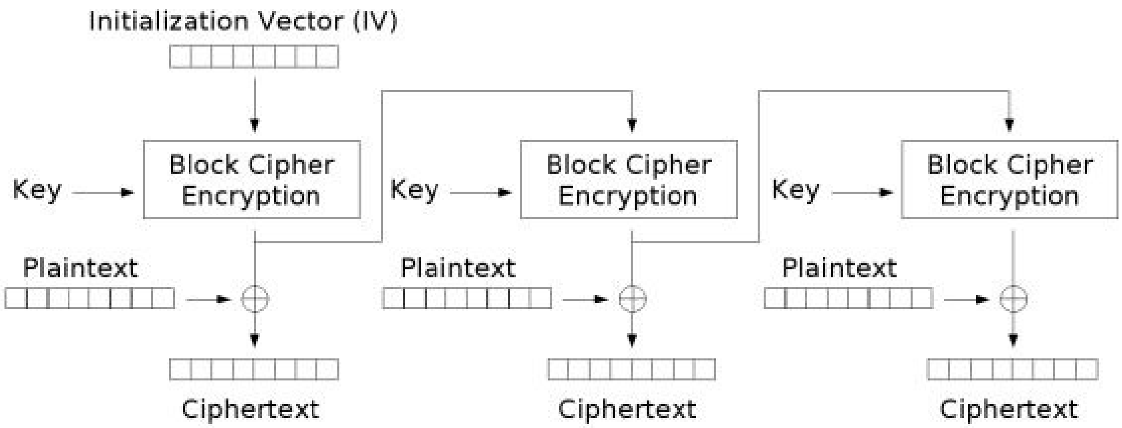
Flipped ciphertext bits
Initialization Vector (IV)
Ciphertext
Ciphertext
Ciphertext
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Plaintext
Plaintext
Plaintext
Garbled decryption
Flipped plaintext bits



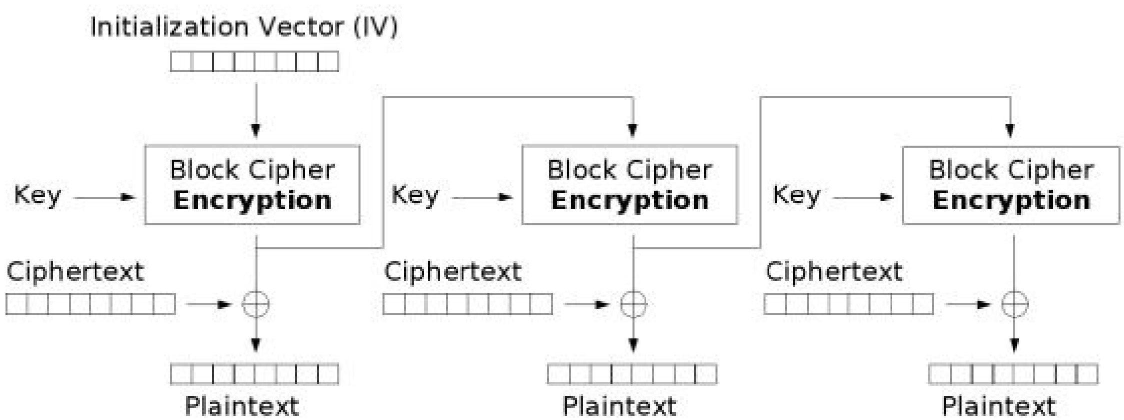
Cipher Feedback (CFB) mode encryption



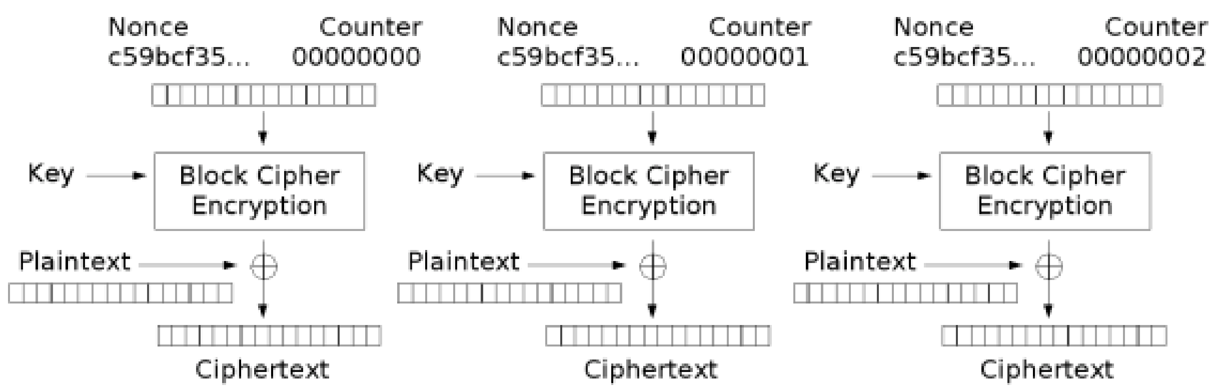
Cipher Feedback (CFB) mode decryption



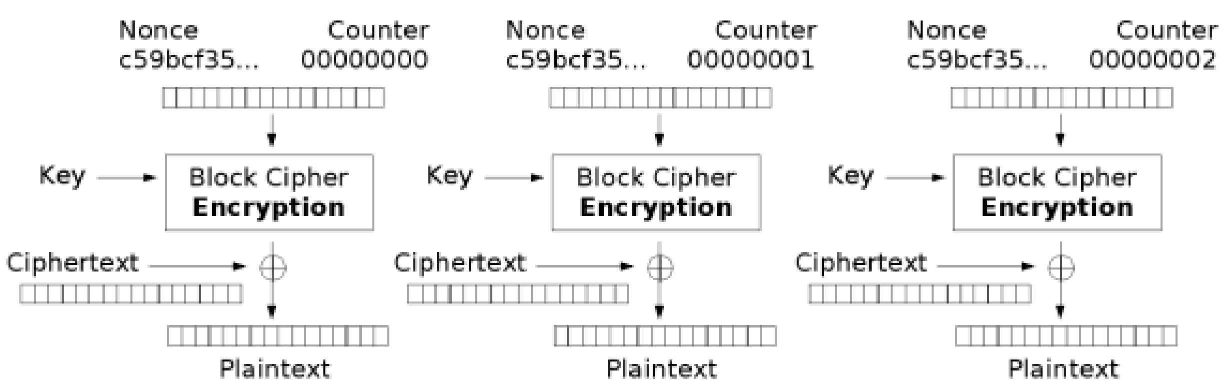
Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption



Counter (CTR) mode encryption



Counter (CTR) mode decryption

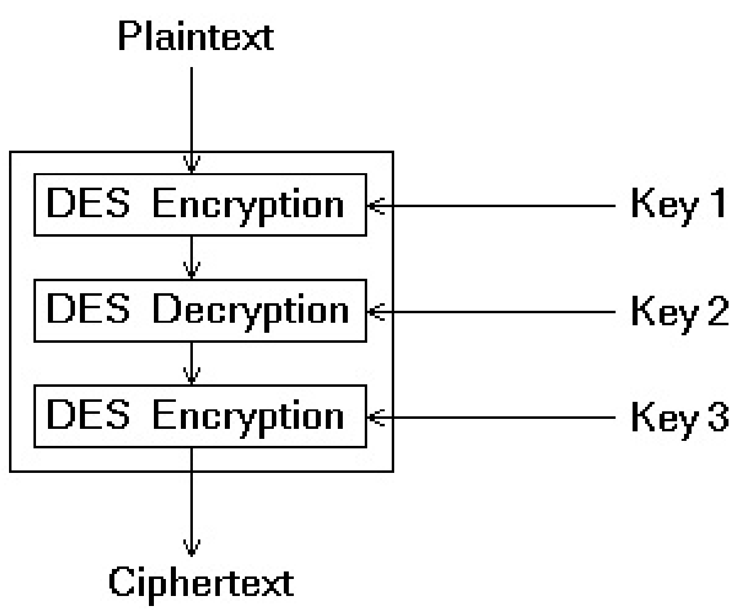
Triple-DES

Brute force attack on DES achievable

DES Challenges (1997-1999): EFF's Deep Crack and Distributed.net

First reaction: 3DES = $E_{k_3}(D_{k_2}(E_{k_1}(x)))$

where E=DES and D=DES^{-1}



Advanced Encryption Standard

10 candidates at the first round, 5 candidates at the last round:

- RC6 (RSA Lab, USA)
- Rijndael (UE)
- Twofish (USA)
- Serpent (UE)
- MARS (IBM, USA)

http://csrc.nist.gov/encryption/aes

AES: Rijndael

Joan Daemen and Vincent Rijmen

Rijndael in 1997 - AES is a standard in 2000

AES: Rijndael

Iterative bloc cipher: 128-bit blocs with:

- 128-bits key $\rightarrow$ 10 rounds

- 192-bits key $\rightarrow$ 12 rounds

- 256-bits key $\rightarrow$ 14 rounds

Here: 128-bits blocs and 128-bits keys

AES - State

State = plaintext, inner structure and ciphertext

4×4 table of bytes:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

Remark: state has always 4 lines, the number of columns is the key size divided by 32

AES - Assignation

$state = x$ is realized as following:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}$$

where x_i is the i^{th} byte of x

AES - ByteSub

ByteSub: non-linear substitution applied on a byte

Each byte of state is transformed by a sbox

The sbox has an algebrical explanation:

```
byte ByteSub(byte z)
{
   if(z!=0)
      z=z^-1 in GF(2^8)
   c=011000111
   for(i=0;i<8;++i)
      b[i]=z[i]+z[i+4]+z[i+5]+
           z[i+6]+z[i+7]+c[i] mod 2
   return(b)
}
```

where z is the byte to be transformed

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 7. S-box: substitution values for the byte `xy` (in hexadecimal format).

AES - ShiftRow

State is modified in the following way:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

AES - MixColumn

The columns of state are modified as following:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

where the elements are expressed in hexadecimal and s_i is the i^{th} column of state

AES - Sub-keys

At each round, a roundkey is derivated from the main secret key

AddRoundKey: is the binary xor between state and the roundkey

State and roundkey are two 4×4 table of bytes

The bytes located at the same coordinates in state and roundkey are xored (bit by bit), and the result is stored at the same place in state

AES - Sub-keys

AES needs 11 roundkeys

The main 128-bit secret key is expanded into an 1408-bits expandedkey (thanks to the algorithm KeyExpansion)

The 11 roundkeys are extracted from the expandedkey

The output of the KeyExpansion algorithm is composed by 44 $\times$ 32 bits $=$ 1408 bits ($w[0] \ldots w[43]$)

In the KeyExpansion algorithm, $key[i]$ is the i^{th} byte of the main secret key

AES - Sub-keys

$Rotword(B_0, B_1, B_2, B_3) \ = \ (B_1, B_2, B_3, B_0)$
where B_i is a byte

$Subword(B_0, B_1, B_2, B_3) \ = \ (B'_0, B'_1, B'_2, B'_3)$
where B'_i is the result of the sbox applied to B_i

```
proc KeyExpansion(K↓, w↑);
begin   external RotWord, SubWord;
        RCon[1] := 01000000;  RCon[2] := 02000000;
        RCon[3] := 04000000;  RCon[4] := 08000000;
        RCon[5] := 10000000;  RCon[6] := 20000000;
        RCon[7] := 40000000;  RCon[8] := 80000000;
        RCon[9] := 1B000000;  RCon[10] := 36000000;
        for i := 0 to 3 do w[i] := (K[4i], K[4i+1], K[4i+2], K[4i+3]) endfor;
        for i := 4 to 43 do
          temp := w[i−1];
          if i≡0(mod 4) → temp := SubWord(RotWord(temp)) ⊛ RCon[i/4] ▯ else → skip fi;
          w[i] := w[i−4] ⊛ temp
        endfor;
        return(w[0] ... w[43])
end
```

AES - Encryption

```
void AES(state &,key)
{
  KeyExpansion(key,expandedkey)
  AddRoundKey(state,expandedkey)
  for(i=1;i<10;++i)
  {
    ByteSub(state)
    ShiftRow(state)
    MixColumn(state)
    AddRoundKey(state,expandedkey+4*i)
  }
  ByteSub(state)
  ShiftRow(state)
  AddRoundKey(state,expandedkey+4*10)
}
```

The final content of state is the ciphertext

AES - Decryption

```
void InvAES(state &,key)
{
  KeyExpansion(key,expandedkey)
  AddRoundKey(state,expandedkey+4*10)
  for(i=9;i>=1;i=i-1)
  {
    InvShiftRow(state)
    InvByteSub(state)
    AddRoundKey(state,expandedkey+4*i)
    InvMixColumn(state)
  }
  InvShiftRow(state)
  InvByteSub(state)
  AddRoundKey(state,expandedkey)
}
```

The final content of $state$ is the plaintext

InvShiftRow realizes the inverse circular rotations and AddRoundKey is its own inverse

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 14. Inverse S-box: substitution values for the byte `xy` (in hexadecimal format).

`InvMixColumns()` is the inverse of the **`MixColumns()`** transformation. **`InvMixColumns()`** operates on the State column-by-column, treating each column as a four-term polynomial as described in Sec. 4.3. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by

$$a^{-1}(x) = \{\texttt{0b}\}x^3 + \{\texttt{0d}\}x^2 + \{\texttt{09}\}x + \{\texttt{0e}\}. \tag{5.9}$$

As described in Sec. 4.3, this can be written as a matrix multiplication. Let

$s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}. \tag{5.10}$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{\texttt{0e}\} \bullet s_{0,c}) \oplus (\{\texttt{0b}\} \bullet s_{1,c}) \oplus (\{\texttt{0d}\} \bullet s_{2,c}) \oplus (\{\texttt{09}\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{\texttt{09}\} \bullet s_{0,c}) \oplus (\{\texttt{0e}\} \bullet s_{1,c}) \oplus (\{\texttt{0b}\} \bullet s_{2,c}) \oplus (\{\texttt{0d}\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{\texttt{0d}\} \bullet s_{0,c}) \oplus (\{\texttt{09}\} \bullet s_{1,c}) \oplus (\{\texttt{0e}\} \bullet s_{2,c}) \oplus (\{\texttt{0b}\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{\texttt{0b}\} \bullet s_{0,c}) \oplus (\{\texttt{0d}\} \bullet s_{1,c}) \oplus (\{\texttt{09}\} \bullet s_{2,c}) \oplus (\{\texttt{0e}\} \bullet s_{3,c})$$