Introduction to Information Security

Lecture 3: Block and Stream Ciphers

2007. 6.

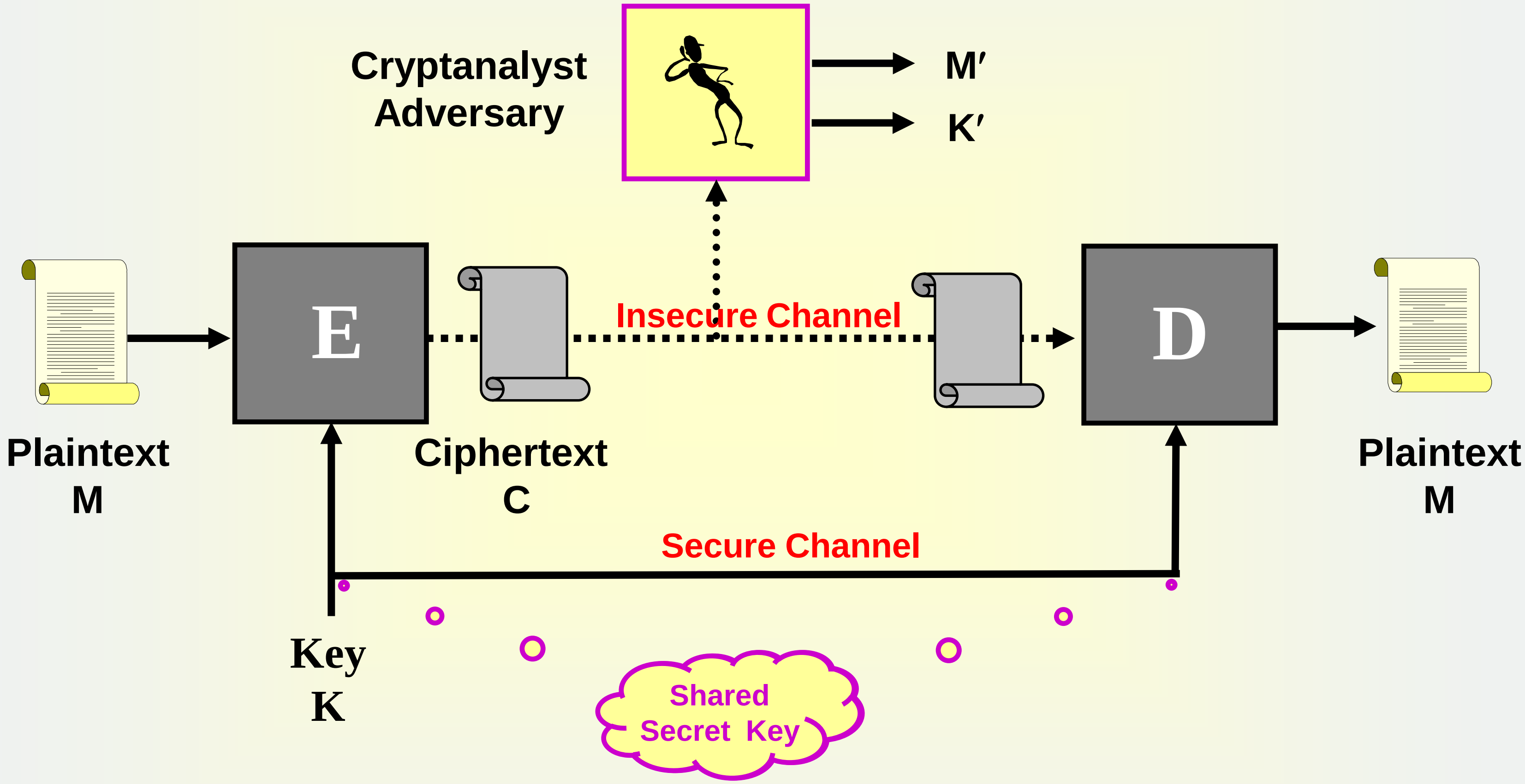
Prof. Byoungcheon Lee
sultan (at) joongbu . ac . kr

Information and Communications University

Contents

1. Feistel Network
2. DES
3. SEED
4. AES
5. Mode of operation
6. Cryptanalysis – DC, LC
7. Stream ciphers

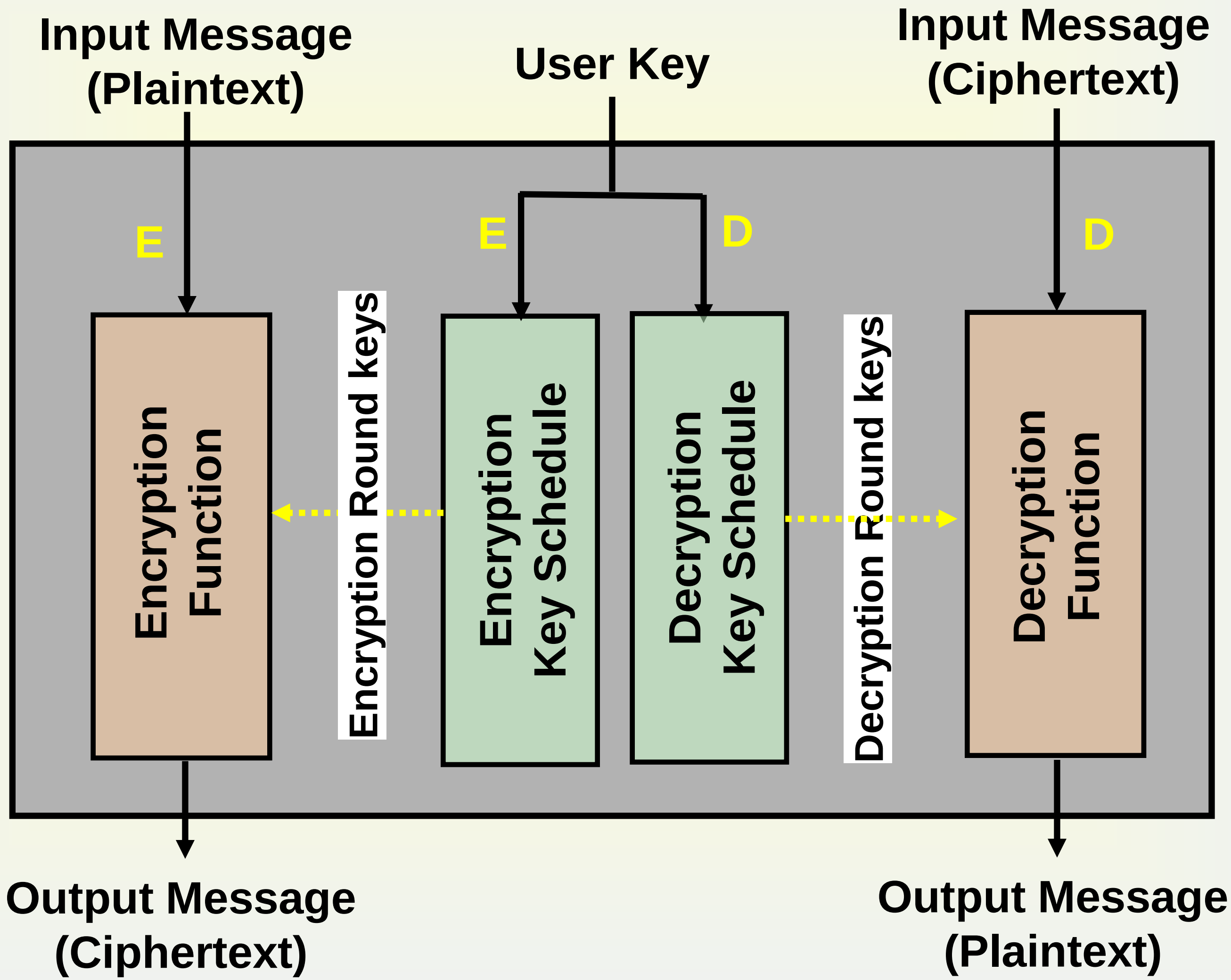
Symmetric Encryption Model



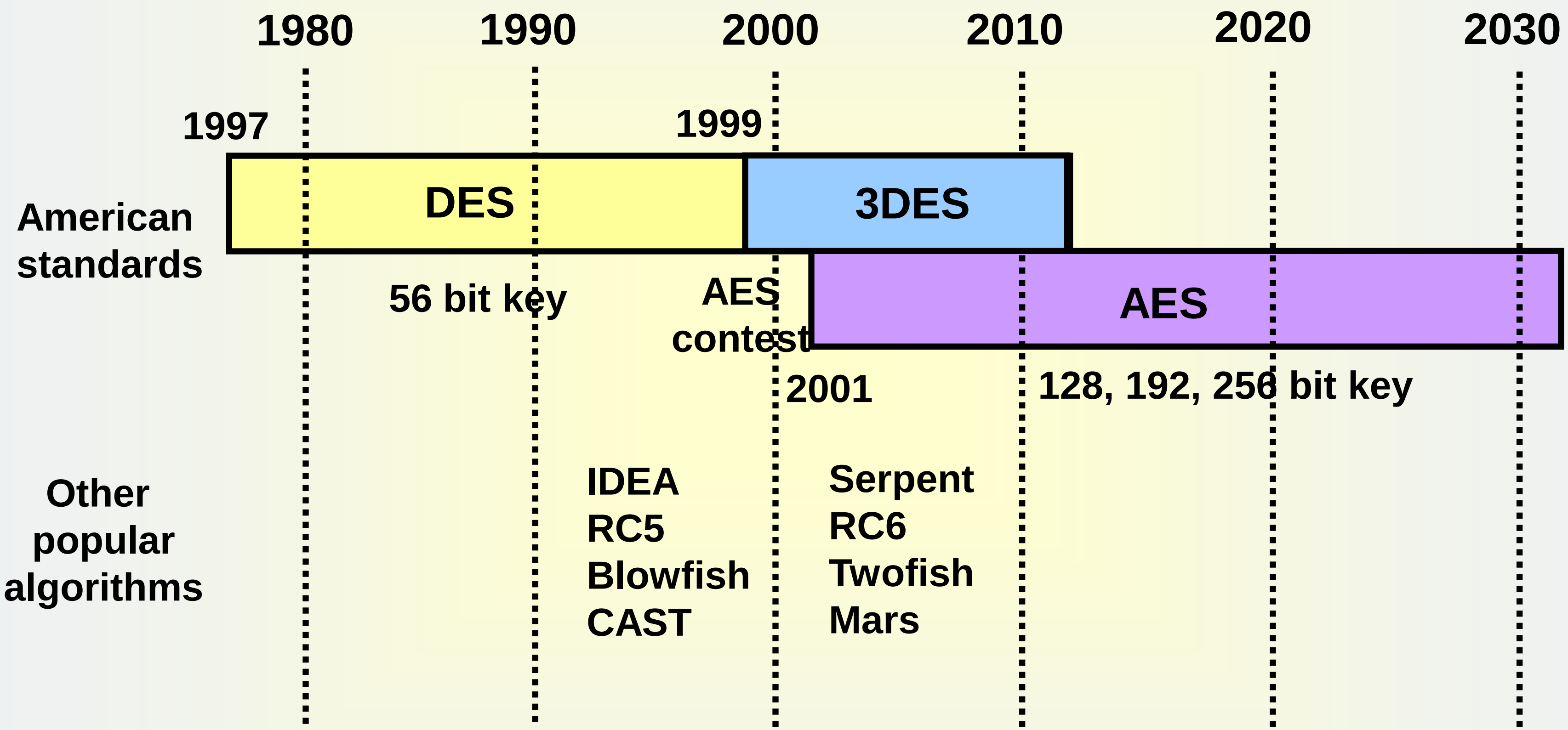
$C = E_K(M)$

$D_K(C) = M$

Block Cipher – A Simplified View



Most Popular Symmetric Ciphers



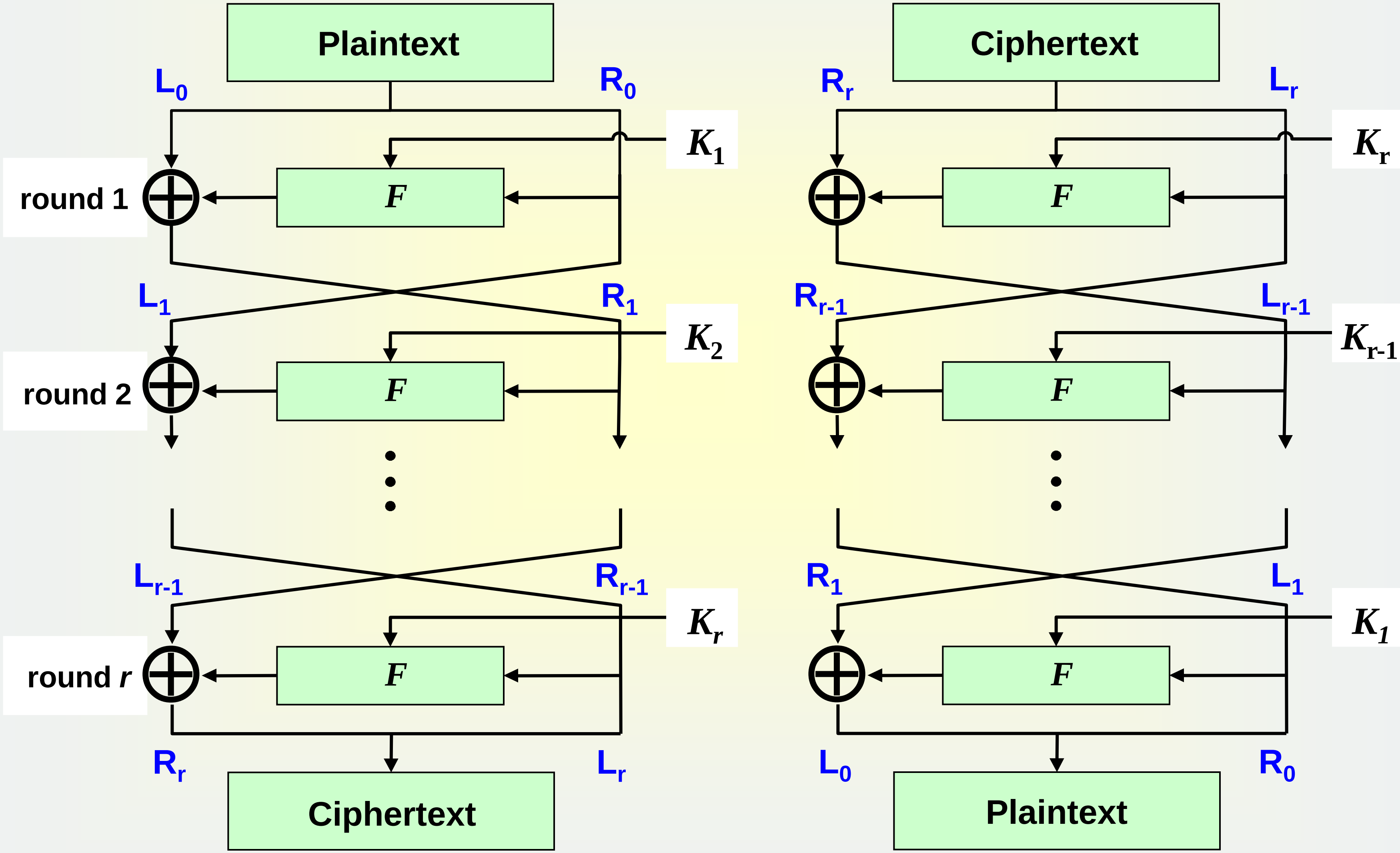
1. Feistel Network

Feistel-type Ciphers

- **Feistel network**
  - **An elegant variant of S-P networks that could be implemented using a single algorithm for both encryption and decryption**
  - **It is always a permutation regardless of the form of the *F( )* function**
  - ***F( )* does not need to be invertible**

Horst Feistel

Block Cipher Architecture : Feistel-type



Feistel-type Cipher

Encryption	Decryption
$P = L_0 \| R_0$	$C = R_r \| L_r$
$L_1 = R_0$ $R_1 = L_0 \oplus F(K_1, R_0)$	$R_{r-1} = L_r$ $L_{r-1} = R_r \oplus F(K_r, R_{r-1})$
$L_i = R_{i-1}$ $R_i = L_{i-1} \oplus F(K_i, R_{i-1})$	$R_{i-1} = L_i$ $L_{i-1} = R_i \oplus F(K_i, R_{i-1})$
$L_r = R_{r-1}$ $R_r = L_{r-1} \oplus F(K_r, R_{r-1})$	$R_0 = L_1$ $L_0 = R_1 \oplus F(K_1, R_0)$
$C = R_r \| L_r$	$P = L_0 \| R_0$

Encryption	Decryption

$P = L_0 \| R_0$
for i=1 to r
    $L_i = R_{i-1}$
    $R_i = L_{i-1} \oplus F(K_i, R_{i-1})$
$C = R_r \| L_r$
	$C = R_r \| L_r$
	for i=r-1 to 0
	    $R_i = L_{i+1}$
	    $L_i = R_{i+1} \oplus F(K_{i+1}, R_i)$
	$P = L_0 \| R_0$

Design of Feistel-type Ciphers

- **Design of F-function**
  - ✓ The only non-linear part in the Feistel-type cipher
  - ✓ Need not be invertible
  - ✓ Typically uses S-boxes (Substitution boxes) for non-linearity
  - ✓ May also contain mixing (permutation) part of the S-box outputs
  - ✓ Determines the ultimate security

- **Design of Key scheduling algorithm**
  - ✓ Algorithm for deriving as many round keys as necessary from a fixed user key
  - ✓ On-the-fly vs. off-line calculation

- **Number of rounds**
  - ✓ Depends on the strength of round function (F-function)
  - ✓ A safety margin should be considered for long-term security
  - ✓ Determined through the analysis of the whole algorithm against most powerful known cryptanalysis techniques

Lucifer

- Feistel-type Block Cipher
- Developed by H. Feistel, W. Notz, and J. L. Smith at IBM Watson research lab. in the early 1970s.
- 128-bit message space
- 128-bit ciphertext space
- 128-bit key space
- 16 rounds

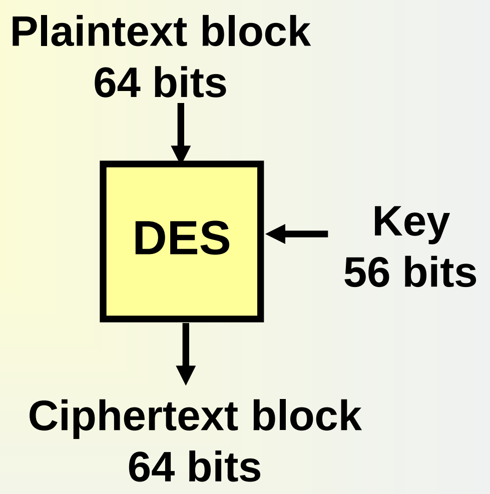
2. Data Encryption Standard

Data Encryption Standard (DES)

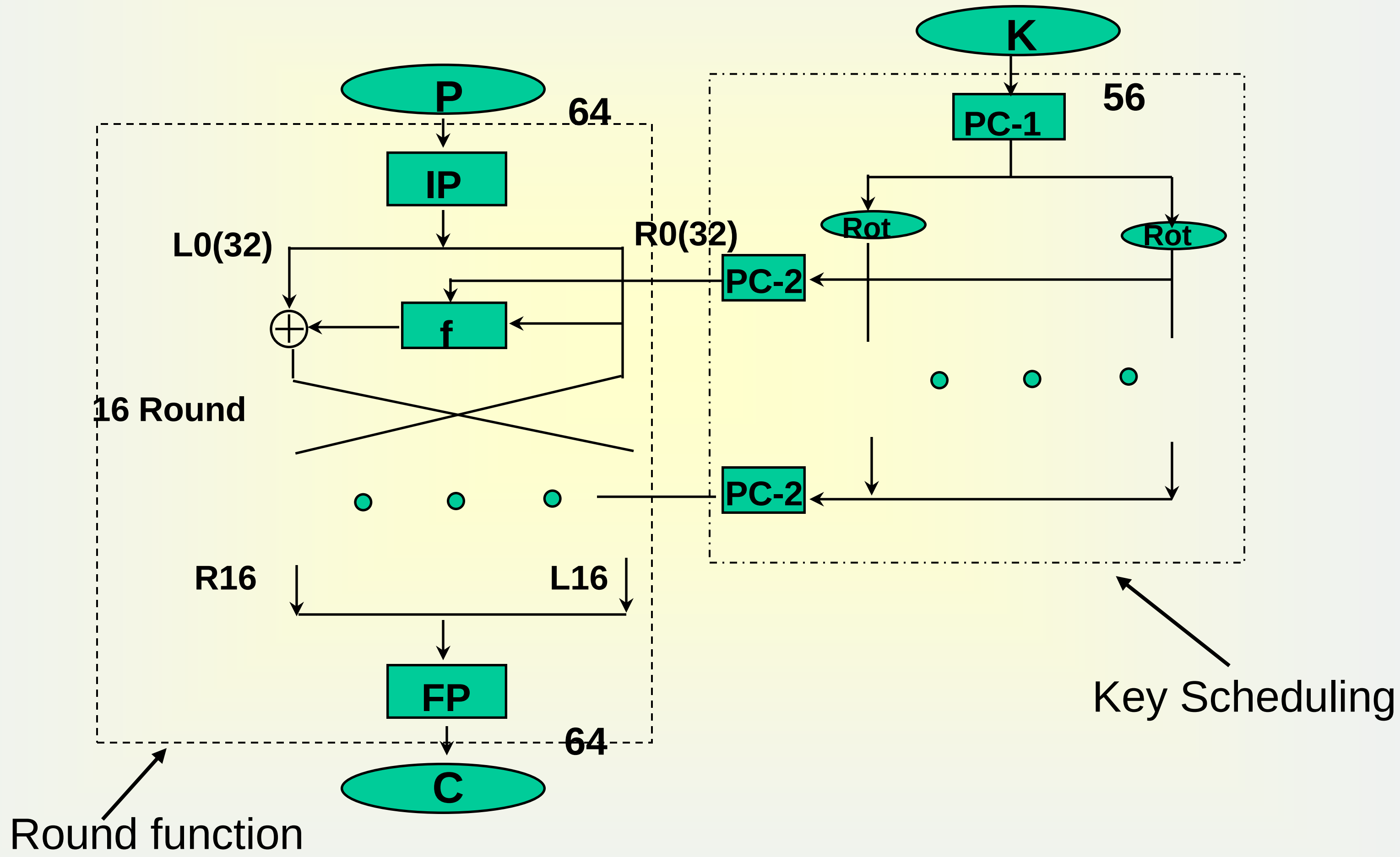
- **DES - History**
  - ✓ 1976 – adopted as a federal standard
  - ✓ 1977 – official publication as FIPS PUB 46
  - ✓ 1983, 1987, 1993 – recertified for another 5 years

* Federal Information Processing Standards

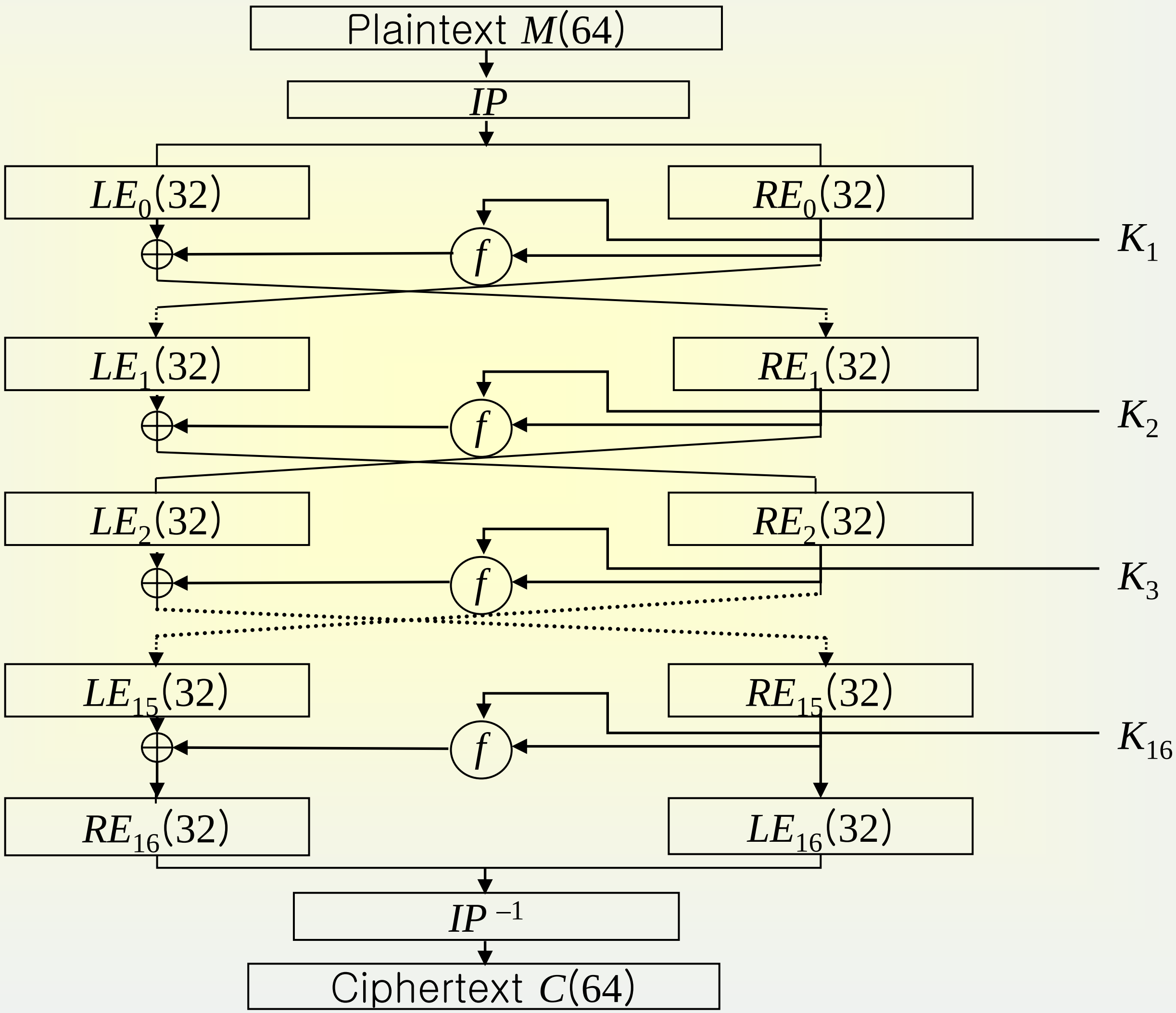
- **Design Criteria of DES**
  - ✓ Provide a high level of security
  - ✓ Completely specify and easy to understand
  - ✓ *Security must depend on hidden key, not algorithm*
  - ✓ Available to all users
  - ✓ Adaptable for use in diverse applications
  - ✓ Economically implementable in electronic device
  - ✓ Able to be validated
  - ✓ Exportable



DES Overview



DES Overview



Initial Permutation and Final Permutation

IP **(Initial permutation)**

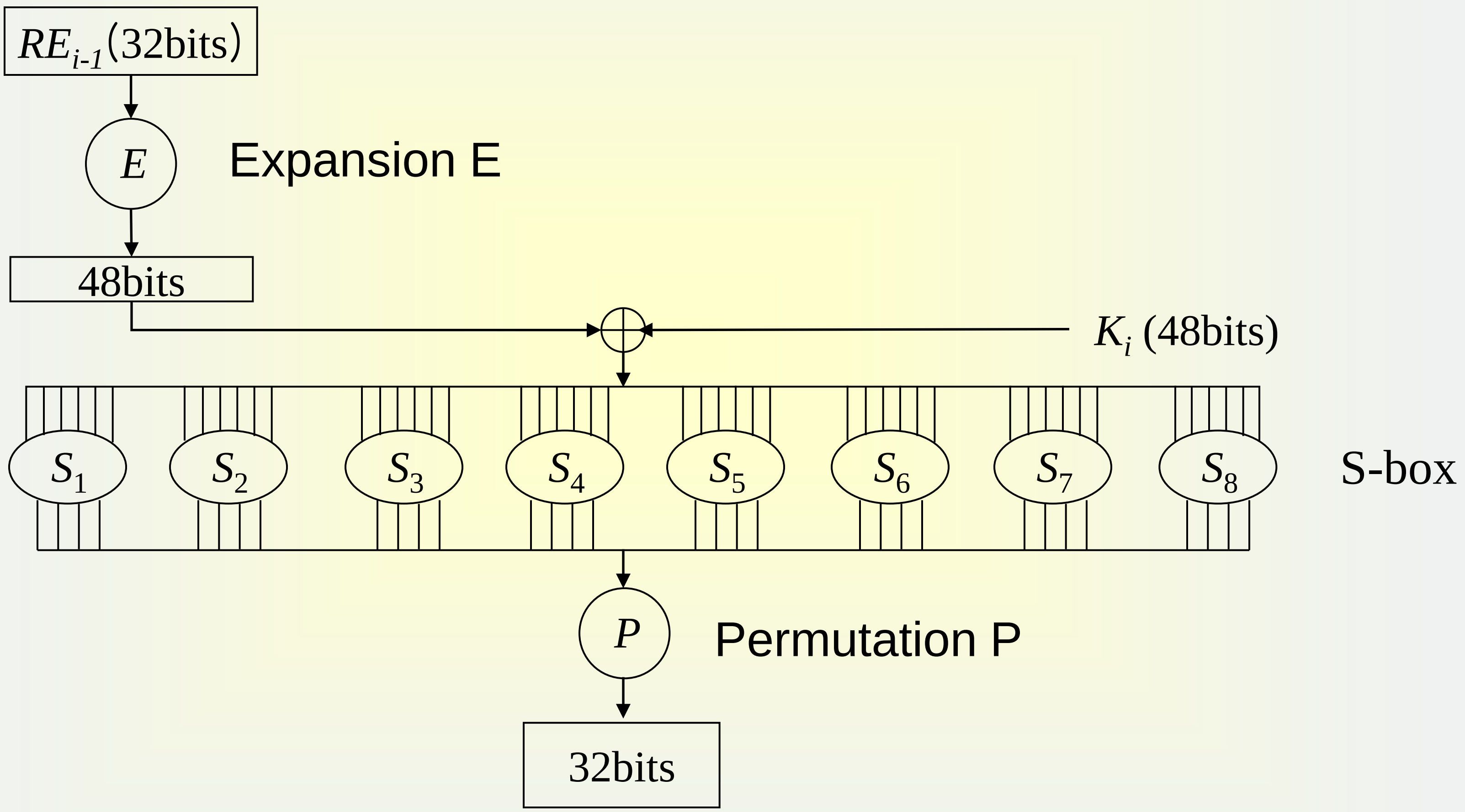
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1} (Final permutation)

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

cf.) The 58th bit of x is the first bit of IP(x)

Function $f(k_i, RE_{i-1})$



Expansion E and Permutation P

Expansion *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

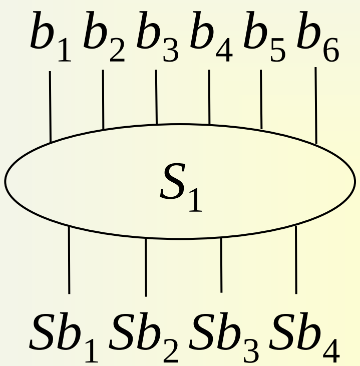
cf.) 32-bits are expanded into 48-bits.
Some bits are selected more than once.

Permutation *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

32-bit → 32-bit
permutation

S-box (substitution box)



Look-up a value from the table using

$b_1 b_6$: row

$b_2 b_3 b_4 b_5$: column

$b_1 b_6$: row

S_1-box table

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

$b_2 b_3 b_4 b_5$: column

DES S-Boxes

S_3-box

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

S_4-box

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

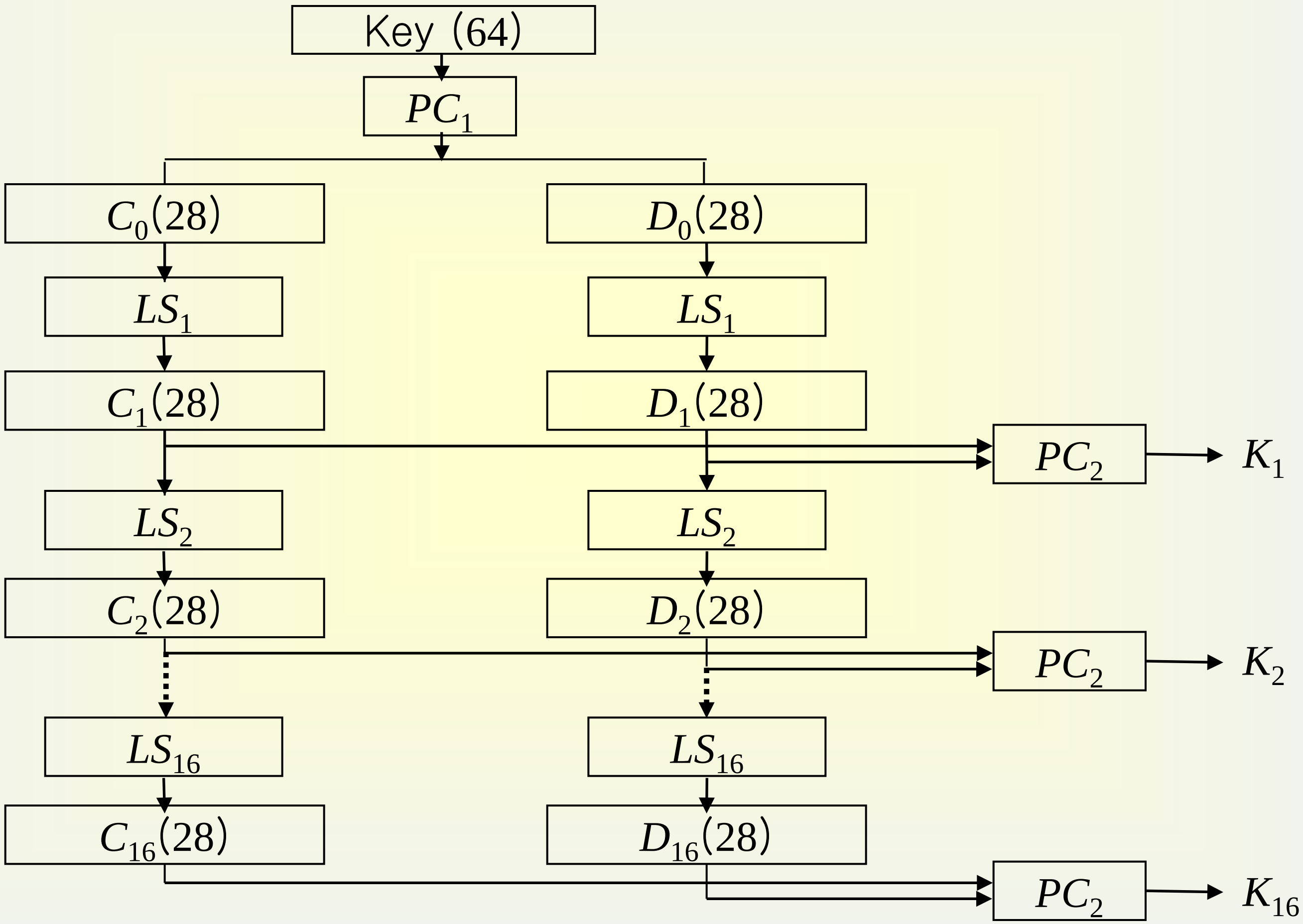
DES S-boxes

- 8 S-boxes (6 → 4 bits)
- each row : permutation of 0-15
- 4 rows : chosen by MSB & LSB of input
- some known design criteria
  - ✓ not linear
  - ✓ Any one bit of the inputs changes at least two output bits
  - ✓ S(x) and S(x ⊕ 001100) differs at least 2bits
  - ✓ S(x) ≠ S(x ⊕ 11*ef*00) for any *ef*
  - ✓ Resistance against DC etc.
  - ✓ The actual design principles have never been revealed (US classified information)

Exercise: For the S_1-box check whether the following property holds

✓ S(x) and S(x ⊕ 001100) differs at least 2bits

Key Scheduling



PC_1

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

64 bit -> 56 bit (Actual key size of DES is 56-bit)
cf.) Do not use the parity check bits.
8 16 24 32 40 48 56 64 was removed

PC_2

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

56 bit -> 48 bit

9, 18, 22, 25, 35, 38, 43, 54 was removed

Left Shift LS_s

Iteration	Shift	Iteration	Shift
LS_1	1	LS_9	1
LS_2	1	LS_{10}	2
LS_3	2	LS_{11}	2
LS_4	2	LS_{12}	2
LS_5	2	LS_{13}	2
LS_6	2	LS_{14}	2
LS_7	2	LS_{15}	2
LS_8	2	LS_{16}	1

Data Encryption Standard (DES)

- **DES - Controversies**
  - ✓ **Unknown design criteria**
  - ✓ **Slow in software**
  - ✓ **Too short key size – 56 bits**

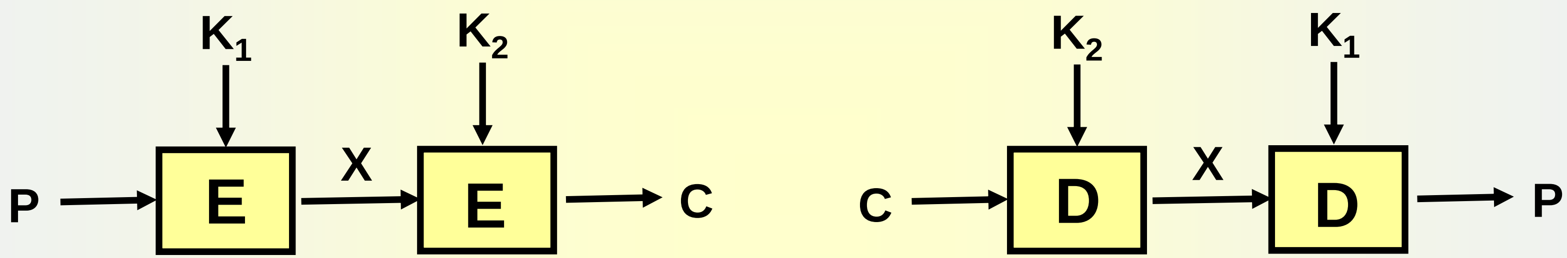
- **DES Crack Machine**
  - ✓ **Can test over 90 billion keys per second**
  - ✓ **EFF's "Deep Crack" and the Distributed.Net computers were testing 245 billion keys per second**
  - ✓ **On Jan. 19, 1999, RSA DES-III Challenge was deciphered after searching 22h. and 15m.**

Identifier: DES-Challenge-III
Cipher: DES
Start: January 18, 1999 9:00 AM PST
Prize: $10,000
IV: da 4b be f1 6b 6e 98 3d
Plaintext: See you in Rome (second AES Conference, March 22-23, 1999)

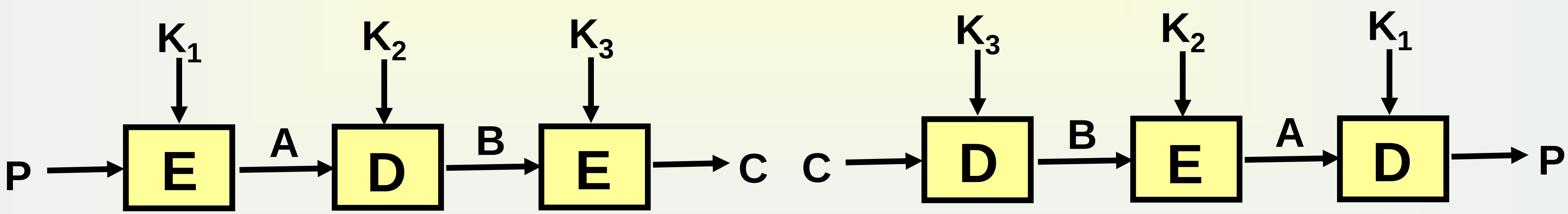
http://www.rsa.com/rsalabs/node.asp?id=2108

Double DES & Triple DES

- How to strengthen existing DES implementations ?
- Double DES
  - Essentially no security increase: $E_{K1}(P) = X = D_{K2}(C)$



- Triple DES
  - Two-key 3DES: $K_1 = K_3$



3. SEED

Korean standard encryption algorithm

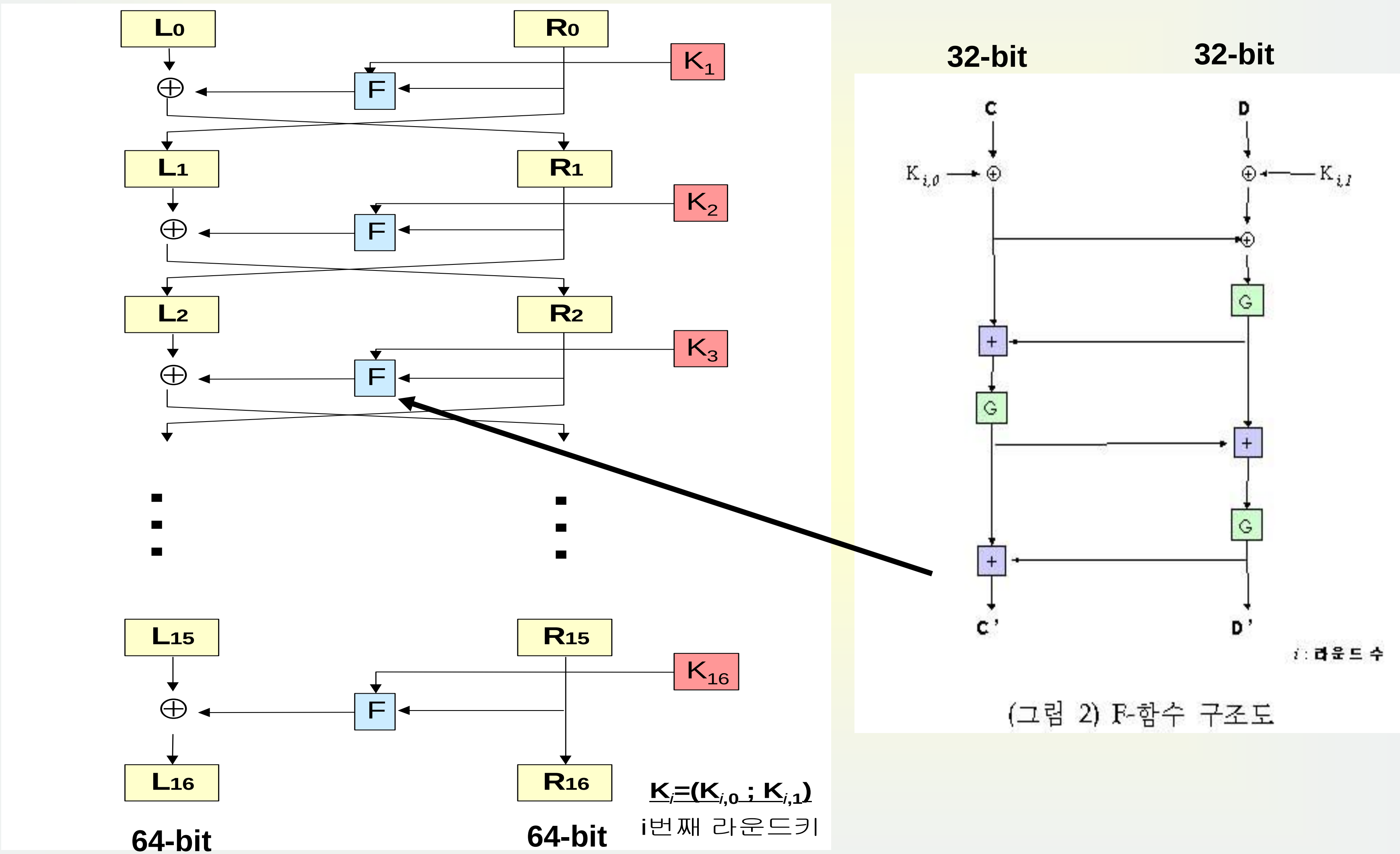
http://www.cyberprivacy.or.kr/kisa/seed/data/Document_pdf/SEED_Specification_english.pdf

SEED Algorithm

Features

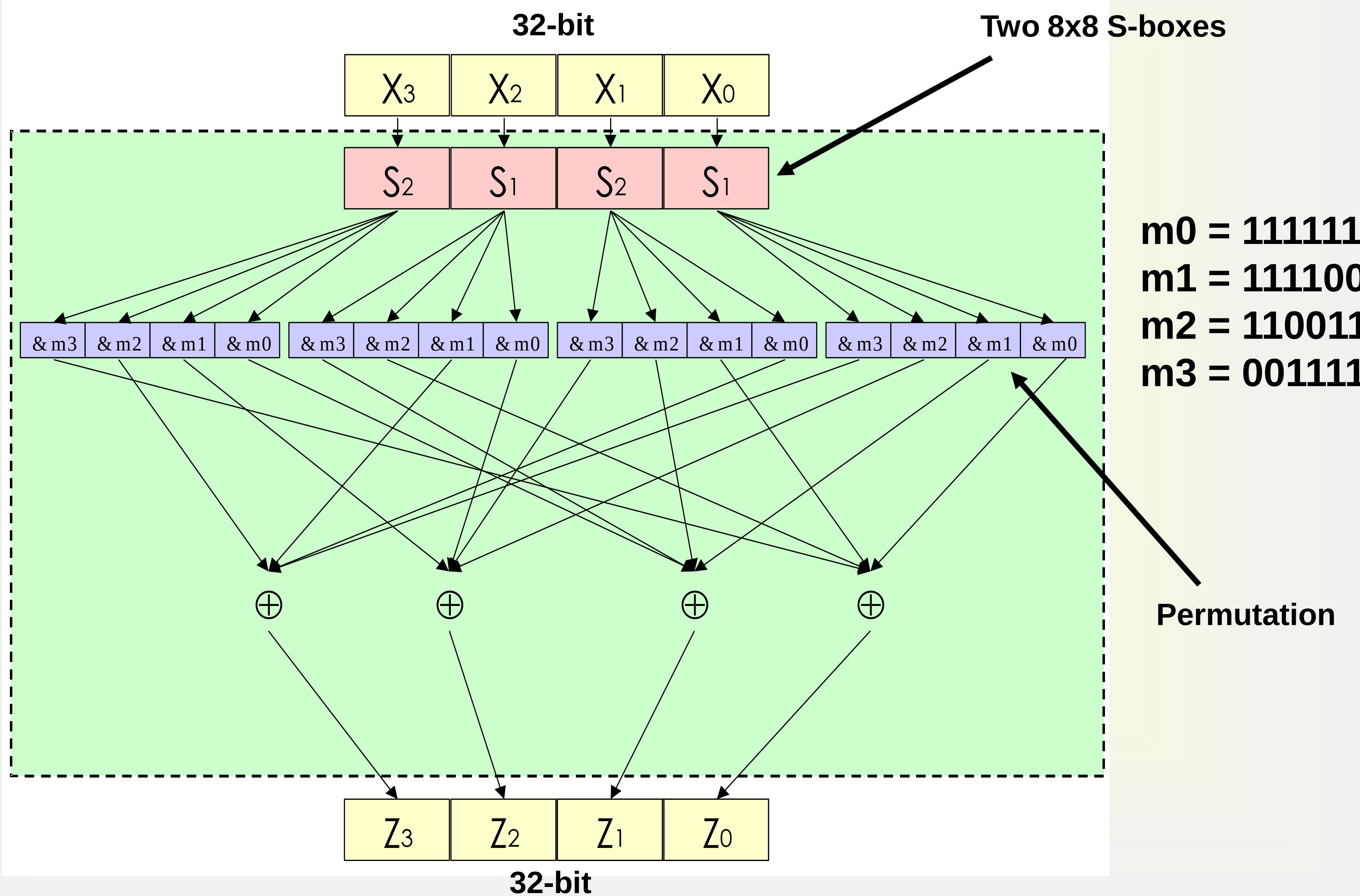
- **Feistel structure with 16 rounds**
- **128-bits input-output data block size**
- **128-bits key size**
- **Two 8x8 S-Boxes**
- **Mixed operation of XOR and modular addition**

SEED Architecture



(그림 2) F-함수 구조도

The G Function



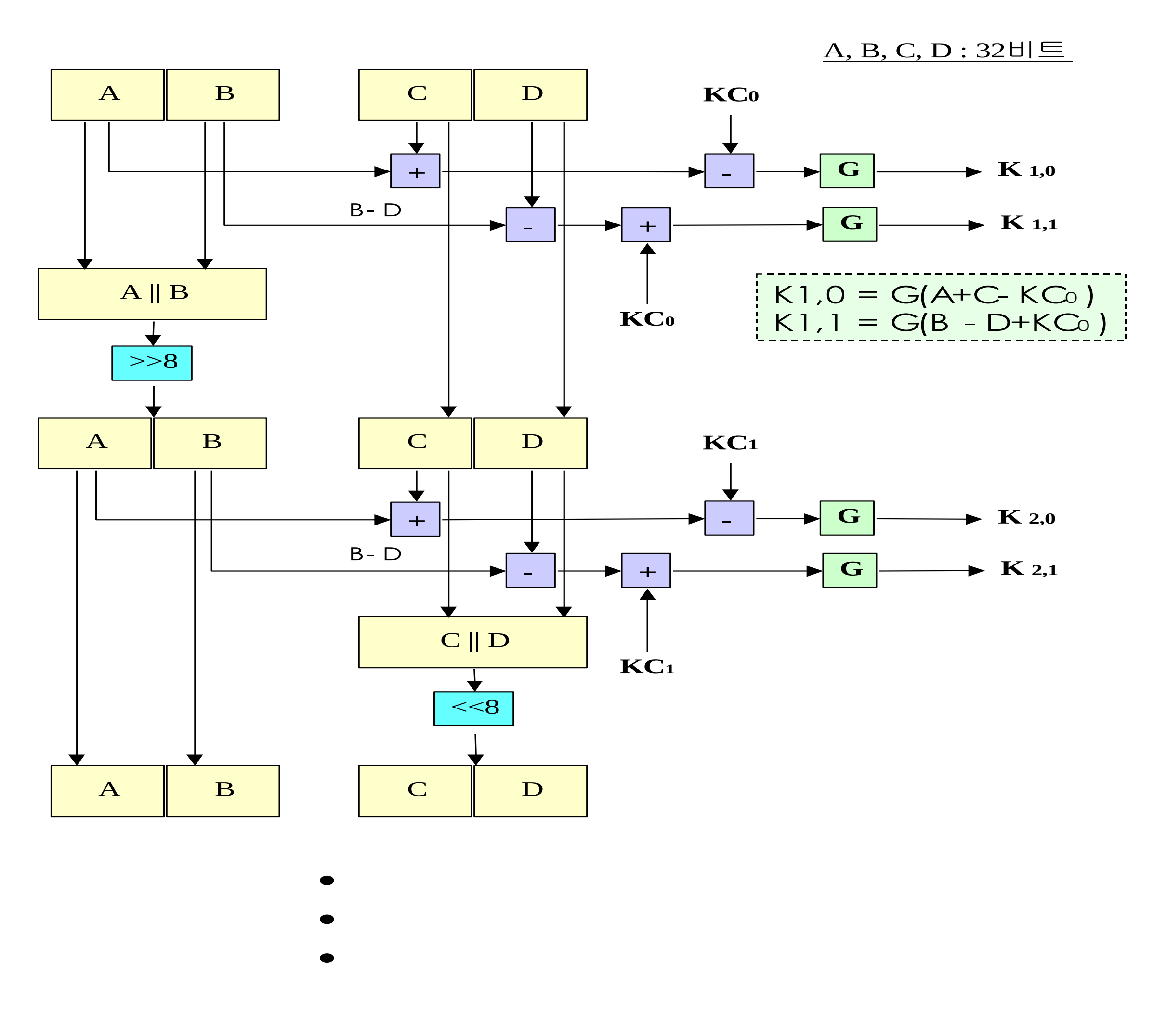
m0 = 11111100
m1 = 11110011
m2 = 11001111
m3 = 00111111

Two 8x8 S-boxes

$$S_i : Z_{2^8} \to Z_{2^8}, S_i(x) = A^{(i)} \bullet x^{n_i} \oplus b_i$$

$$A^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad A^{(2)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

SEED Key Scheduling



4. Advanced Encryption Standard

http://csrc.nist.gov/CryptoToolkit/aes/rijndael/

AES Contest

- **AES Contest Calendar**
  - 1997 : Call For AES Candidate Algorithms by NIST
  - 1998 : 1st Round Candidates – 15 Algorithms
    - Mars, Twofish, RC6, SAFER+, HPC, CAST256, DEAL, Frog, Magenta, Rijndael, DFC, Serpent, Crypton, E2, LOKI97
  - 1999 : 2nd Round Candidates – 5 Algorithms
    - MARS, RC6, Rijndael, Serpent, and Twofish
  - 2000. 10 : Rijndael selected as the finalist
  - 2001. 12: official publication as FIPS PUB 197

* National Institute of Standards and Technology

AES Contest

- 1st Round Candidates – 15 Algorithms

Cipher	Submitted by	Country
CAST-256	Entrust	Canada
Crypton	Future Systems	Korea‡
Deal	Outerbridge	Canada†
DFC	ENS–CNRS	France
E2	NTT	Japan
Frog*	TecApro	Costa Rica
HPC*	Schroeppel	USA
LOKI97*	Brown, Pieprzyk, Seberry	Australia
Magenta	Deutsche Telekom	Germany
Mars	IBM	USA†
RC6	RSA	USA†
Rijndael*	Daemen, Rijmen	Belgium‡
Safer+*	Cylink	USA†
Serpent*	Anderson, Biham, Knudsen	UK, Israel, Norway
Twofish*	Counterpane	USA†

* Placed in the public domain; † and foreign designers; ‡ foreign influence

AES Contest

- 2nd Round Candidates – 5 Algorithms

Cipher	Submitter	Structure	Nonlinear Component
MARS	IBM	Feistel structure	Sbox DD-Rotation
RC6	RSA Lab.	Feistel structure	Rotation
Rijndael	Daemen, Rijmen	SPN structure	Sbox
Serpent	Anderson, Biham, Knudsen	SPN structure	Sbox
Twofish	Schneier et. al	Feistel structure	Sbox

AES Contest

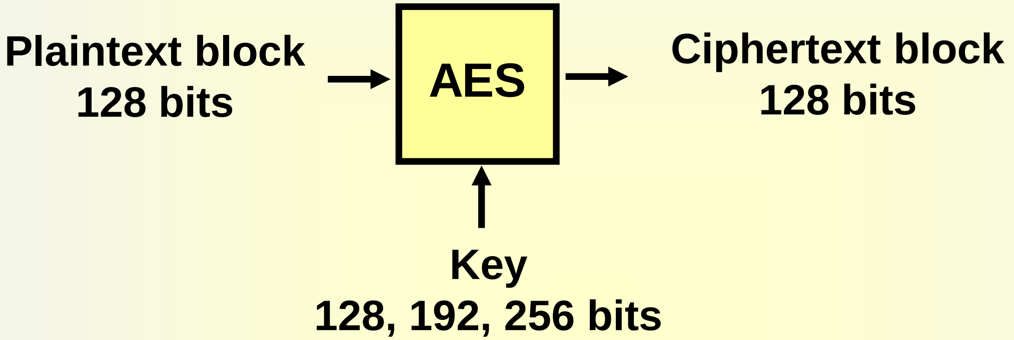
- AES Contest
  - 2000. 10 : Rijndael selected as the finalist
  - 2001. 12: official publication as FIPS PUB 197

Joan Daemen and Vincent Rijmen, " The Design of Rijndael, AES – The Advanced Encryption Standard", Springer, 2002, ISBN 3-540-42580-2

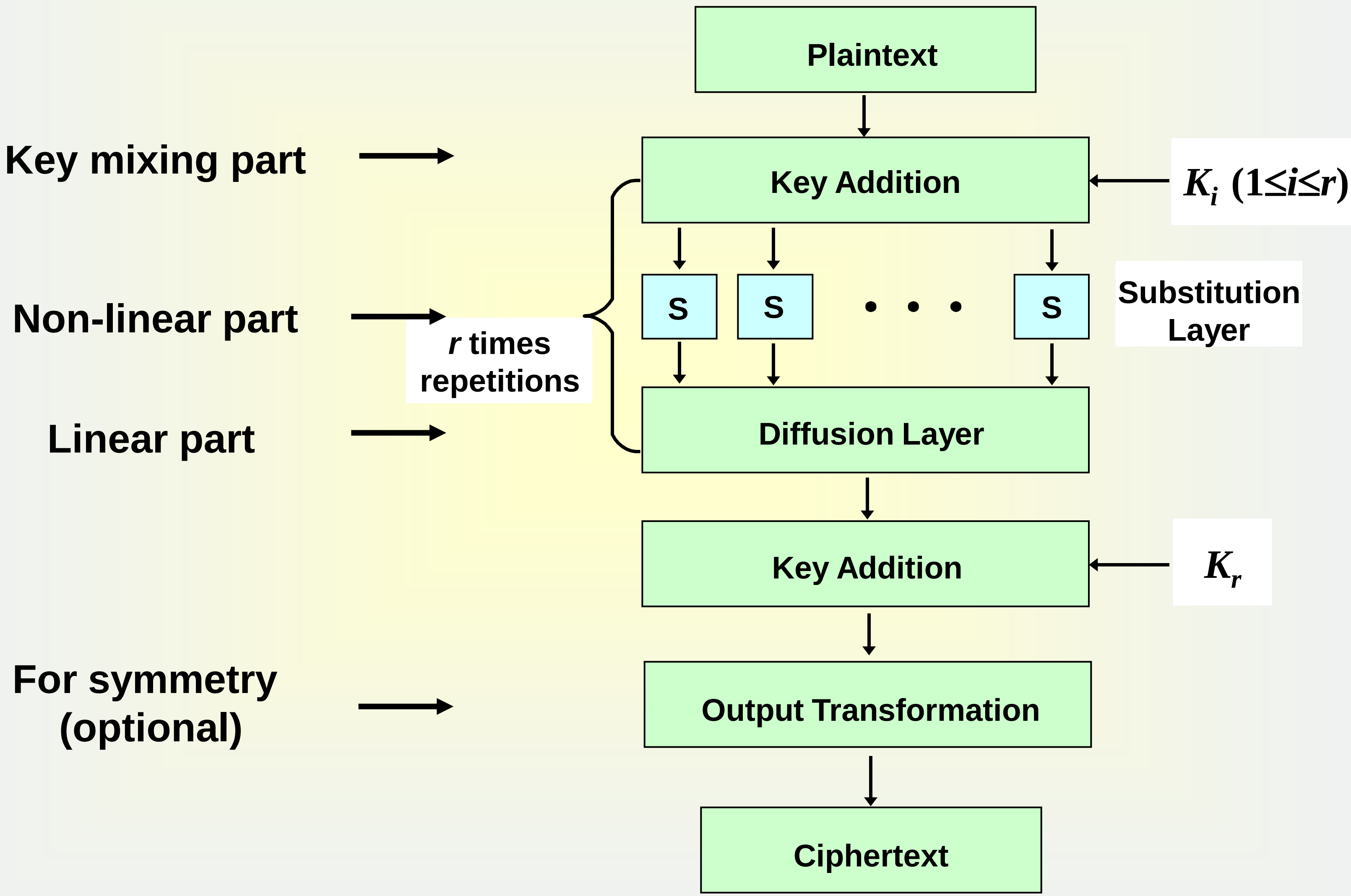
Vincent Rijmen

Advanced Encryption Standard (AES)

- AES External Format



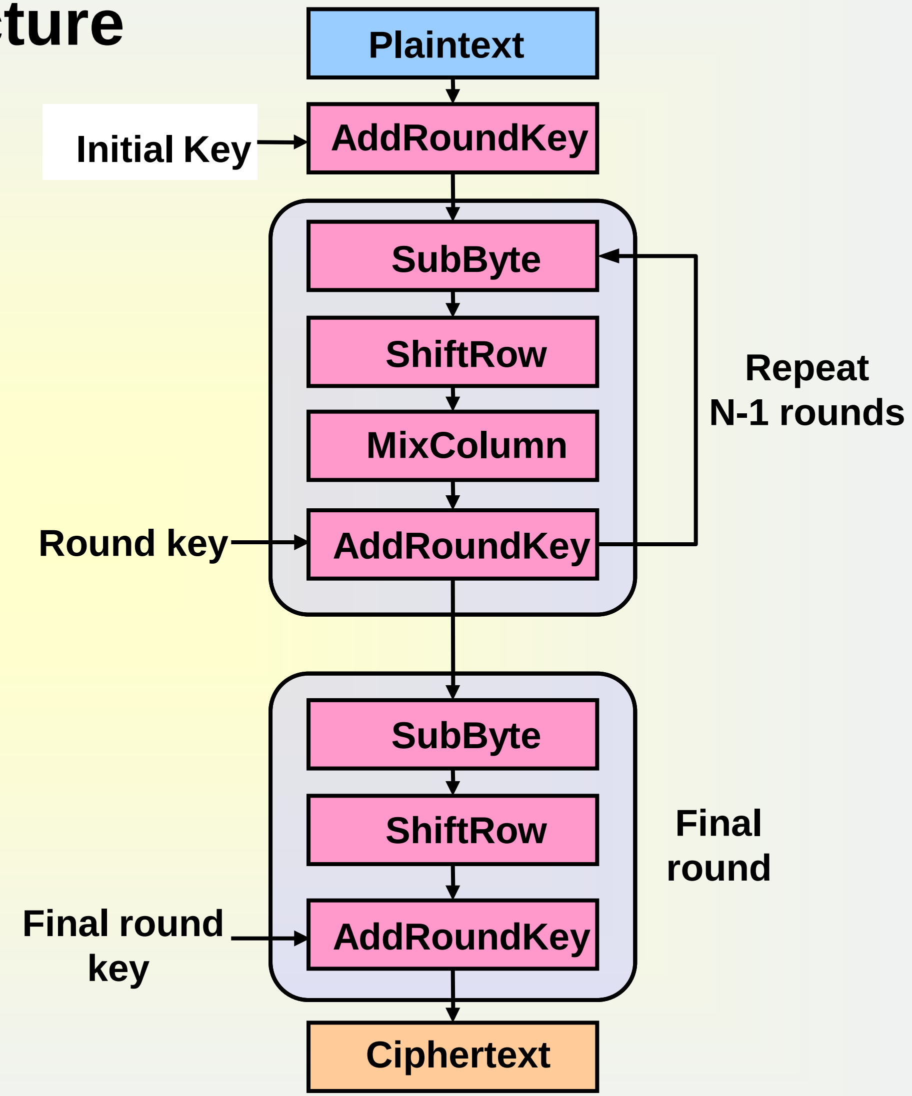
Block Cipher Architecture : SPN-type



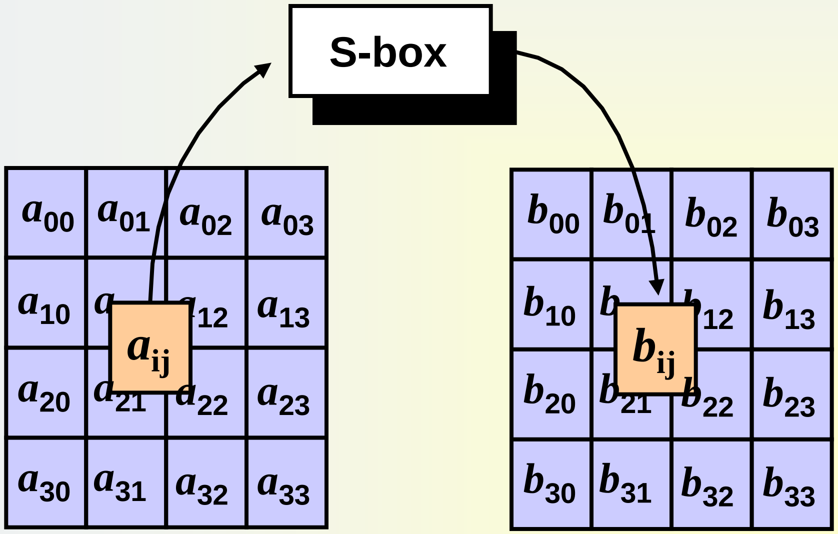
AES Architecture

- SPN-type block cipher
- Block size = 128 bits
- Key size / No. round
  - 128 bits → 10 rounds
  - 192 bits → 12 rounds
  - 256 bits → 14 rounds

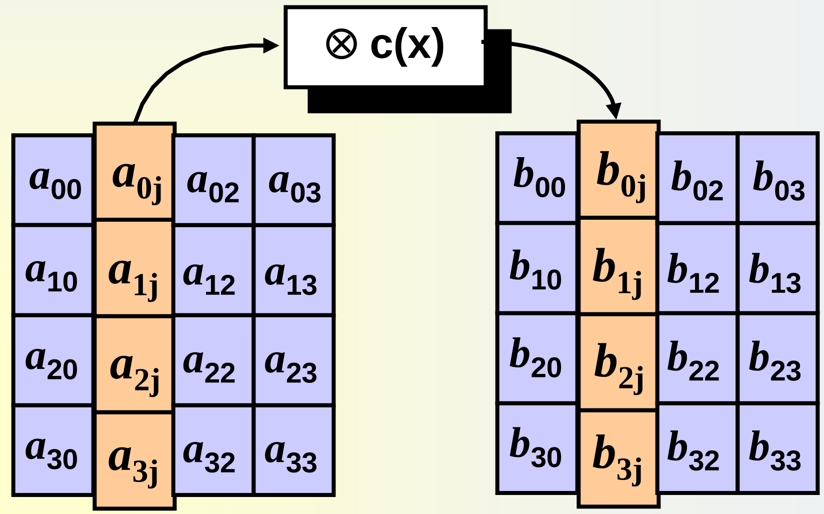
- Round transformation
  - SubBytes
  - ShiftRow
  - MixColumn
  - AddRoundKey



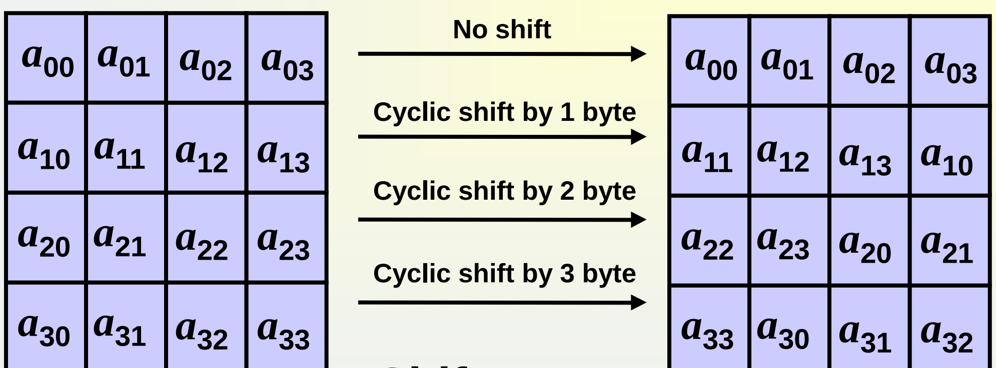
Round Transformation



SubBytes



MixColumns



ShiftRows

The input block is XOR-ed with the round key

AddRoundKey

SubBytes

1. First, taking the multiplicative inverse in $GF(2^8)$, with the representation defined in Section 2.1. '00' is mapped onto itself.
2. Then, applying an affine (over GF(2)) transformation defined by:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

❖ **Inversion in GF(2^8) defined by** $m(x) = x^8 + x^4 + x^3 + x + 1$

S-box for Rijndael

For a 8-bit input *abcdefgh*
look for the entry in the *abcd* row and *efgh* column

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

MixColumns / InvMixColumns

❖ **Constant polynomial multiplication mod x^4+1**

$$c(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$$

$$b_j(x) = c(x) \otimes a_j(x)$$

$$c^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\}$$

$$b_j(x) = c^{-1}(x) \otimes a_j(x)$$

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

MixColumns

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

InvMixColumn

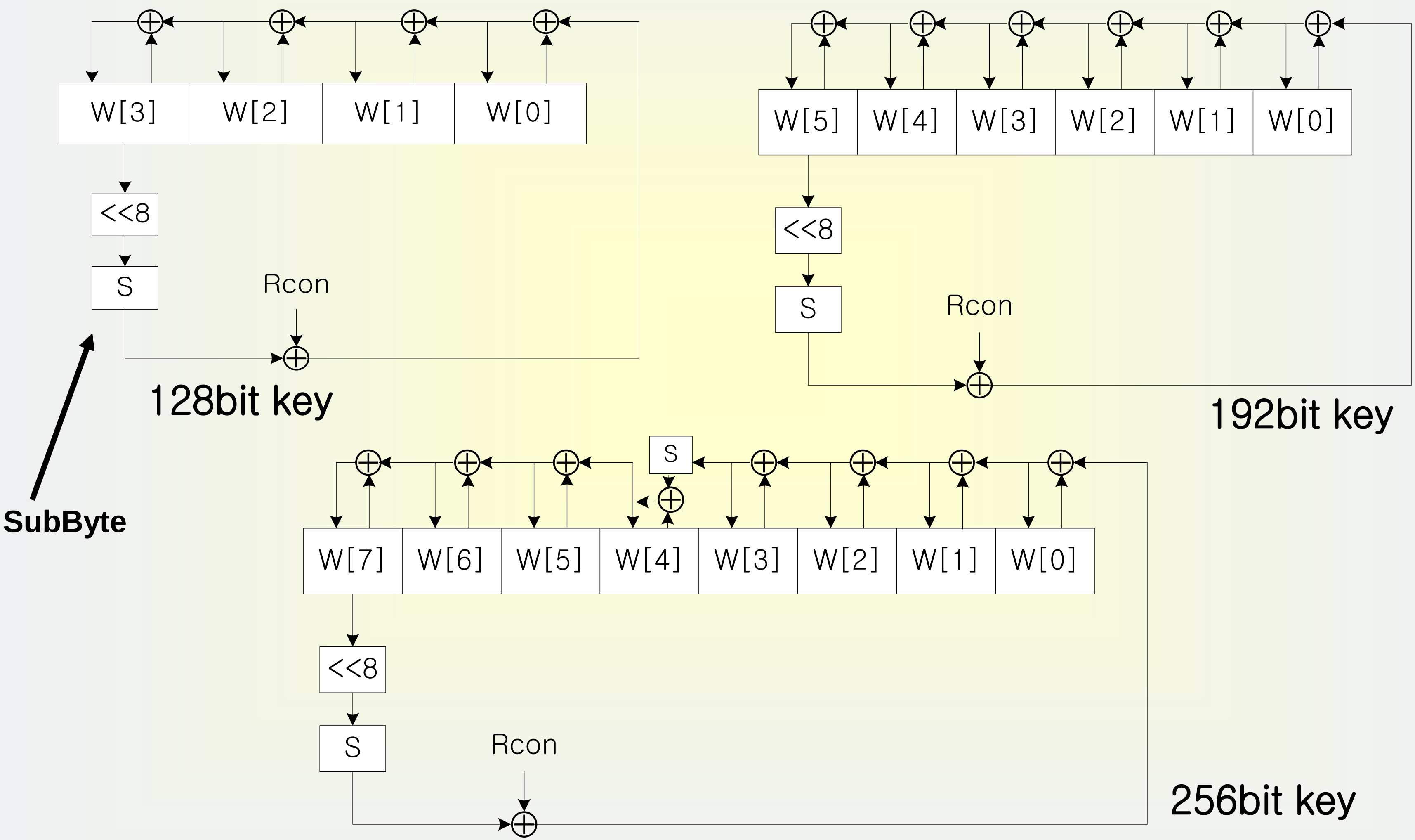
AddRoundKey

- The input block is XOR-ed with the round key

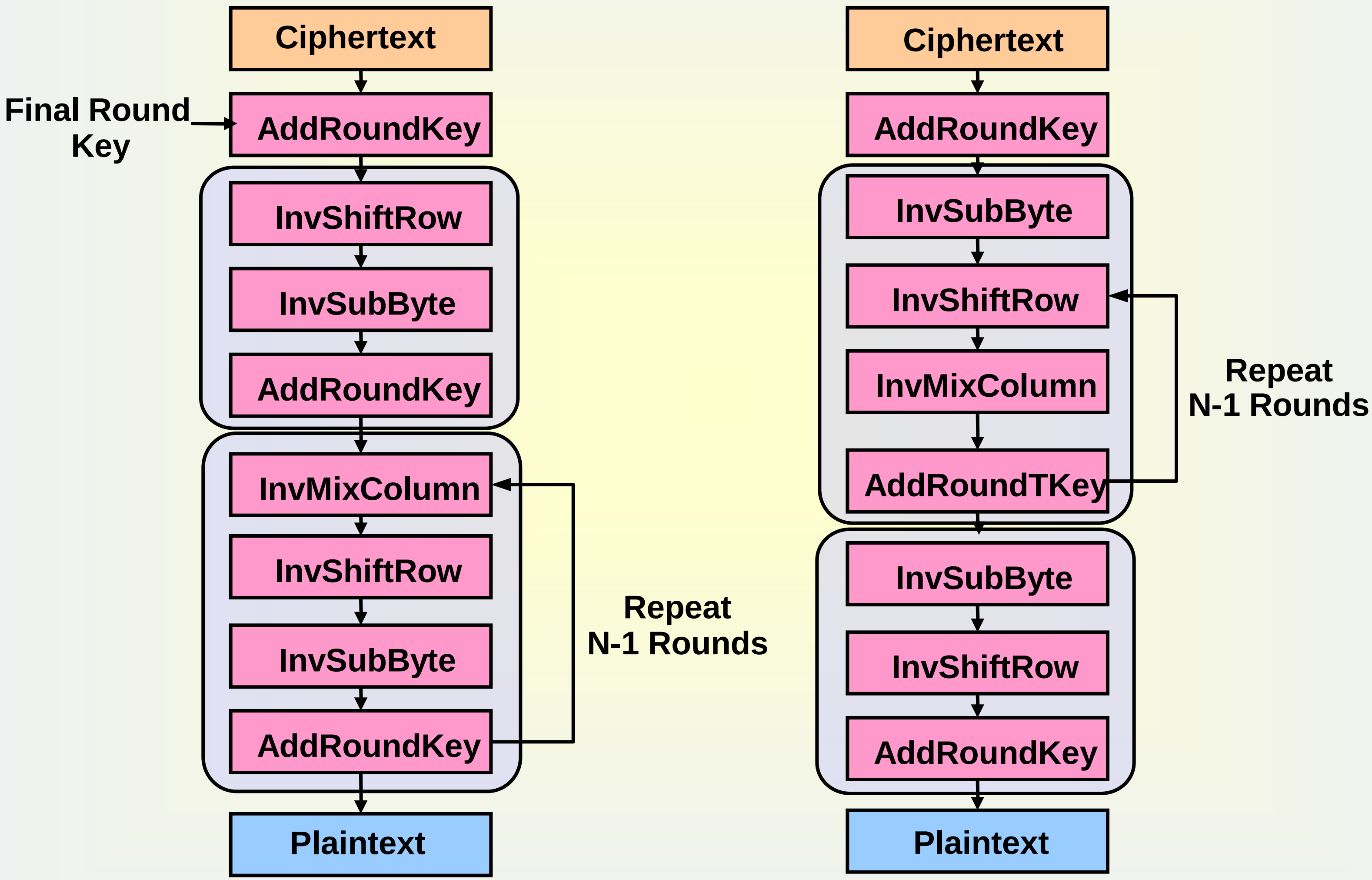
$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \end{bmatrix}$$

Figure 5: In the key addition the Round Key is bitwise EXORed to the State.

Key Scheduling

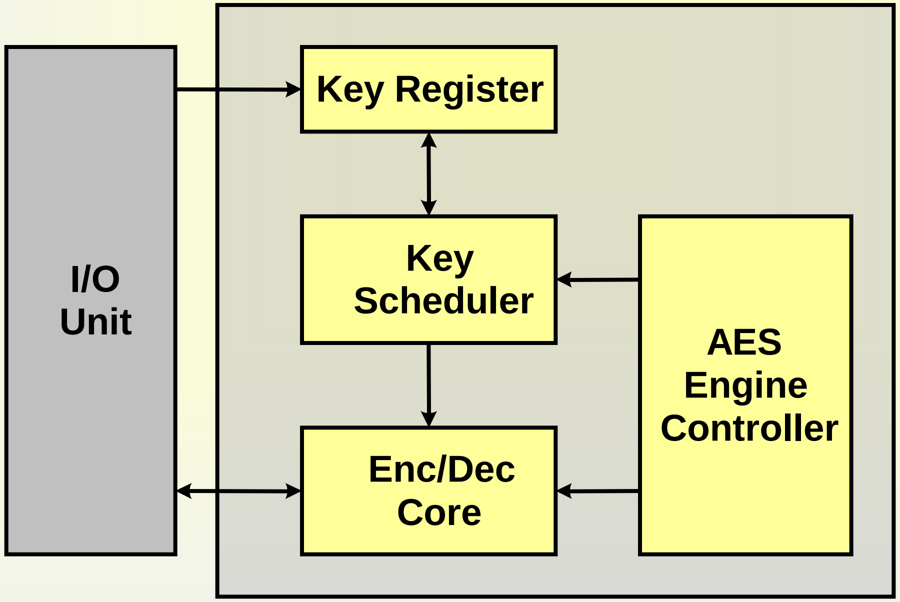


Decryption Architecture



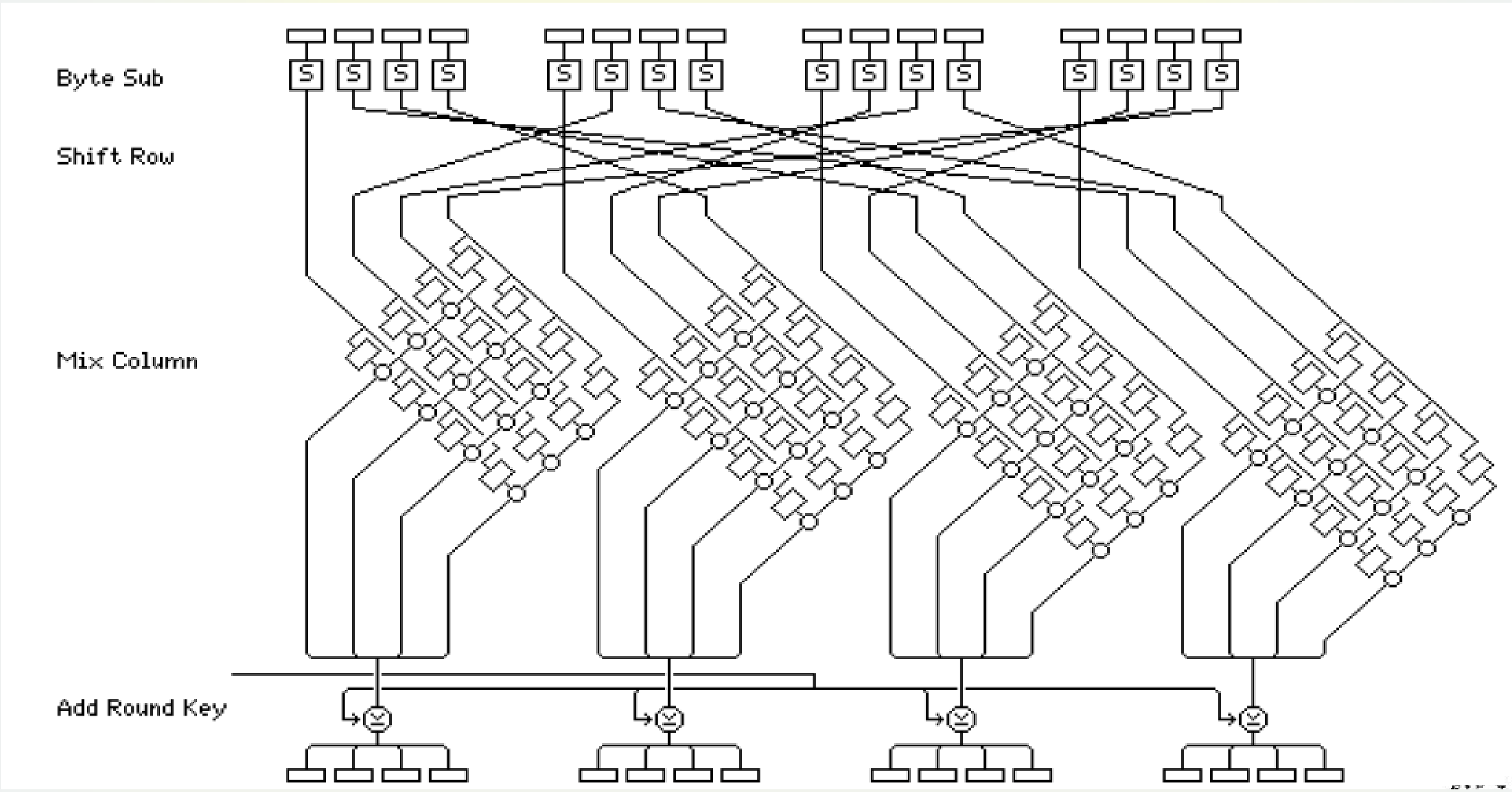
AES Engine – Top-level View

- AES hardware implementation module
  - Control logic
  - Encryption/Decryption Core
  - Key Scheduler



AES Engine

AES Hardware Implementation



AES Hardware Implementation

Table4.1 Hardware evaluation results

Algorithm name	area [Gate]			Key setup time[ns]	Critical-path[ns]	Throughput [Mbps]
	Encryption & Decryption	Key Schedule	Total			
DES	42,204	12,201	54,405	-	55.11	1161.31
Triple-DES	124,888	23,207	148,147	-	157.09	407.4
MARS	690,654	2,245,096	2,935,754	1740.99	567.49	225.55
RC6	741,641	901,382	1,643,037	2112.26	627.57	203.96
Rijndael	518,508	93,708	612,834	57.39	65.64	1950.03
Serpent	298,533	205,096	503,770	114.07	137.4	931.58
Twofish	200,165	231,682	431,857	16.38	324.8	394.08

*** CMOS ASIC Implementation by Ichikawa (Mitsubishi)**

Feistel vs. SPN Structures

Feistel structure

64-bit block	128-bit block
• DES/3DES	• SEED
• BLOWFISH	• TWOFISH
• CAST128	• CAST256
• RC5	• RC6
	• MARS

- Fewer constraints on the round function
- More cryptanalytic experience
- Serial in nature
- Typically E = D with round keys in reverse order

SPN structure

64-bit block	128-bit block
• SAFER	• AES
• SAFER+	• CRYPTON
• IDEA	• SERPENT

- More constraints on the round function: must be invertible
- Less cryptanalytic experience: a little bit new architecture
- more parallelism
- Typically $E \neq D$

Padding for Block Cipher

- Different padding methods according to applications

			To be padded				

- PKCS Padding : general

			05	05	05	05	05

- OneAndZero Padding : hash

			80	00	00	00	00

- Zero Padding : CBC-MAC

			00	00	00	00	00

- No Padding
  : block-size multiple Data

- TLS Padding
  : variable padding length

			04	04	04	04	04
			padding				length

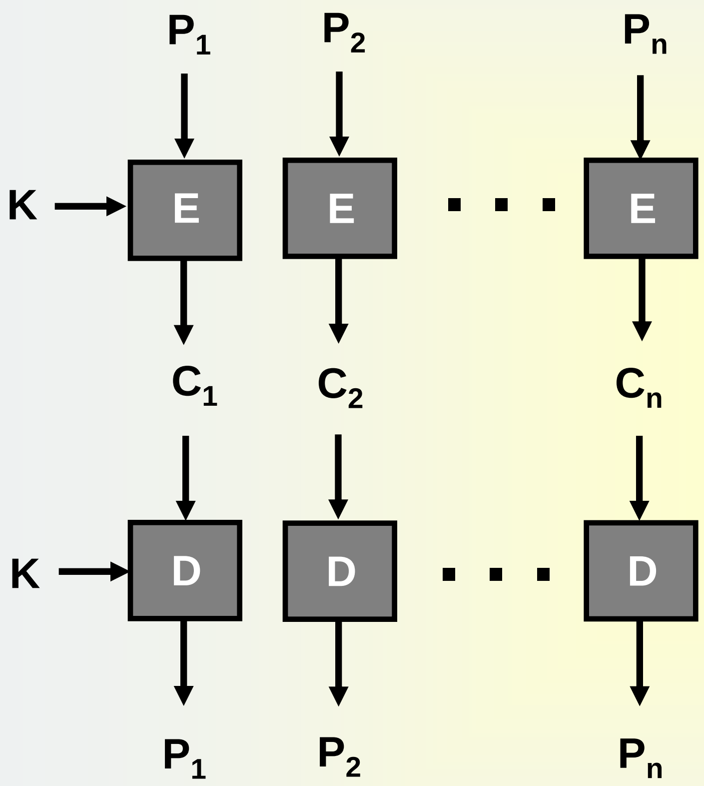
			0C	0C	0C	0C	0C
0C	0C	0C	0C	0C	0C	0C	0C

- SSL Padding

			00	00	00	00	04

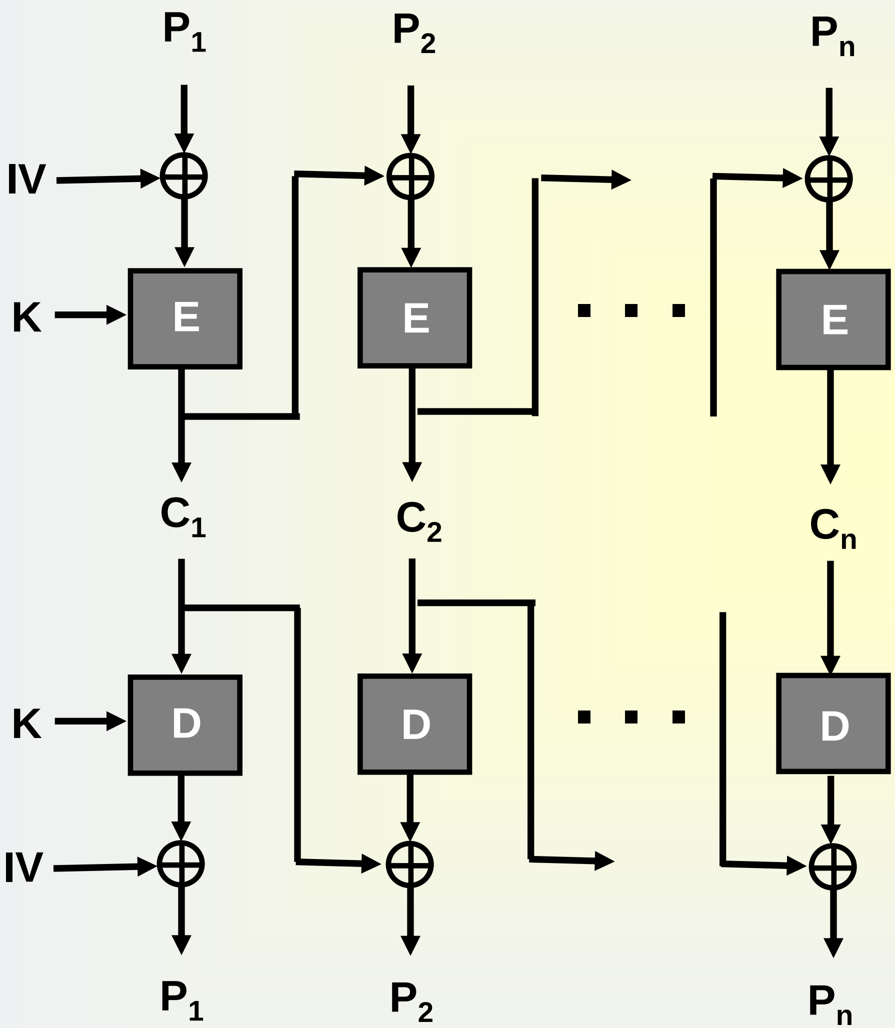
5. Mode of Operation

Modes of Operation – ECB Mode



- **Electronic Code Book Mode**
  - ✓ Break a message into a sequence of plaintext blocks
  - ✓ Each plaintext block is encrypted (or decrypted) independently
  - ✓ The same plaintext block always produces the same ciphertext block
  - ✓ May not be secure; e.g., a highly structured message
  - ✓ Typically used for secure transmission of single vales (e.g., encryption key)

Modes of Operation – CBC Mode



- ***Cipher Block Chaining Mode***
  - ✓ Each ciphertext block is affected by previous blocks
  - ✓ No fixed relationship between the plaintext block and its input to the encryption function
  - ✓ The same plaintext block, if repeated, produces different ciphertext blocks
  - ✓ IV(Initializing Vector) must be known to both ends
  - ✓ Most widely used for block encryption

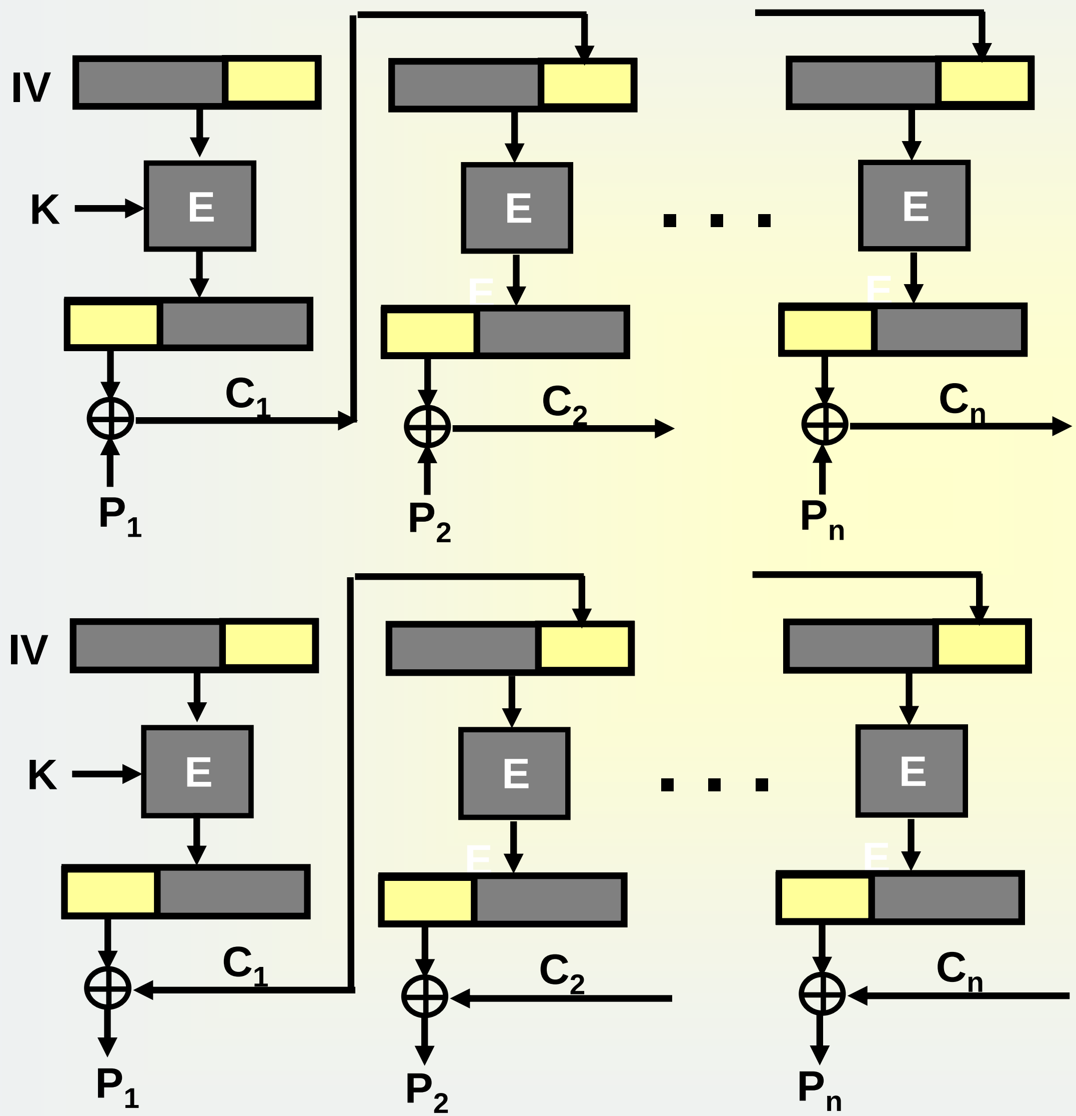
$C_1 = E_K(P_1 \oplus IV)$ $\quad$ $C_3 = E_K(P_3 \oplus C_2)$

$P_1 = IV \oplus D_K(C_1)$ $\quad$ $P_3 = C_2 \oplus D_K(C_3)$

$C_2 = E_K(P_2 \oplus C_1)$ $\quad$ $C_4 = E_K(P_4 \oplus C_3)$

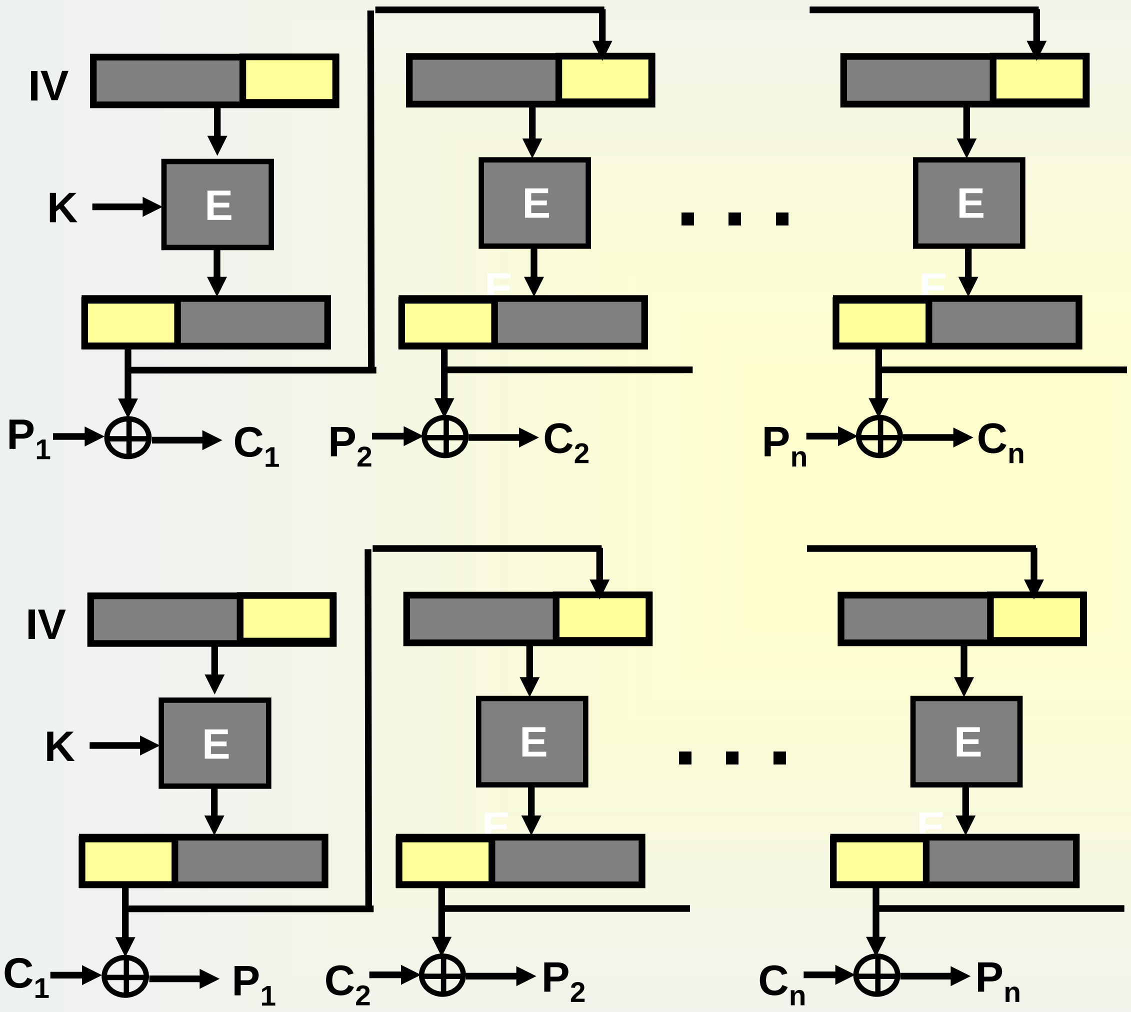
$P_2 = C_1 \oplus D_K(C_2)$ $\quad$ $P_4 = C_3 \oplus D_K(C_4)$

Modes of Operation – CFB Mode



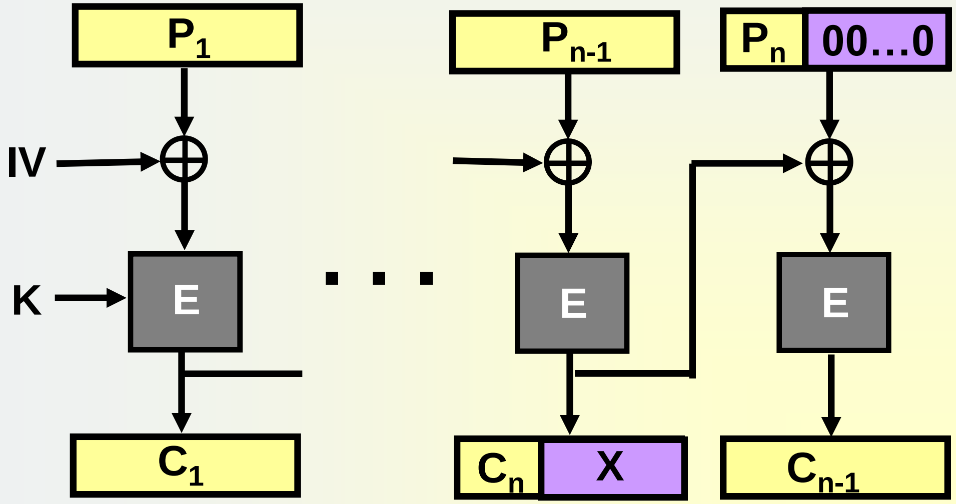
- ***Cipher Feedback Mode***
  - ✓ A way of using a block cipher as a stream cipher
  - ✓ A shift register of block size maintains the current state of the cipher operation, initially set to some IV
  - ✓ The value of the shift register is encrypted using key K and the leftmost j bits of the output is XORed with j-bit plaintext P_i to produce j-bit ciphertext C_i
  - ✓ The value of the shift register is shifted left by j bits and the C_i is fed back to the rightmost j bits of the shift register
  - ✓ Typically j = 8, 16, 32, 64 …
  - ✓ Decryption function D_K is never used

Modes of Operation – OFB Mode

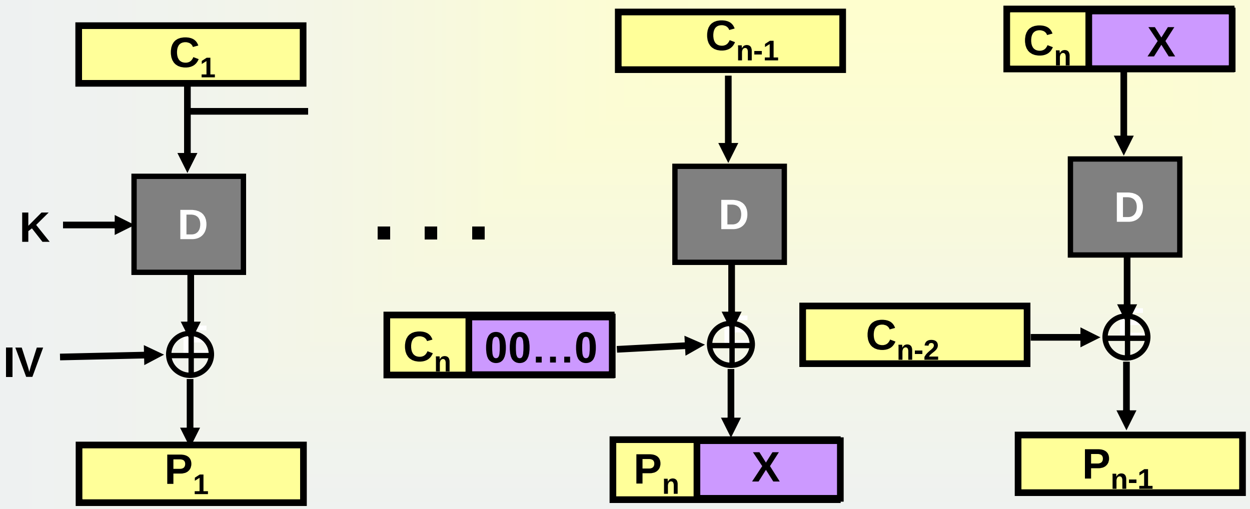


- ***Output Feedback Mode***
  - ✓ The structure is similar to that of CFB, but
    - CFB: Ciphertext is fed back to the shift register
    - OFB: Output of E is fed back to the shift register
  - ✓ For security reason, only the full feedback (j = block size) mode is used
  - ✓ No error propagation
  - ✓ More vulnerable to a message stream modification attack
  - ✓ May useful for secure transmission over noisy channel (e.g., satellite communication)

Modes of Operation – CTS Mode



- *Ciphertext Stealing Mode*
  - ✓ Eliminates the padding requirement for block ciphers
  - ✓ The same as CBC mode, except for the encryption/decryption of the the last two blocks (one complete block and the remaining partial block)
  - ✓ Adopted in H.235 as one of operating modes for block ciphers



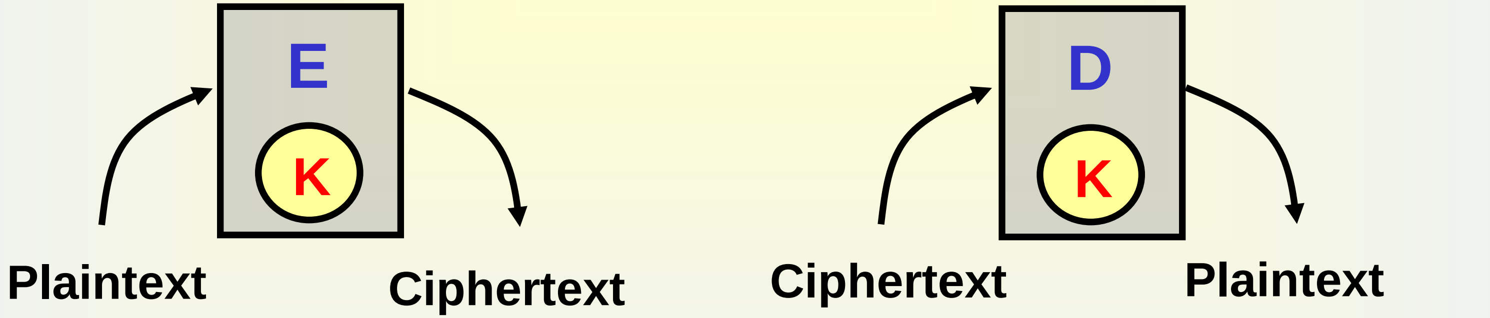
* H.235 covers security and encryption for H.323 and other H.245 based terminals.

* H.323 covers multimedia communication on any packet network

6. Cryptanalysis

Block Cipher – Attack Scenarios

- Attacks on encryption schemes
  - Ciphertext only attack: only ciphertexts are given
  - Known plaintext attack: (plaintext, ciphertext) pairs are given
  - Chosen plaintext attack: (chosen plaintext, corresponding ciphertext) pairs
  - Adaptively chosen plaintext attack
  - Chosen ciphertext attack: (chosen ciphertext, corresponding plaintext) pairs
  - Adaptively chosen ciphertext attack



Encryption Oracle

Decryption Oracle

Cryptanalysis of Block Ciphers

- **Statistical Cryptanalysis**
  - **Differential cryptanalysis (DC)**
  - **Linear Cryptanalysis (LC)**
  - **Various key schedule cryptanalysis**

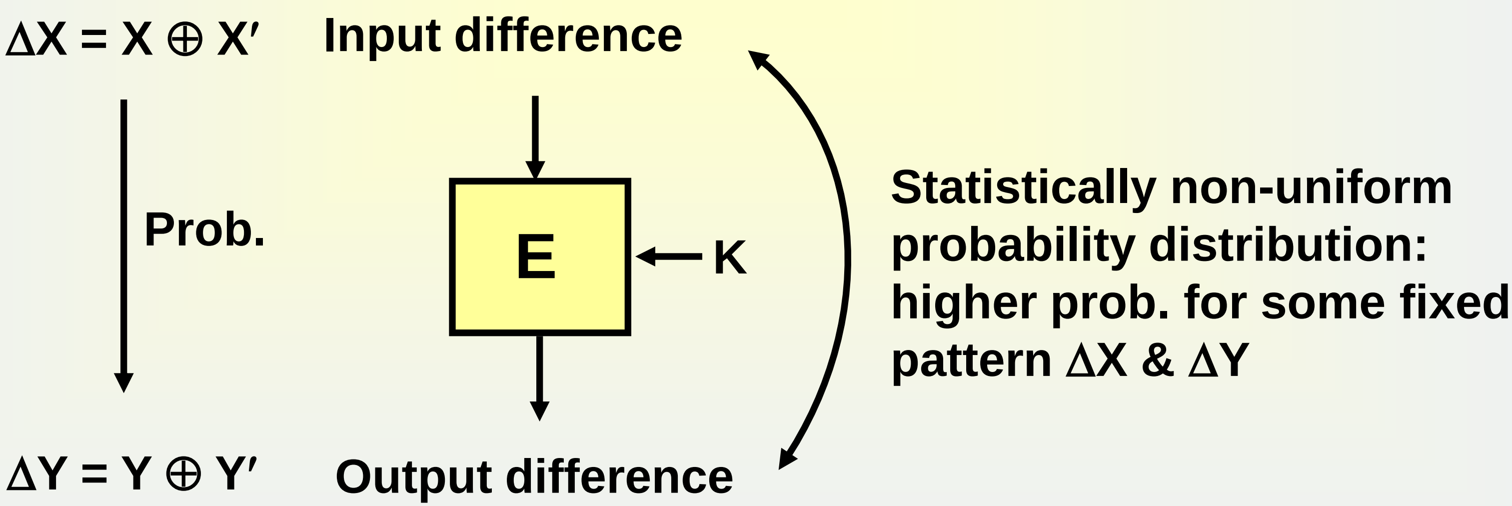
- **Algebraic Cryptanalysis**
  - **Interpolation attacks**

- **Side Channel Cryptanalysis**
  - **timing attacks**
  - **differential fault analysis**
  - **differential power analysis, etc.**

Cryptanalysis of Block Ciphers - DC

- **Differential Cryptanalysis**
  - ✓ **E. Biham and A. Shamir : Crypto90, Crypto92**
  - ✓ **Chosen plaintext attack, O(Breaking DES_{16} ~ 2^{47})**
  - ✓ **Look for correlations in Round function input and output (DES : 2^{47})**
    - **high-probability differentials, impossible differentials**
    - **truncated differentials, higher-order differentials**

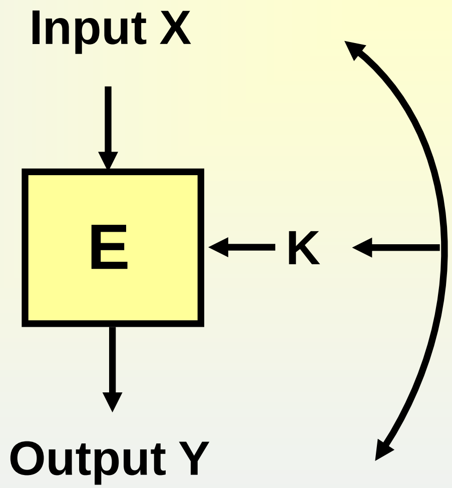
* E.Biham, A. Shamir,"Differential Cryptanalysis of the Data Encryption Standard", Springer-Verlag, 1993



Cryptanalysis of Block Ciphers - LC

- **Linear Cryptanalysis**
  - ✓ **Matsui : Eurocrypt93, Crypto94**
  - ✓ **Known Plaintext Attack, O(Breaking DES_{16}) ~ 2^{43}**
  - ✓ **Look for correlations between key and cipher input and output**
    - **linear approximation, non-linear approximation,**
    - **generalized I/O sums, partitioning cryptanalysis**

* M. Matsui, "Linear Cryptanalysis Method for DES Cipher", Proc. of Eurocrypt'93,LNCS765, pp.386-397



Linear equation between some bits of X, Y and K may hold with higher prob. than others

Other Attacks on Block Ciphers

- **Algebraic Cryptanalysis**
  - ✓ deterministic/probabilistic interpolation attacks

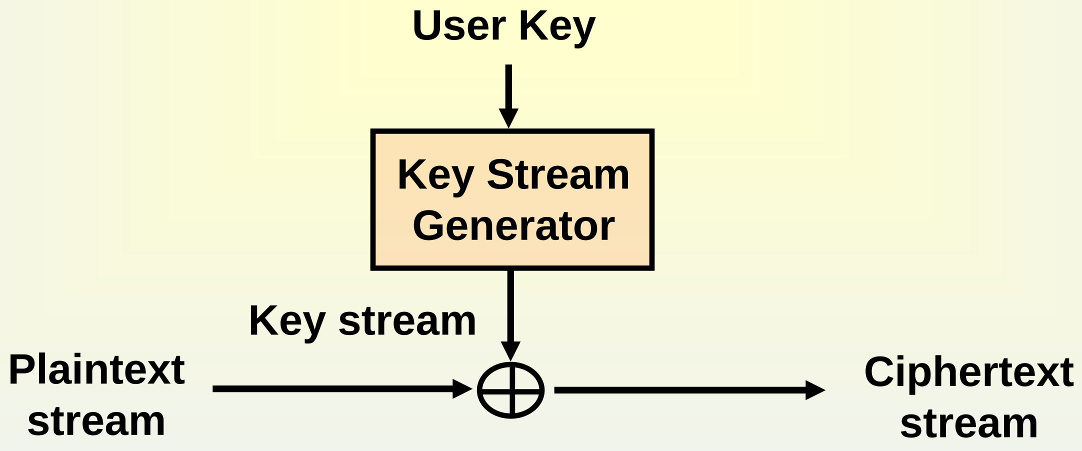
- **Key Schedule Cryptanalysis**
  - ✓ Look for correlations between key changes & cipher input/output
    - equivalent keys, weak or semi-weak keys
    - related key attacks

- **Side-Channel Cryptanalysis**
  - ✓ timing attacks
  - ✓ differential fault analysis
  - ✓ differential power analysis, etc.

7. Stream Ciphers

Stream Cipher

- Plaintext stream is bit-by-bit XORed with key stream to generate ciphertext stream
- The encryption function may vary as plaintext is processed; stateful
- Encryption depends not only on the key and plaintext, but also on the current state
- Advantages
  - No need for padding: the length of ciphertext = the length of plaintext
  - Real-time transmission: encrypt and transmit character by character



Binary Additive Stream Cipher

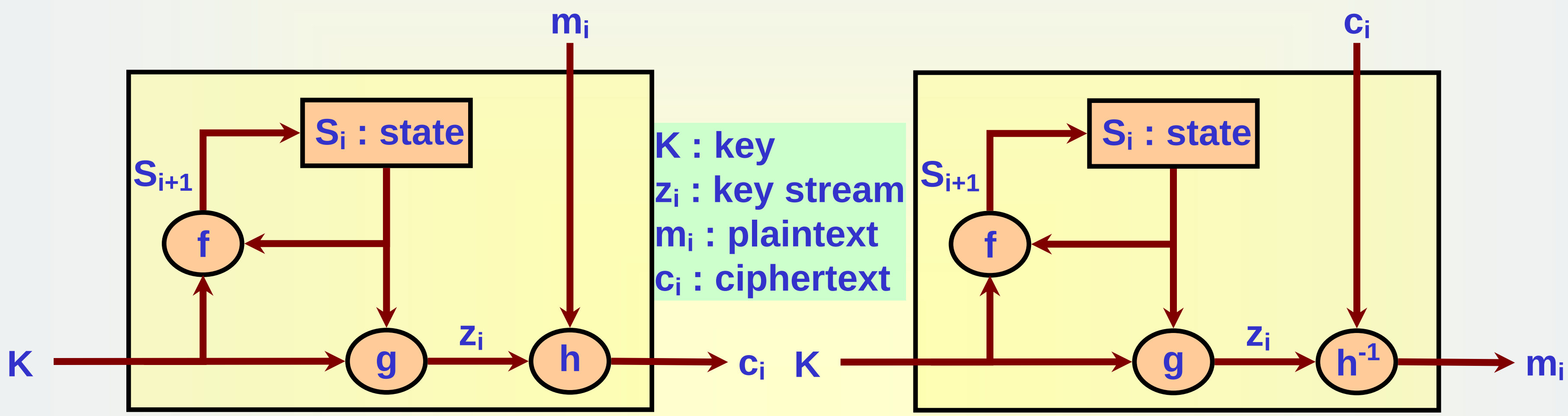
Stream Cipher - Example

- RC4
- SEAL

Plaintext		0	1	1	0	1	0	0	1	1	1
Key Stream	$\oplus$	1	1	0	0	1	0	0	0	1	0
Ciphertext		1	0	1	0	0	0	0	1	0	1
Key Stream	$\oplus$	1	1	0	0	1	0	0	0	1	0
Plaintext		0	1	1	0	1	0	0	1	1	1

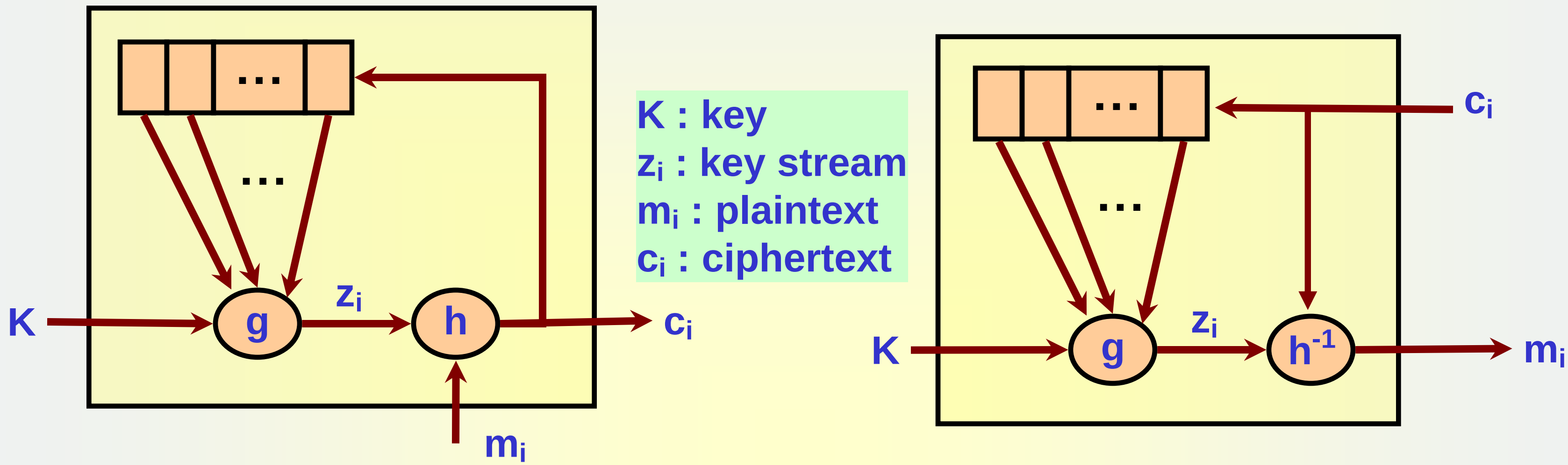
- Extremely fast Encryption/Decryption; Easy to implement in HW
  BUT
- Vulnerable to traffic analysis (|Plaintext| = |Ciphertext|)
- Vulnerable to various attacks without integrity check
- Using two Ciphertexts from the same key stream,
  we can recover the XOR of the Plaintexts
  $\Rightarrow$ **NEVER reuse a key stream !!**

General Model of Stream Ciphers : Synchronous



- The key stream is generated independently of the plaintext & of the ciphertext
- Properties
  - Synchronization requirements
  - No error propagation
- Active attacks
  - Insertion, deletion or replay of ciphertext digits – immediate loss of sync
  - Selective change of ciphertext digits
- Additional mechanisms must be used to provide message origin authentication and message integrity

General Model of Stream Cipher : Self-synchronizing



- The key stream is generated as a function of the key & a fixed number of previous ciphertext digits
- Properties
  - Self-synchronization
  - Limited error propagation
  - Diffusion of plaintext statistics
- Active attacks
  - Insertion, deletion or replay of ciphertext digits – more difficult to detect
  - Modification of ciphertext digits – more likelihood of being detected
- Additional mechanisms must be used for message origin authentication and integrity

Stream Cipher vs. Block Cipher

- **Stream Cipher**
  - ✓ Encrypt individual character (often 1 bit)
  - ✓ Have memory; stateful cipher
  - ✓ Generally extremely faster than block ciphers
  - ✓ Suitable for multimedia streaming data (audio, video)
  - ✓ Limited / No error propagation
  - ✓ Problem : Re-sync. of key stream generator state
  - ✓ Problem : insertion/deletion, replay of ciphertext digits
    - → Need additional Integrity Check
- **Block Cipher**
  - ✓ Encrypt simultaneously a group of characters (64 / 128 bits)
    - → Need Padding
  - ✓ Memoryless
  - ✓ Substitution Permutation Networks(SPN) or Feistel-type
  - ✓ Various modes of operation : ECB, CBC, OFB, CFB, CTS, etc…

Symmetric Key Ciphers - Implementation

➢ **Block Ciphers**

		Block Size	Key Size	Key Scheduling	Enc./Dec. (Mbps)
Feistel	DES	64	56	3.5/3.5 μs	73.8/74.1
	3DES	64	112/168	10.4/10.3 μs	25.1/25.2
	RC5	64	0~2048(128)	4.4/4.4 μs	173.5/191.5
	SEED	128	128	1.3/1.3 μs	88.2/88.8
SPN	AES	128	128/192/256	3.3/3.3 μs	130.0/135.0
	IDEA	64	128	1.5/13.3 μs	57.6/57.2
	Crypton	128	0~256(128)	0.9/1.0 μs	130.9/130.0

➢ **Stream Ciphers**

	Key Setup	Encryption (1 MB data)
RC4	7.9 μs	189.5 Mbps
SEAL	697.5 μs	327.9 Mbps

PIII 450MHz
Widows 98
MSVC++ 6.0

Homework #3

- **Block Cipher Design and Implementation**

The biggest criticism on DES is that its key length (56-bit) is too short compared with current computing environment. Design your own block cipher algorithm which has 256-bit block size and 256-bit key length.
In this homework I do not require any cryptanalysis of your proposed block cipher algorithm. You can study and modify any existing block cipher algorithm (like DES) and their source code freely, but never copy directly. (2DES, 3DES, 4DES are not considered a new algorithm.)
If you think you are not good at programming, you can make a temporary team of 2-3 members and do this homework.

1. **Describe your design strategy or policy**
2. **Describe your algorithm clearly using the techniques learned in this lecture**
3. **Implement a C program (or use any language you prefer) which can encrypt and decrypt message in your algorithm**
4. **Provide a performance analysis of your algorithm and your implementation**