

EE 4215

Security Technology

Lecture 4: Secret-key Cryptosystems

2018/19 Semester B

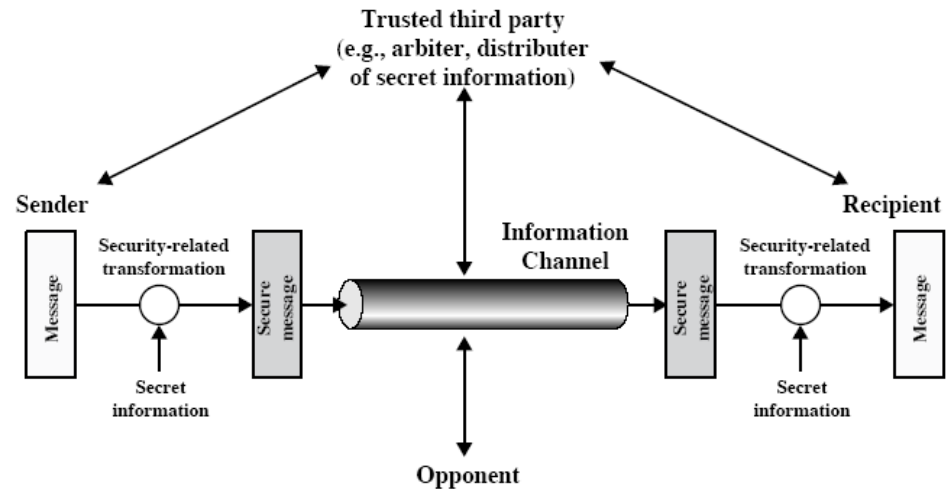
Course Structure

- Week 1: Security Fundamental
- Week 2: Classical Cryptography
- Week 3: Number Theory & DES
- **Week 4: Secret-key Cryptography**
- Week 5: Public-key Cryptography
- Week 5: Quiz 1
- Week 6: Hashing Algorithm
- Week 7: PKI, SSL/TLS
- Week 8: Web Security
- Week 9: Wi-Fi & P2P Security
- Week 10: Cloud Computing Security
- Week 10: Quiz 2
- Week 11: Smart-card Security
- Week 12: Blockchain Security Technology
- Week 13: Secure-code, Trusted Hardware, Review

Lecture 4: Symmetric Key Ciphers

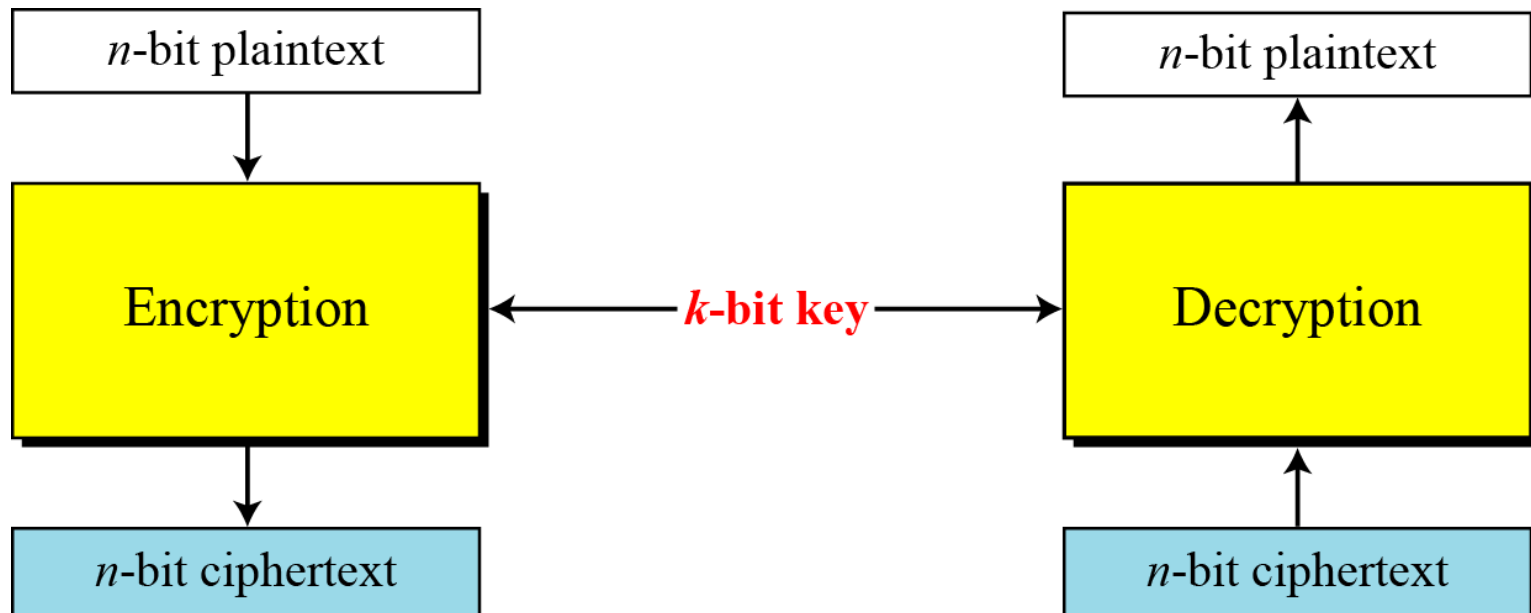
- DES
- AES
- Stream Cipher
- Mode of Operations
- Considerations

Recap



- Sender and recipient share a secret – the key which is used to encrypt & decrypt the messages [hence the name symmetric key ciphers]
- Algorithm assumed to be known
- Strength depends on the secrecy of the key
- Techniques
 - Use a number of iterations of operations including substitution and permutation
 - **diffusion** – spread statistics of **plaintext** into many bits of **ciphertext** to hide their relationship
 - **confusion** – use complex substitution to hide relation between **key** and **ciphertext**

Modern Block Cipher



Note: padding bits are required to add to the message so as to make the length of the plaintext to be n -bit exactly.

History of DES

- In 1973, NIST published a request for proposals for a national symmetric-key cryptosystem in US
- The DES algorithm was designed by Horst Feistel at IBM in 1972
 - original design: 64 bit block size, 64 bit key
 - NSA worked with IBM to tune the algorithm:
 - reduced the key size to 56 bits.
 - changed the “S-Boxes” design
 - approved by the National Bureau of Standards (now NIST) and became a Federal standard (FIPS 46) in 1977
 - renewed as a standard in 1983, 1988, 1993 and 1999. FIPS 46-3 was withdrawn in 2005

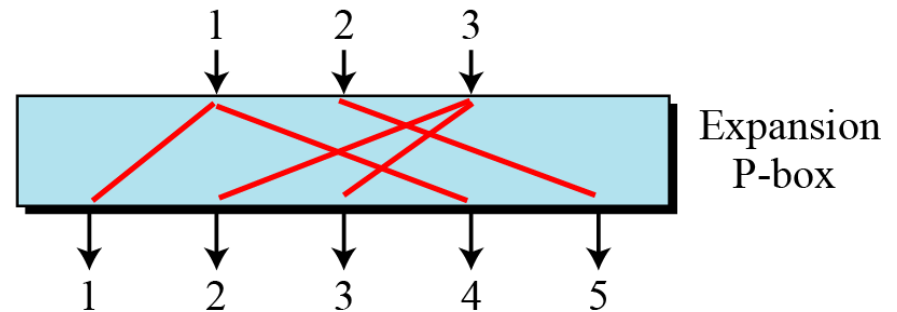
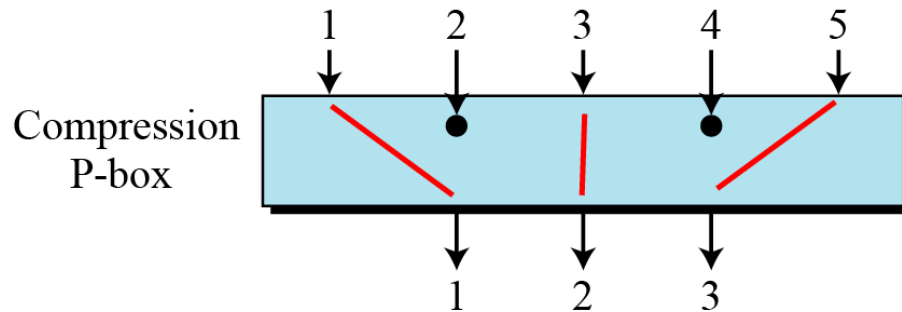
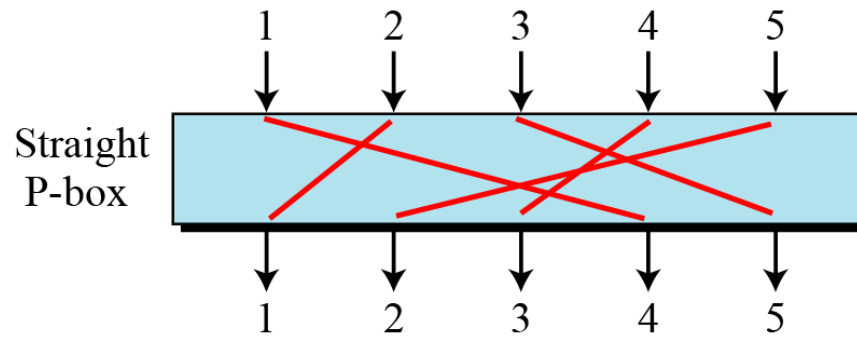
P-Box

- Modern block ciphers normally are keyed substitution ciphers
- P-box (permutation box) parallels the traditional transposition cipher for characters
 - it transposes bits
- Three types of P-boxes
 - straight : invertible
 - compression : not invertible
 - expansion : not invertible

Major Components of Modern Block Cipher

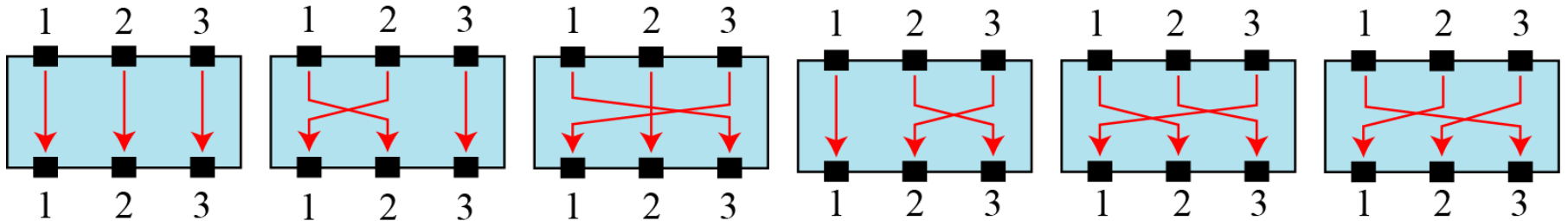
- P-Box
- S-Box
- Exclusive OR (or XOR)
- Circular Shift

P-Boxes



Example

- All 6 possible mappings of a 3×3 P-box



- What about a 64×64 P-box?

S-Box

- An S-box is an $m \times n$ substitution unit
 - m and n are not necessarily the same
- Example: 3x2 S-box
 - Given the S-box:

Leftmost bit	00	01	10	11	Rightmost bits
0	00	10	01	11	
1	10	00	11	01	

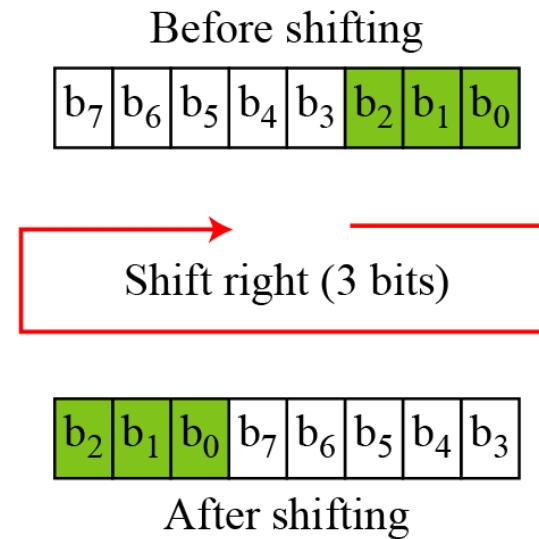
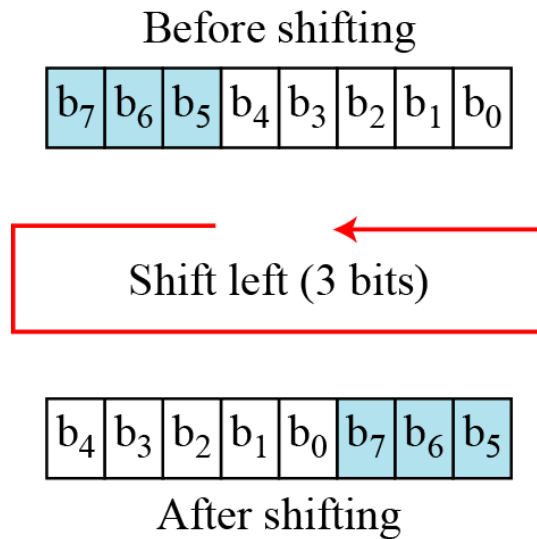
- input of 010 → output will be 01
- input of 101 → output will be 00

XOR

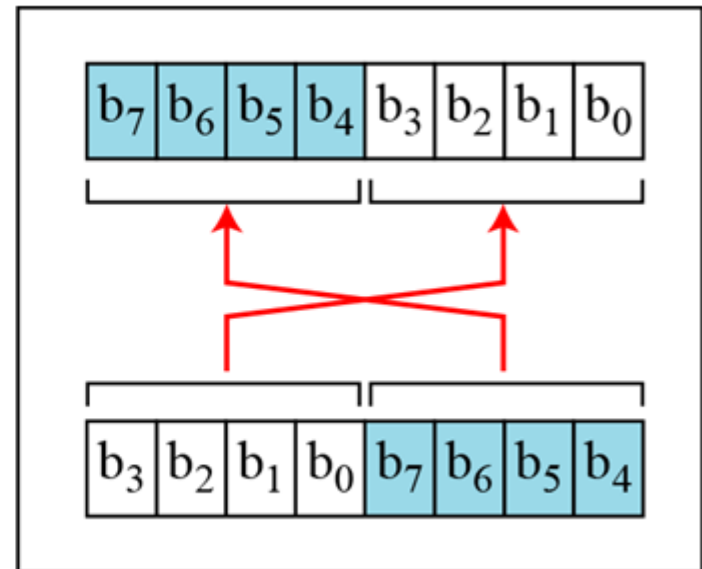
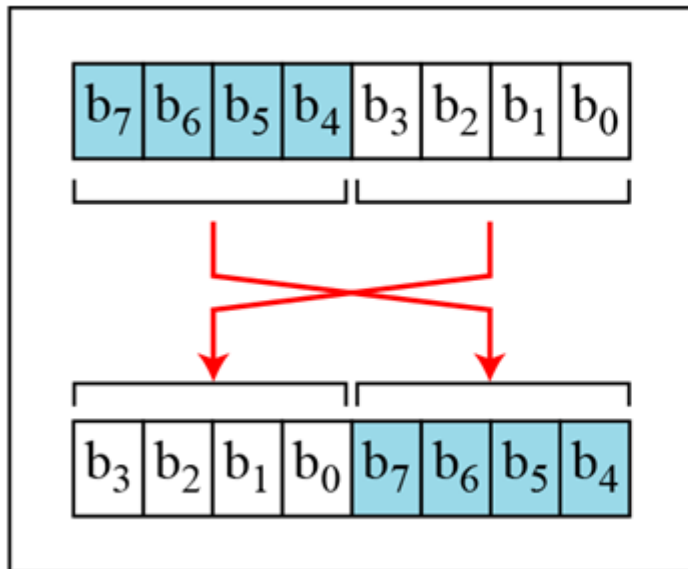
- The five properties of XOR in the $GF(2^n)$ field makes it useful in a block cipher:
 - closure
 - associativity
 - commutativity
 - existence of identity
 - existence of inverse

Circular Shift

- Example:

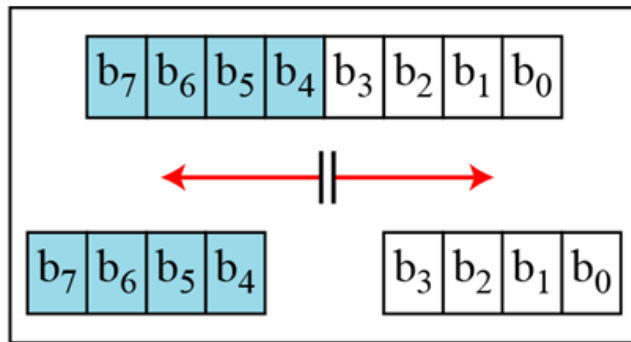


Swap – special case of shifting

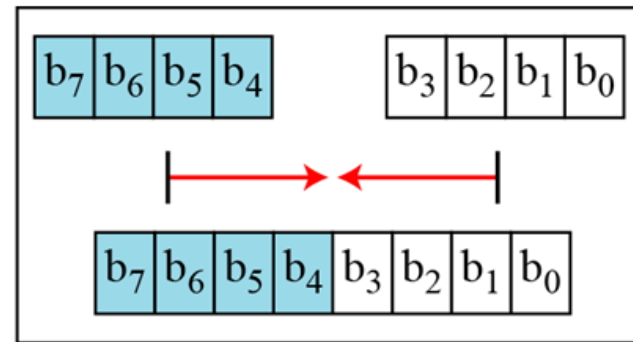


Split and Combine

Split



Combine



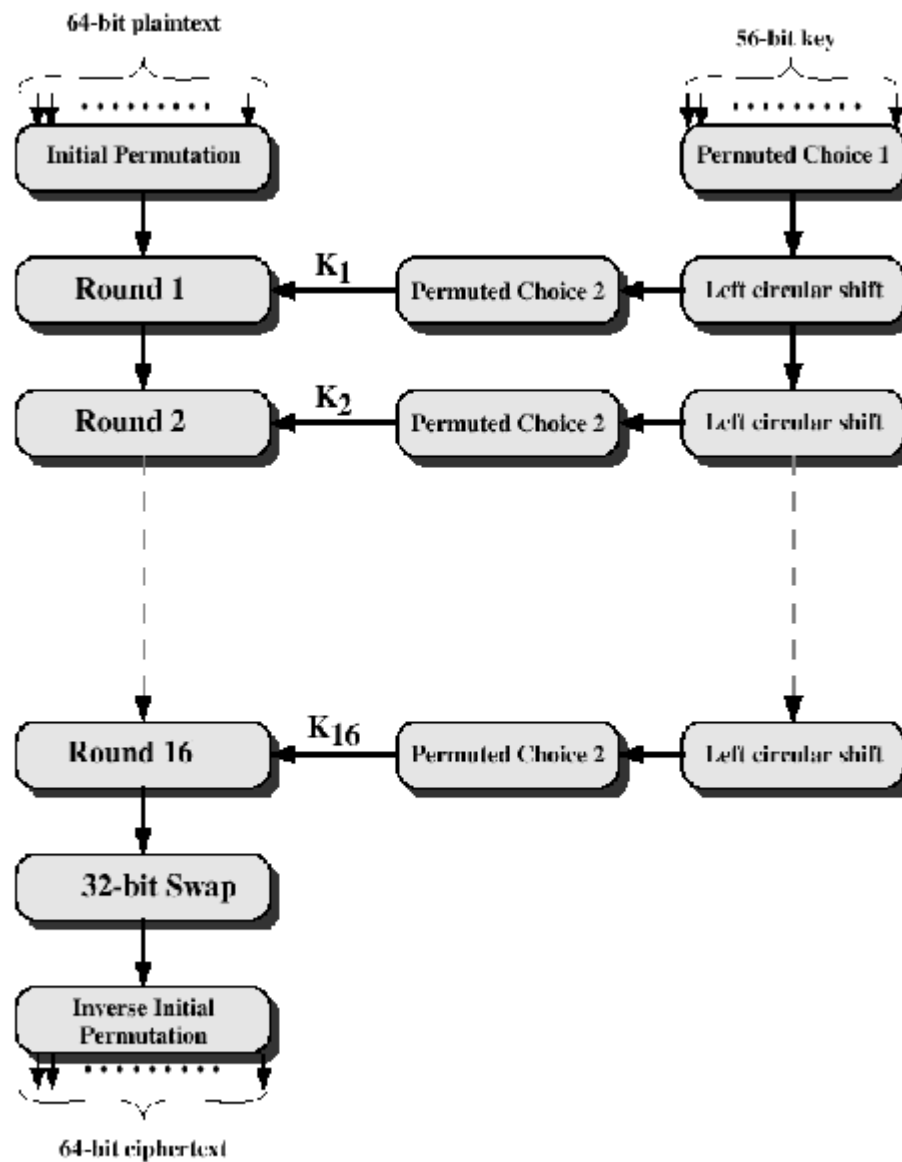
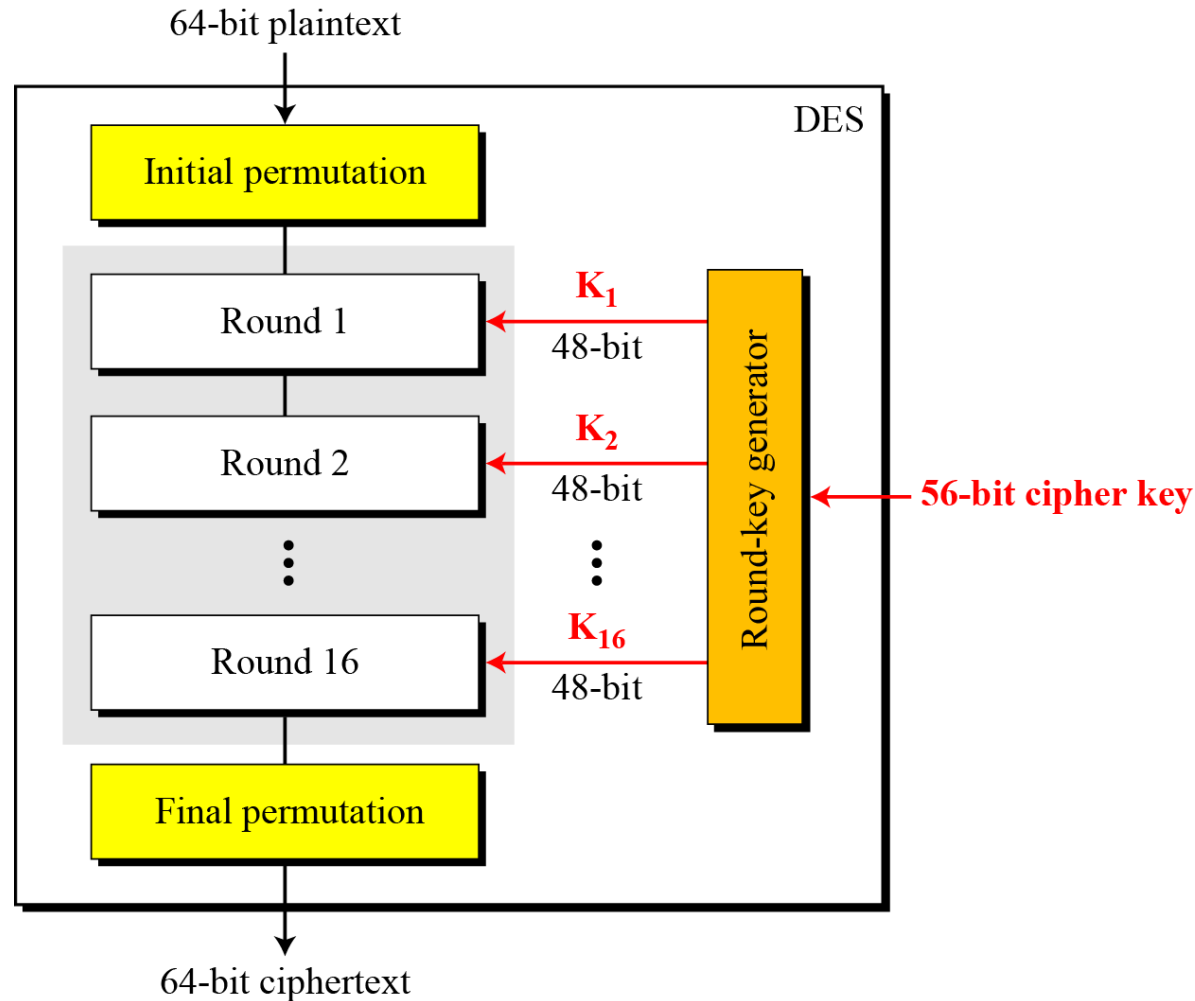


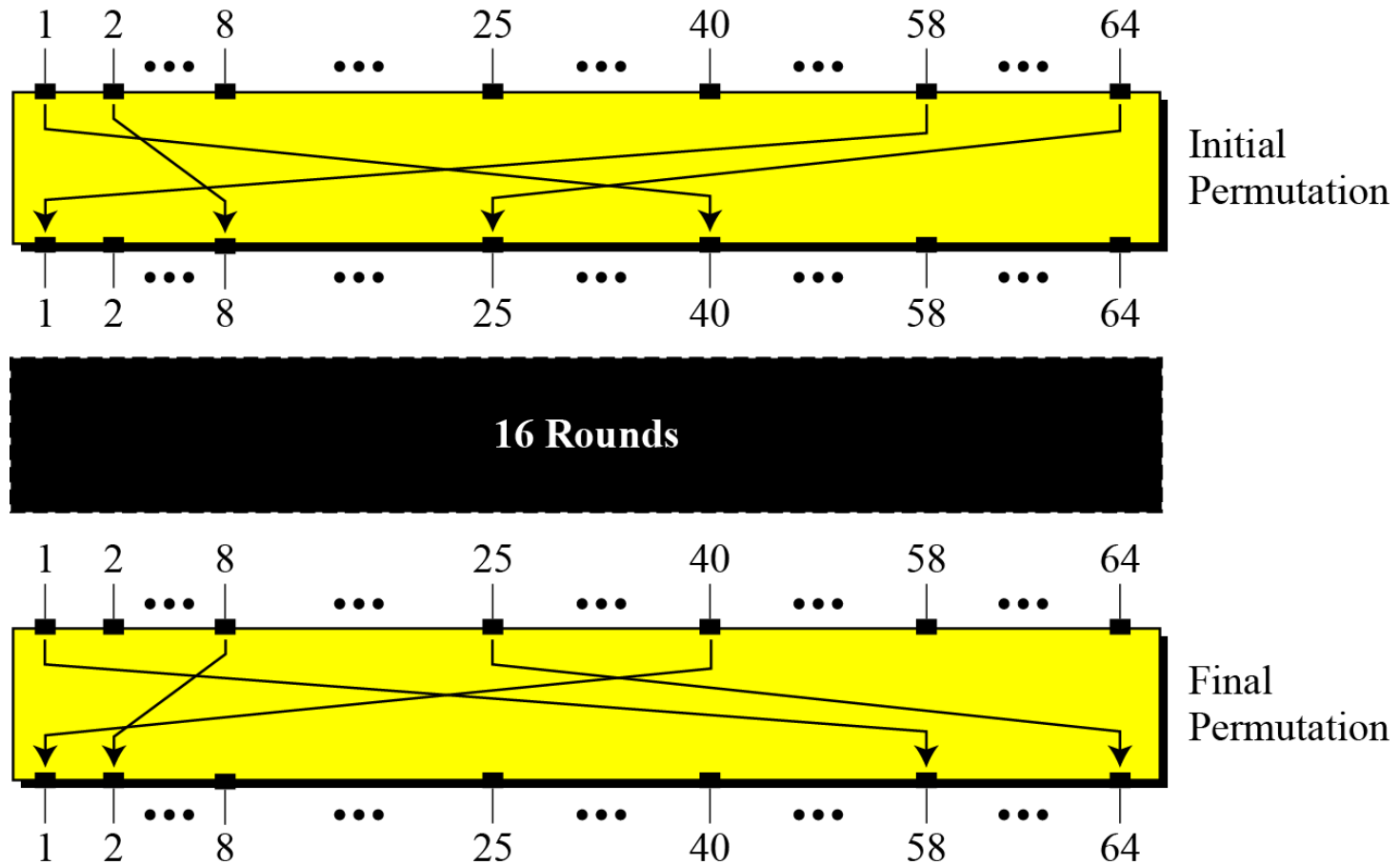
Figure 2.3 General Depiction of DES Encryption Algorithm

DES

- Two P-boxes
 - initial and final permutations
- Sixteen Feistel rounds



P-Boxes



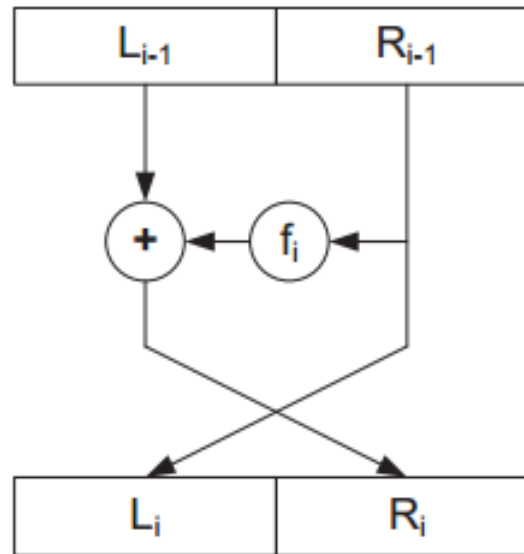
P-Boxes

<i>Initial Permutation</i>	<i>Final Permutation</i>
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

- initial and final permutations are inverses of each other
- no cryptography significance in DES

16 Rounds - Feistel Cipher

- Representative of one round:
 - The block will be split into two portions: Left (L) and Right (R)



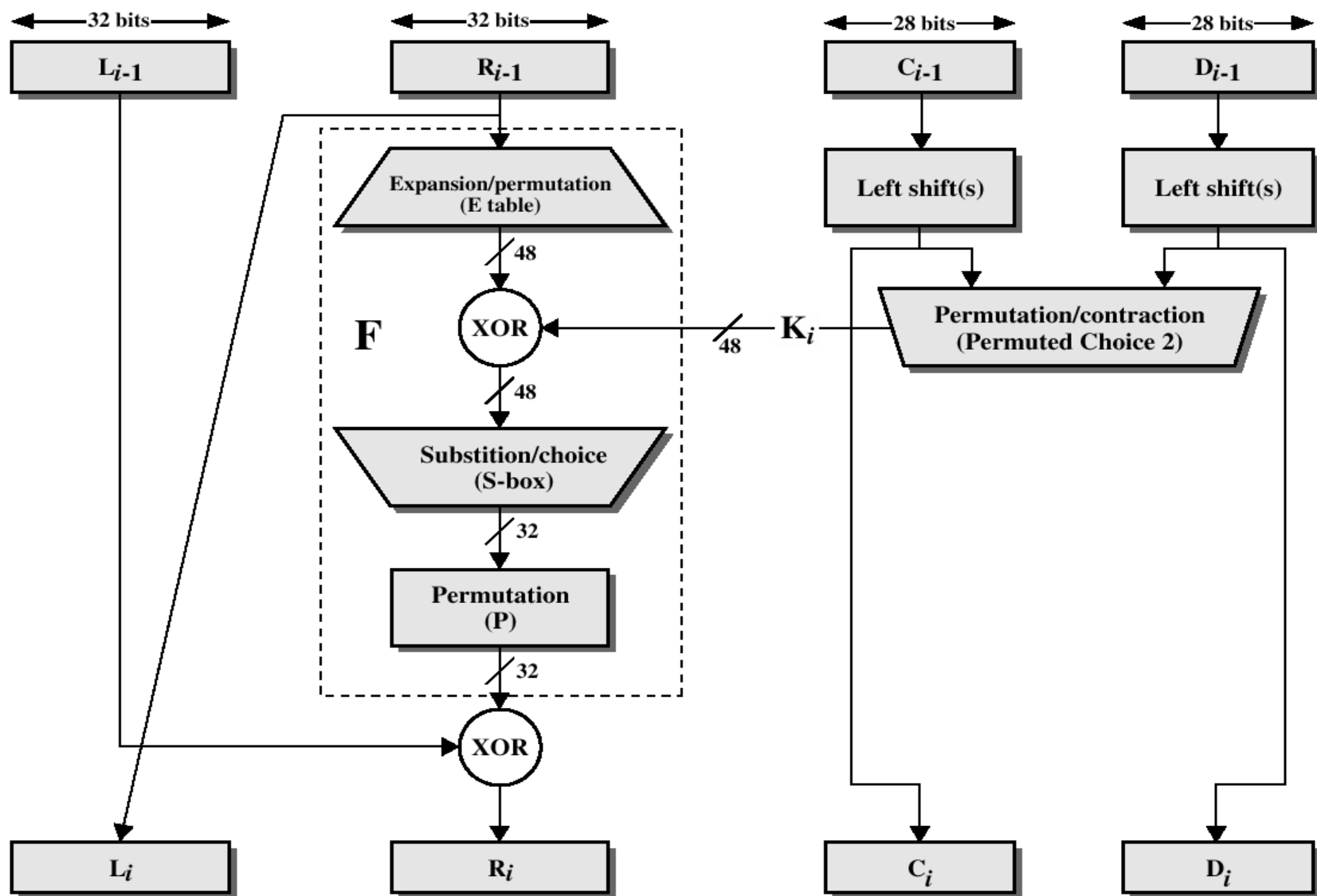
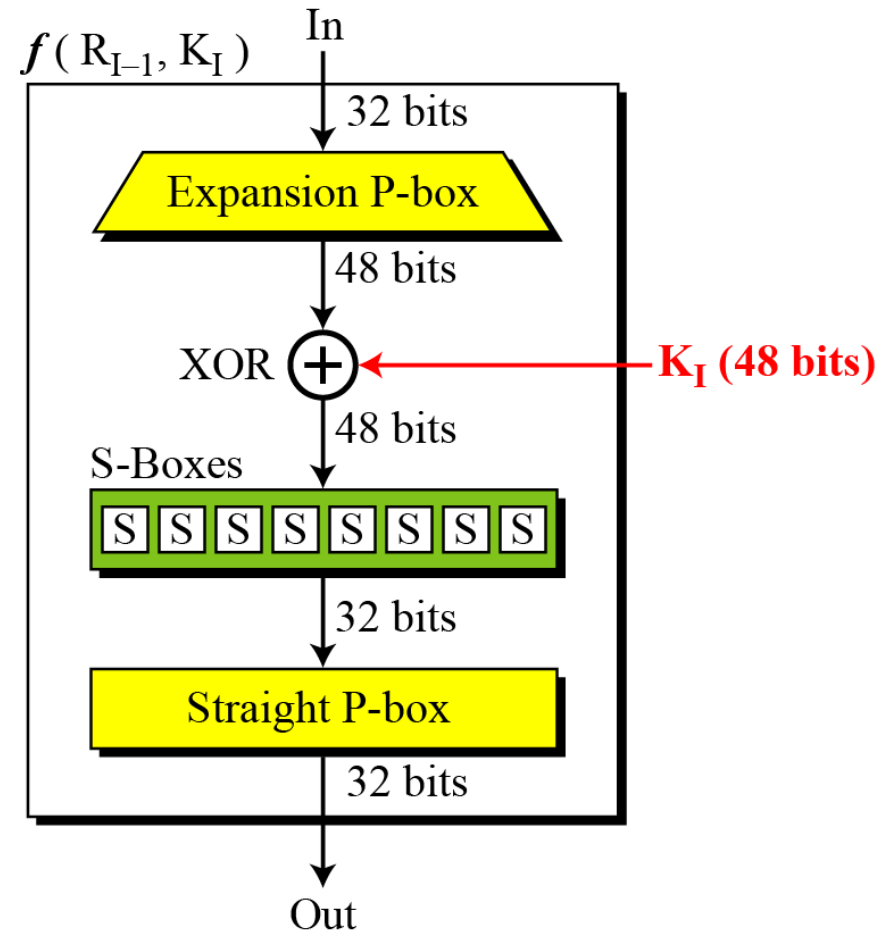
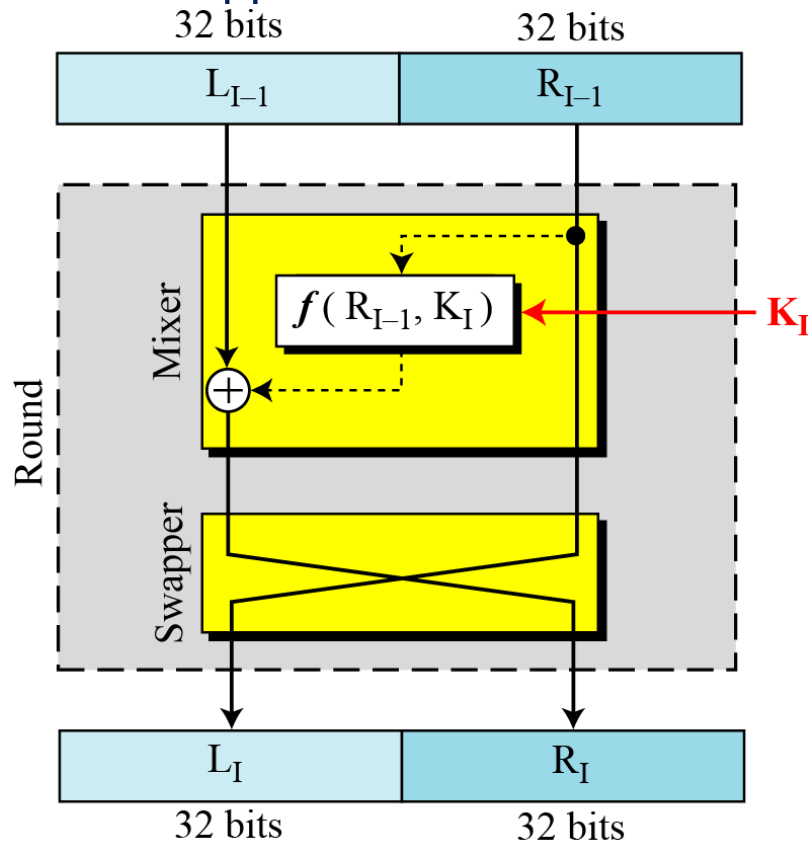


Figure 2.4 Single Round of DES Algorithm

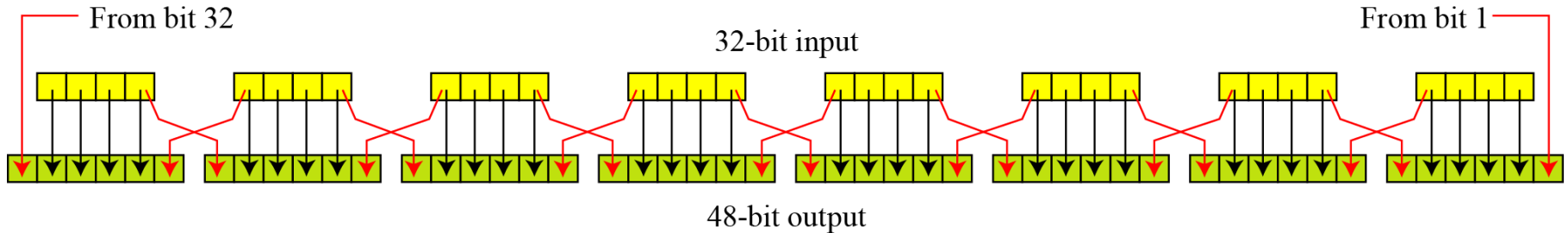
16 Rounds

- Typical round:
 - No swapper in Round 16



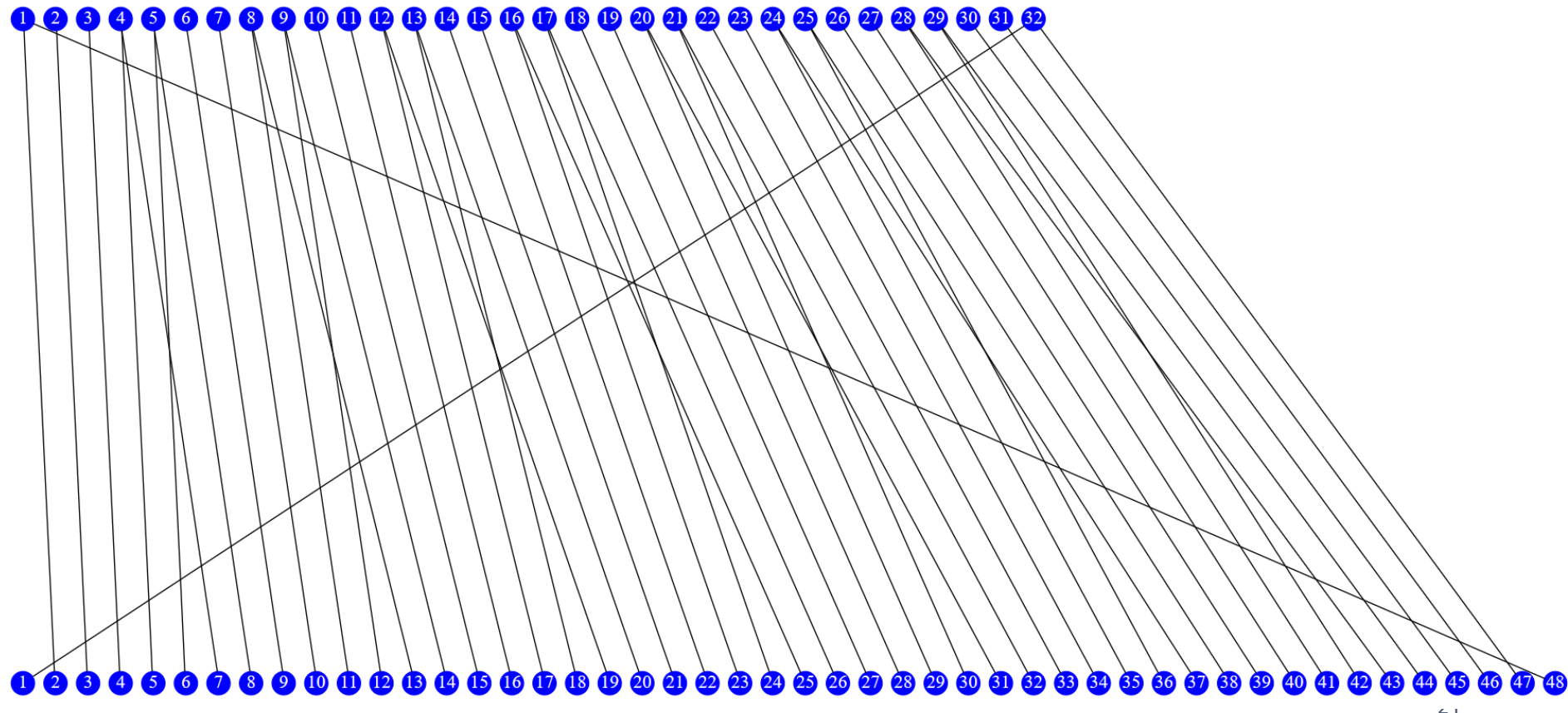
Expansion P-Box in DES Function

- As the key is 48 bits while the input to the DES function f is 32 bits, we need to expand the input 32 bits to 48 bits



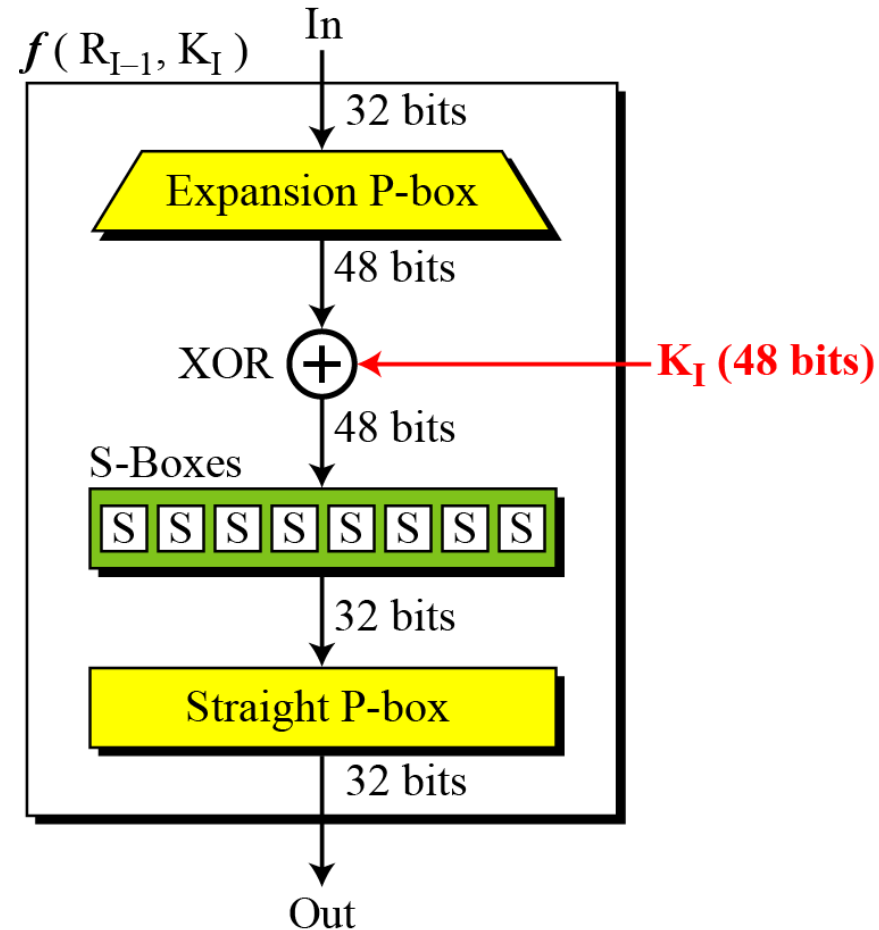
Expansion P-Box

- Mathematically, it is defined by the following table:

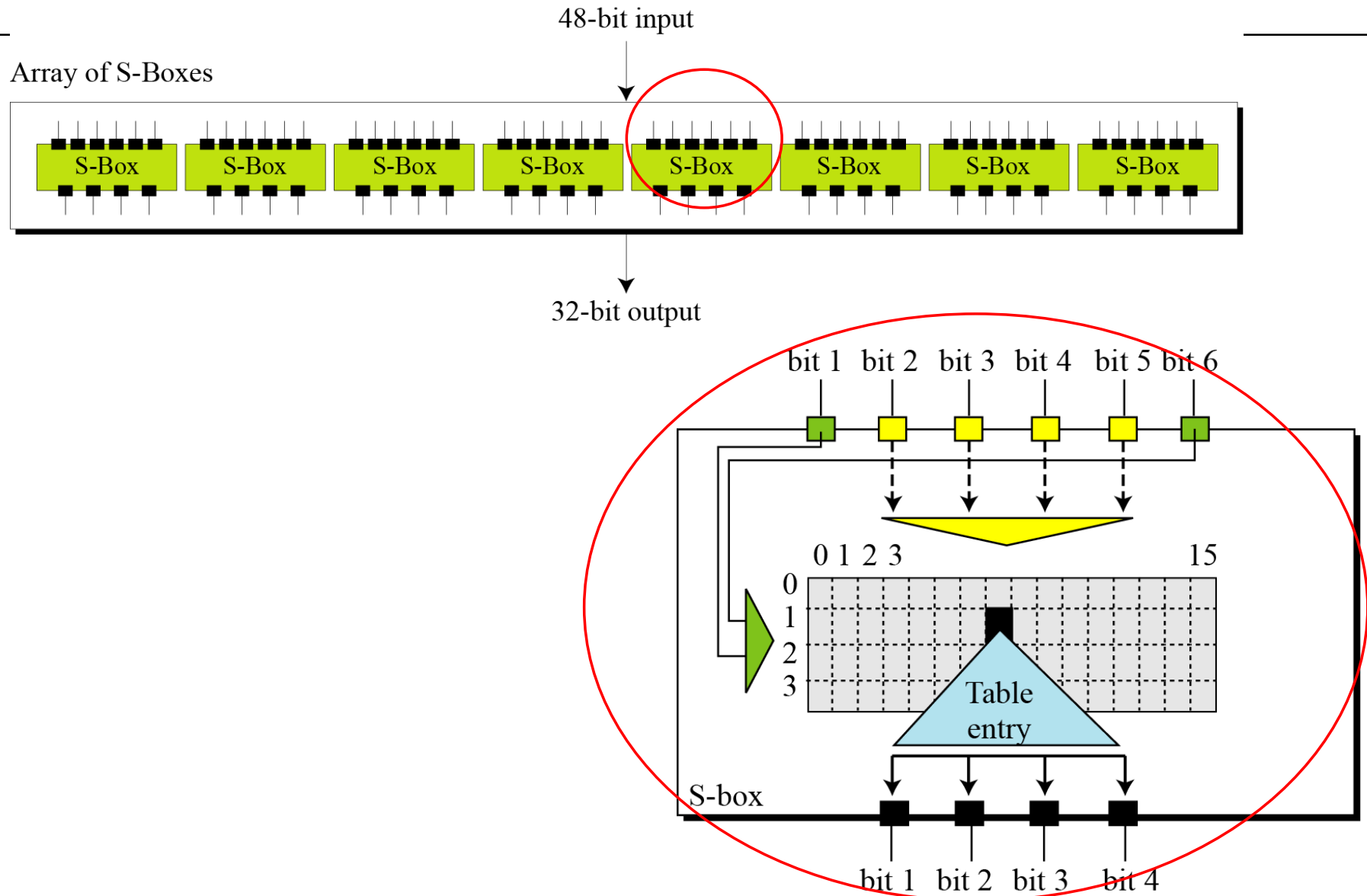


S-Boxes

- The S-boxes do the real mixing (confusion)
- DES uses 8 S-boxes (each with 6-bit input and a 4-bit output)
 - Input: 8 x 6 = 48 bits
 - Output: 8 x 4 = 32 bits



S-Boxes



S-Box 1

- The S-Box 1 can be represented using the following table:

S ₁																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

- Example:
 - Input: 100011
 - Output: 12 or in 4-bit binary "1100"

S-Box 2-4

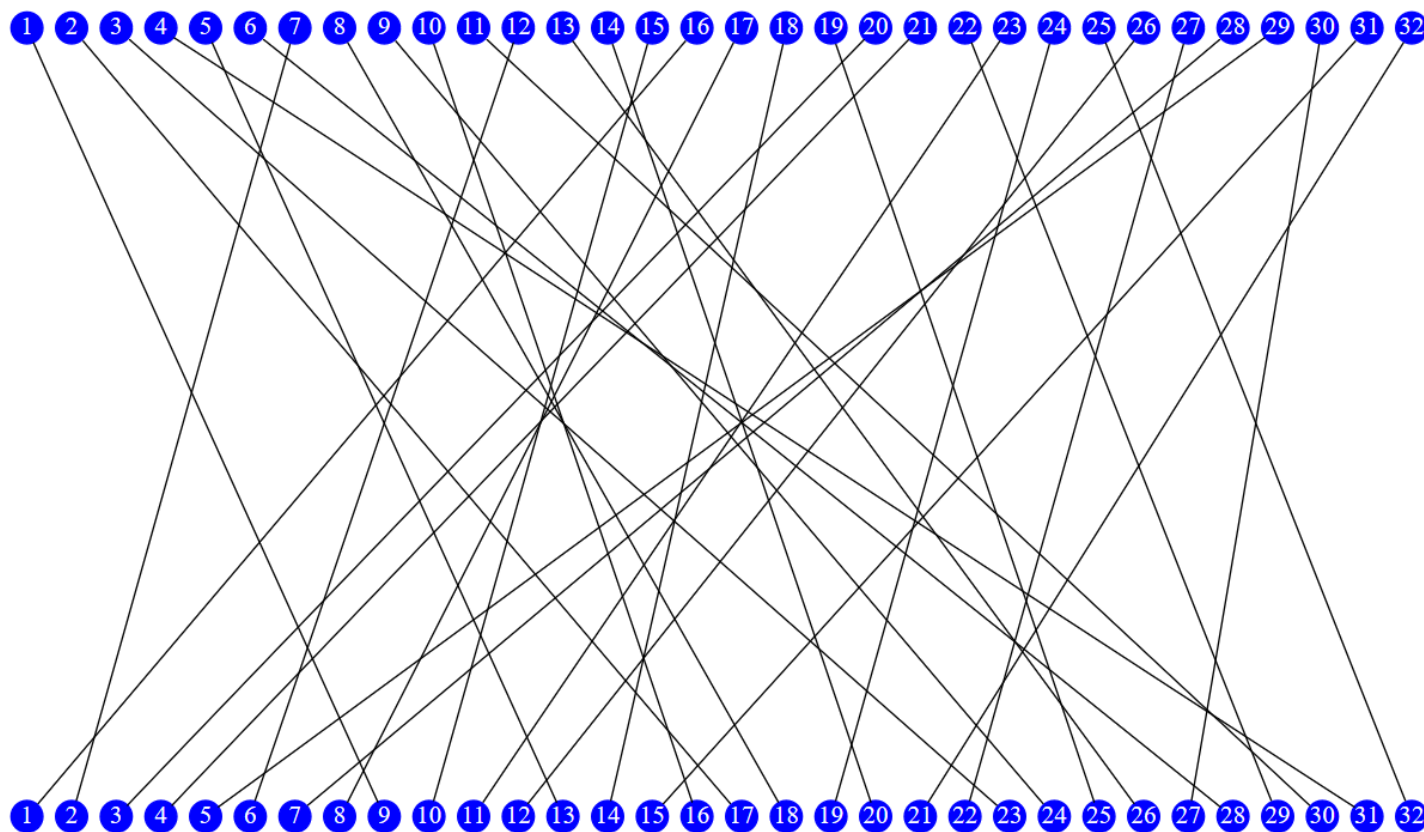
S ₂																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
0yyyy1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
1yyyy0	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
1yyyy1	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S ₃																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
0yyyy1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
1yyyy0	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1yyyy1	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S ₄																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
0yyyy1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
1yyyy0	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
1yyyy1	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box 5-8

S ₅																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
0yyyy1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
1yyyy0	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
1yyyy1	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S ₆																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
0yyyy1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
1yyyy0	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
1yyyy1	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S ₇																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
0yyyy1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1yyyy0	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
1yyyy1	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S ₈																
	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
0yyyy1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
1yyyy0	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
1yyyy1	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

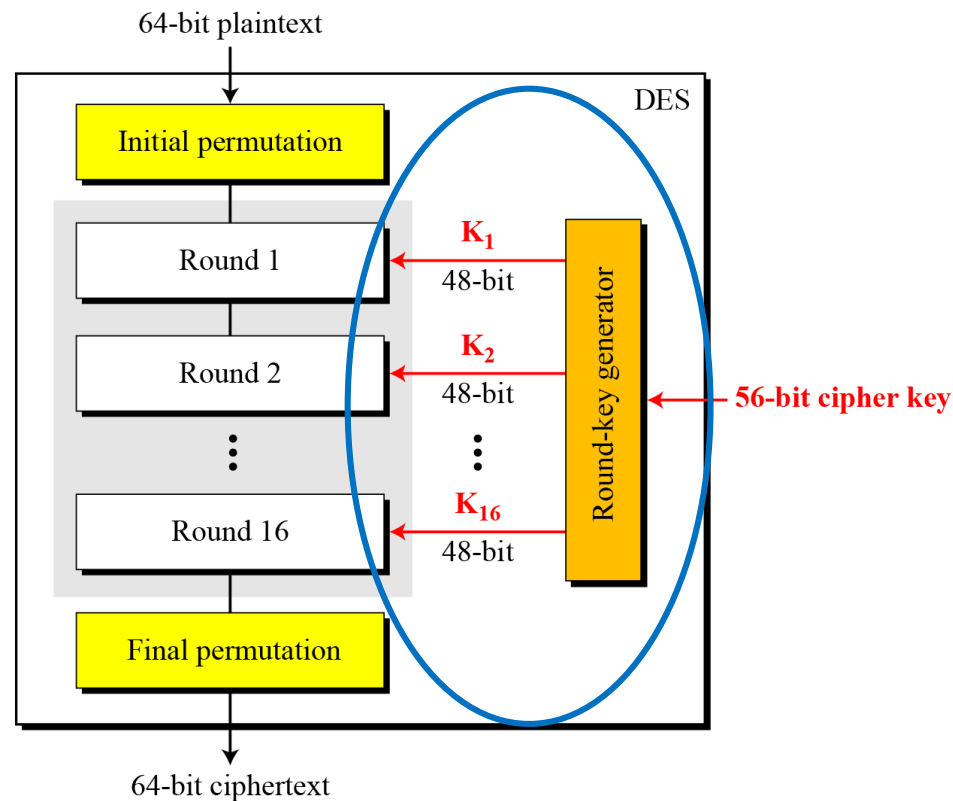
Straight Permutation Table in f

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

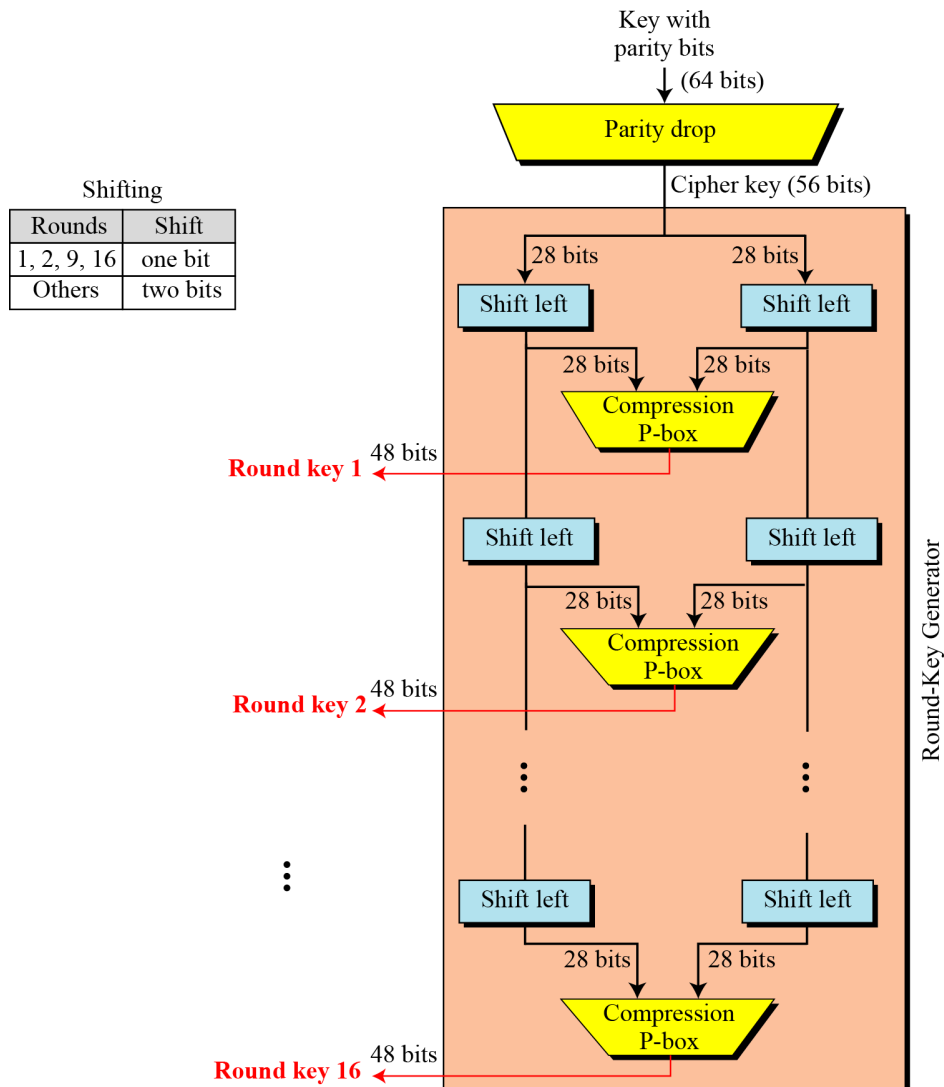


Round-key generator

- Given 56-bit cipher key
 - create sixteen 48-bit key for each of the 16 rounds



Overview of the Key Generation



Parity Bit Drop

- It can be represented by the following table:
- Do you notice which bits have been dropped?
- Why?

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Bit Shifting

- Most round shifts two bits, except round 1, 2, 9, 16
- Split the key k into left and right C, D

Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

<http://page.math.tu-berlin.de/~kant/teaching/hess/kryptows2006/des.htm>

$C_0 = 1111000011001100101010101111$
 $D_0 = 0101010101100110011110001111$

$C_1 = 1110000110011001010101011111$
 $D_1 = 1010101011001100111100011110$

$C_2 = 1100001100110010101010111111$
 $D_2 = 0101010110011001111000111101$

$C_3 = 0000110011001010101011111111$
 $D_3 = 0101011001100111100011110101$

$C_4 = 0011001100101010101111111100$
 $D_4 = 0101100110011110001111010101$

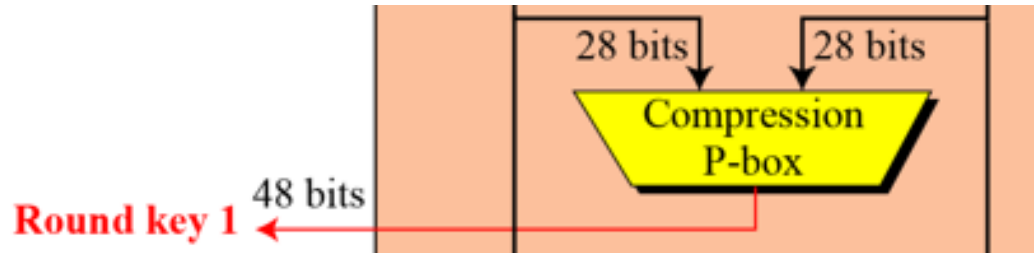
$C_5 = 1100110010101010111111110000$
 $D_5 = 0110011001111000111101010101$

$C_6 = 0011001010101011111111000011$
 $D_6 = 1001100111100011110101010101$

$C_7 = 1100101010101111111100001100$
 $D_7 = 0110011110001111010101010110$

Key Compression Table – P-Box

- Input: 56 bits
- Output: 48 bits



14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Decryption

- The decryption process with DES is essentially the same as the encryption process
 - Use the ciphertext as the input to the DES algorithm
- BUT
 - use the keys K_i in reverse order
 - that is, use K_{16} on the first iteration, K_{15} on the second round, ... K_1 on the 16th (i.e. last) iteration

Exercise

- Google for a DES calculator applet and try to trace for the bit patterns in each round
 - E.g. https://www.tools4noobs.com/online_tools/encrypt/
- Write / Google a DES program in your favorite programming language

Attacks

- Differential Cryptanalysis
 - It is a chosen-plaintext attack
 - Published by Israel researcher Eli Biham and Adi Shamir in late 1980
 - IBM researcher claimed that it was known as early as 1974
 - based on a non-uniform differential distribution table of the S-boxes (due to the XOR operation) in a block cipher

Example Differential Distribution Table

		$C_1 \oplus C_2$			
		<i>00</i>	<i>01</i>	<i>10</i>	<i>11</i>
$P_1 \oplus P_2$	000	1	0	0	0
	001	0.25	0.25	0	0.50
	010	0.25	0.25	0.50	0
	011	0	0.50	0.25	0.25
	100	0.25	0.25	0.50	0
	101	0	0.50	0.25	0.25
	110	0.50	0	0.25	0.25
	111	0	0	0.25	0.75

Attacks

- Linear cryptanalysis
 - uses known plaintext attacks
 - presented by Mitsuru Matsui in 1993
 - idea: some S-boxes might not be totally nonlinear; they can be approximated, probabilistically, by some linear functions

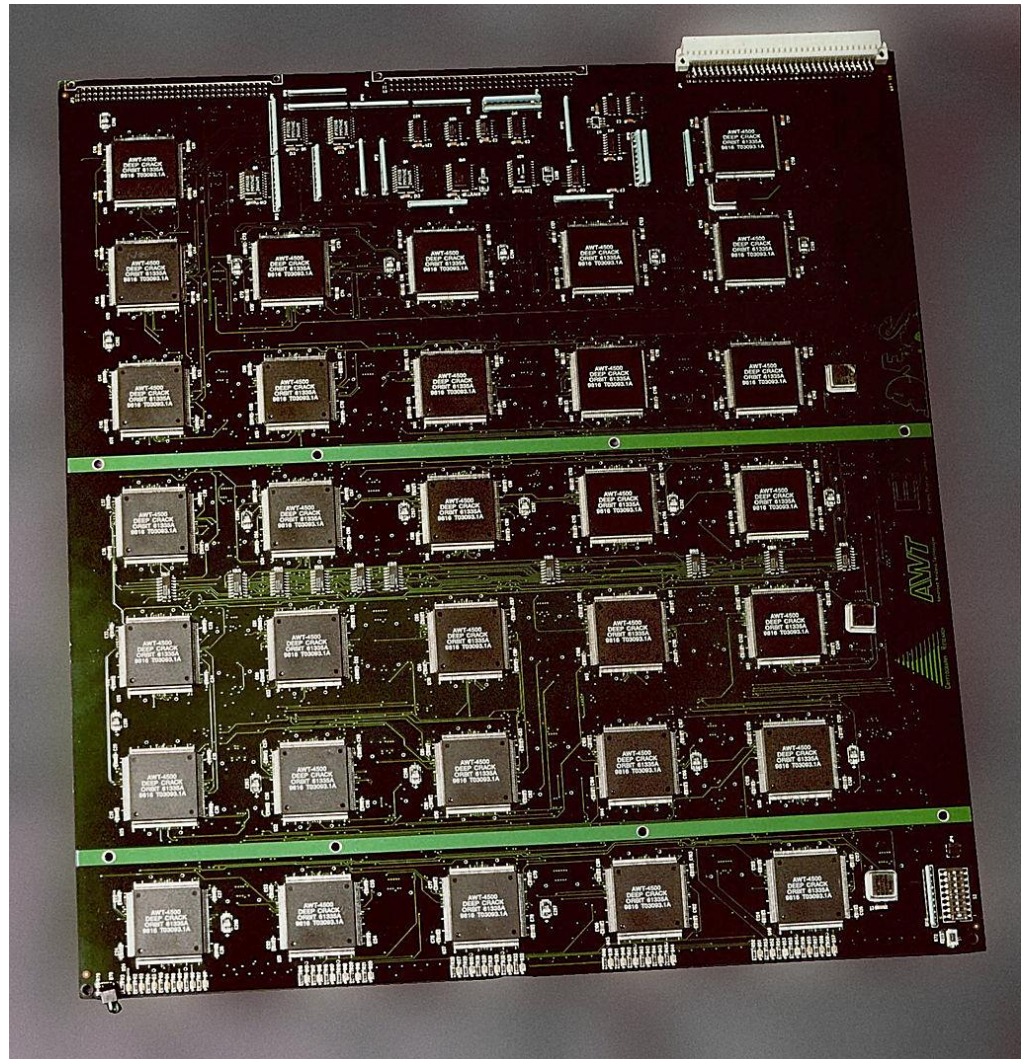
Attack History of DES

Year	Notes	Cost	Runtime
1977	Hellman design	\$20M (est.)	1 day
1993	Wiener design	\$1M (est.)	7 hours
1997	DESMITH	free (as in lemonade)	96 days
1998	EFF (“Deep Crack”)	\$250k	56 hours
2006–2008	COPACOBANA(s)	\$10k	7 days

- COPACOBANA (Cost-Optimized Parallel COde Breaker) consisted of 120 FPGAs (field-programmable gate arrays) of type Xilinx Spartan3-1000 running in parallel

EFF Hacking

- Wiki: The [EFF's](#) US\$250,000 [DES cracking machine](#) contained 1,856 custom chips and could brute force a DES key in a matter of days—the photo shows a DES Cracker circuit board fitted with several Deep Crack chips.



So, what can we do?

- Use Triple Data Encryption Algorithm (TDEA)
 - aka Triple DES (3DES)
 - extend the key length to 168 bits (3TDEA) or 112 bits (2TDEA)
 - $C = E_{K_3}\{D_{K_2}[E_{K_1}(P)]\}$
 - K_i 's are all different $\rightarrow$ effective key length: 168 bits
 - $K_1 \neq K_2$ but $K_3 = K_1$ $\rightarrow$ effective key length: 112 bits
 - BUT the operation is computationally slow
- Develop a new strong algorithm $\rightarrow$ AES

Lecture 4: Symmetric Key Ciphers

- DES
- AES
- Stream Cipher
- Mode of Operations
- Considerations

History of AES

- In 1997 NIST called for candidates to replace DES and the new algorithm would become the Advanced Encryption Standard (AES)
 - key-sizes: 128, 192, 256
 - 1998: 15 candidates
 - 1999: 5 finalists
 - MARS, RC6, Rijndael, Serpent, Twofish
- Rijndael was selected to be the AES and it became the new standard effective from 2002
 - FIPS 197 standard

AES selection criteria

- security
- software implementations C/java, 8/32/64-bit processors
- ROM/RAM requirements
- hardware implementations ASIC/FPGA
- instruction-level parallelism
- speed
- susceptibility to timing/power attacks
- encryption vs decryption
- key agility (switch keys quickly)
- versatility (block/key/round sizes)

Choosing AES

(Table from Twofish Paper)

Cipher	Speed (32)	Speed (8)	Safety Factor	Simplicity (code size)
Serpent	62	69	3.56	341 KB
MARS	23	34	1.90	85 KB
RC6	15	43	1.18	48 KB
Rijndael	18	20	1.11	98 KB
Twofish	16	18	2.67	104 KB

(cycles/byte encrypt)

Rijndael AES evaluation

General Security

Rijndael has no known security attacks. Rijndael uses S-boxes as nonlinear components. Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have facilitated its security analysis during the timeframe of the AES development process.

Software Implementations

Rijndael performs encryption and decryption very well across a variety of platforms, including 8-bit and 64-bit platforms, and DSPs. However, there is a decrease in performance with the higher key sizes because of the increased number of rounds that are performed. Rijndael's high inherent parallelism facilitates the efficient use of processor resources, resulting in very good software performance even when implemented in a mode not capable of interleaving. Rijndael's key setup time is fast.

Restricted-Space Environments

In general, Rijndael is very well suited for restricted-space environments where either encryption or decryption is implemented (but not both). It has very low RAM and ROM requirements. A drawback is that ROM requirements will increase if both encryption and decryption are implemented simultaneously, although it appears to remain suitable for these environments. The key schedule for decryption is separate from encryption.

Hardware Implementations

Rijndael has the highest throughput of any of the finalists for feedback modes and second highest for non-feedback modes. For the 192 and 256-bit key sizes, throughput falls in standard and unrolled implementations because of the additional number of rounds. For fully pipelined implementations, the area requirement increases, but the throughput is unaffected.

Rijndael AES evaluation

Attacks on Implementations

The operations used by Rijndael are among the easiest to defend against power and timing attacks. The use of masking techniques to provide Rijndael with some defense against these attacks does not cause significant performance degradation relative to the other finalists, and its RAM requirement remains reasonable. Rijndael appears to gain a major speed advantage over its competitors when such protections are considered.

Encryption vs. Decryption

The encryption and decryption functions in Rijndael differ. One FPGA study reports that the implementation of both encryption and decryption takes about 60% more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than for encryption.

Key Agility

Rijndael supports on-the-fly subkey computation for encryption. Rijndael requires a one-time execution of the key schedule to generate all subkeys prior to the first decryption with a specific key. This places a slight resource burden on the key agility of Rijndael.

Other Versatility and Flexibility

Rijndael fully supports block sizes and key sizes of 128 bits, 192 bits and 256 bits, in any combination. In principle, the Rijndael structure can accommodate any block sizes and key sizes that are multiples of 32, as well as changes in the number of rounds that are specified.

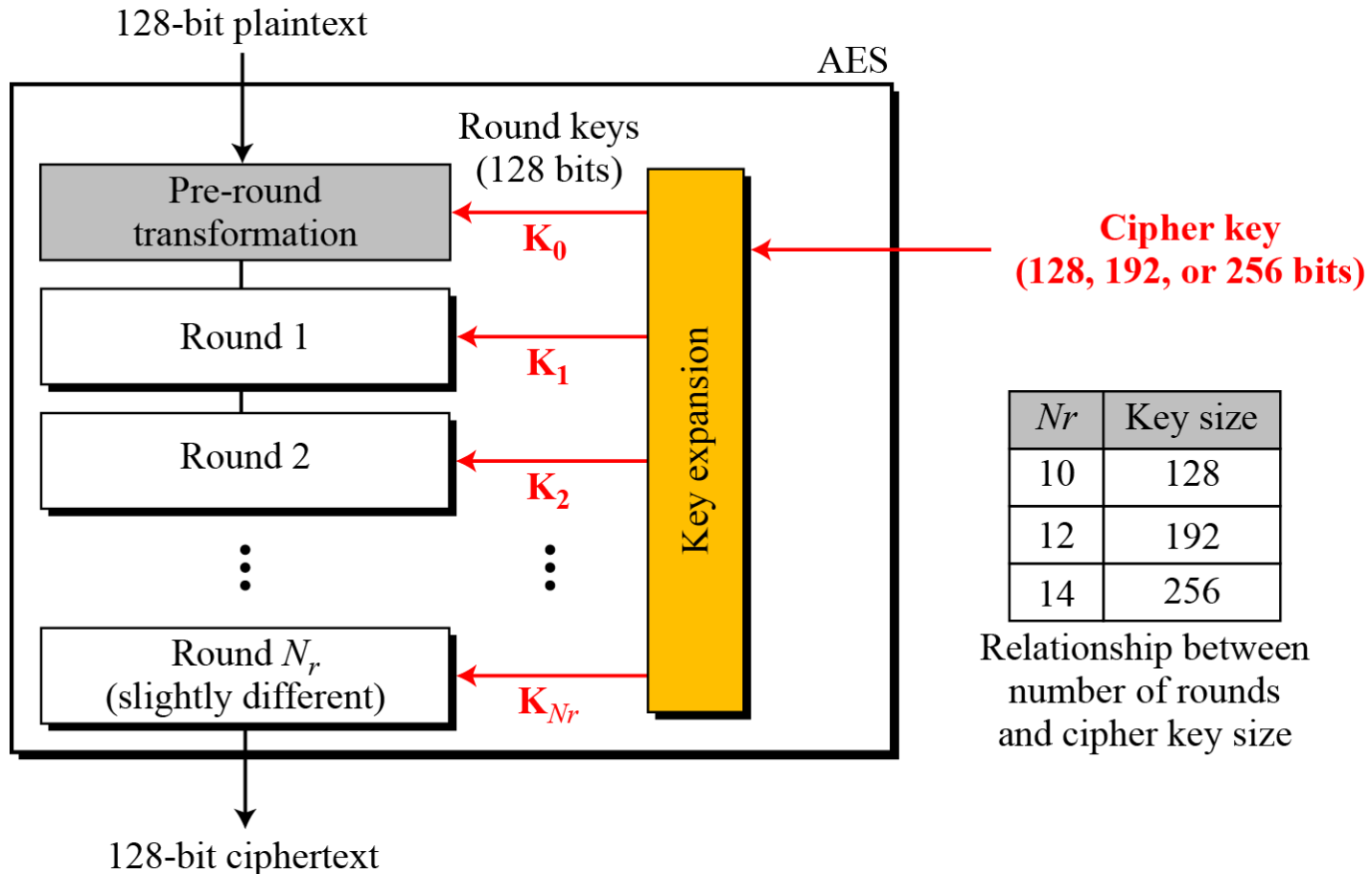
Potential for Instruction-Level Parallelism

Rijndael has an excellent potential for parallelism for a single block encryption.

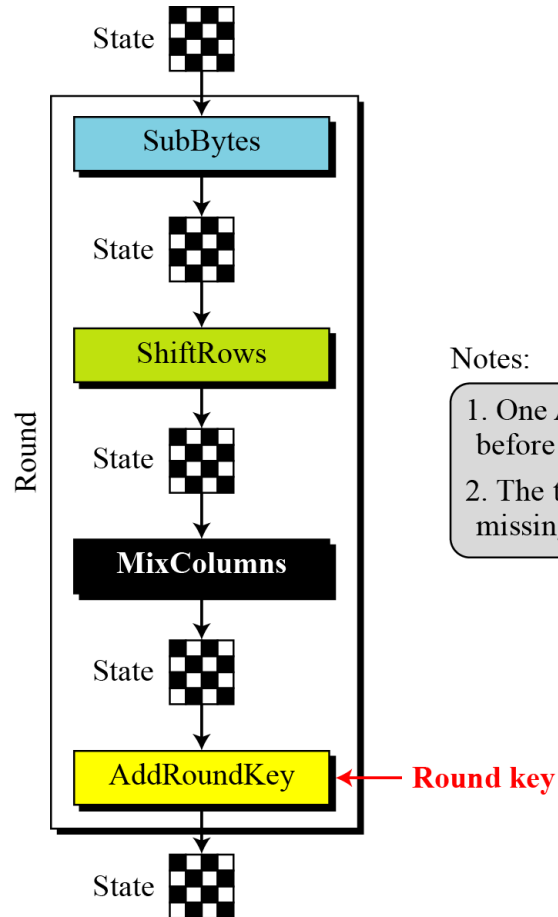
Basic Information

- AES is a non-Feistel cipher
- AES has defined three versions, with 10, 12, and 14 rounds
 - data block size: 128 bits
 - key size: can be 128, 192, or 256 bits

General Design of AES



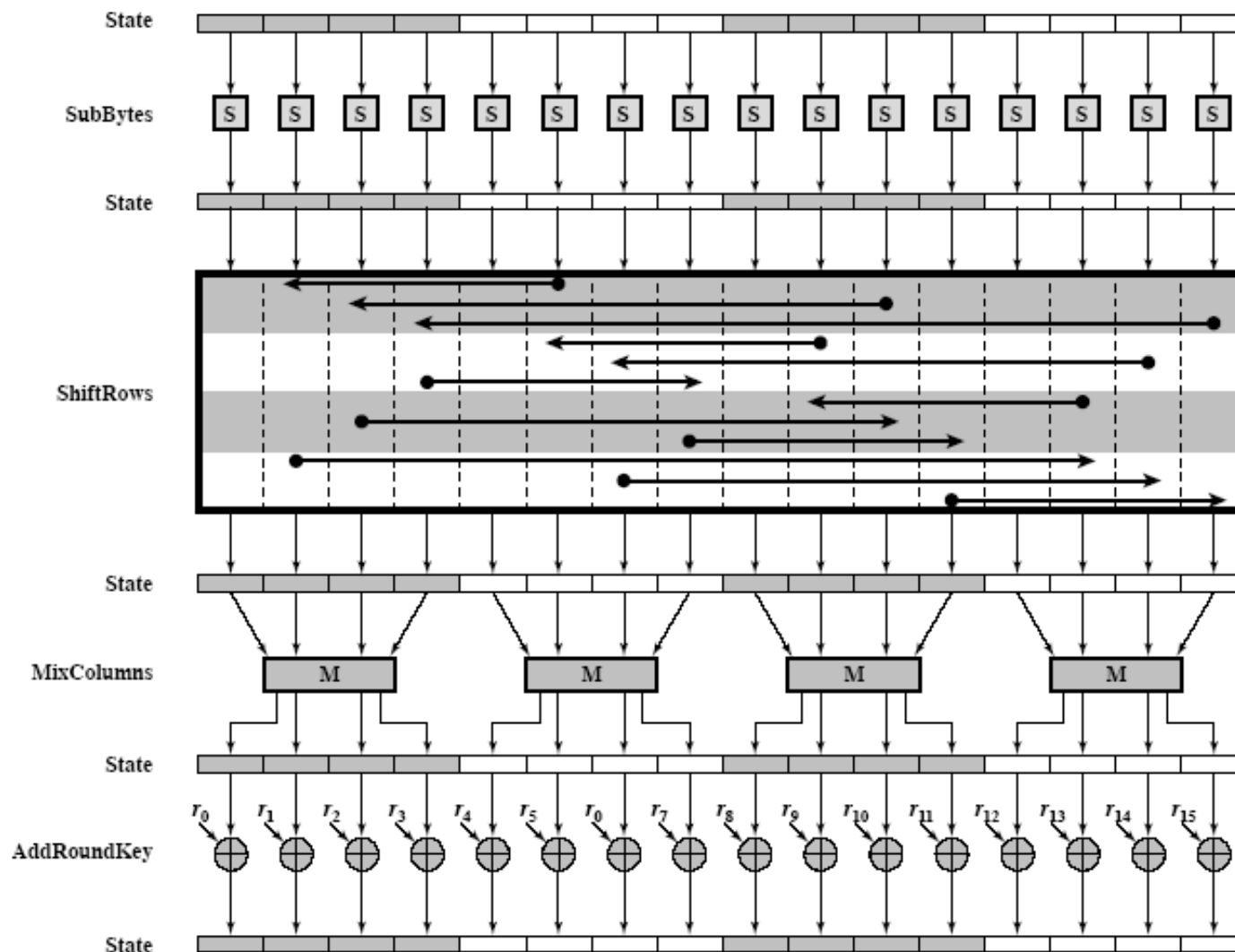
Operation in Each Round



Notes:

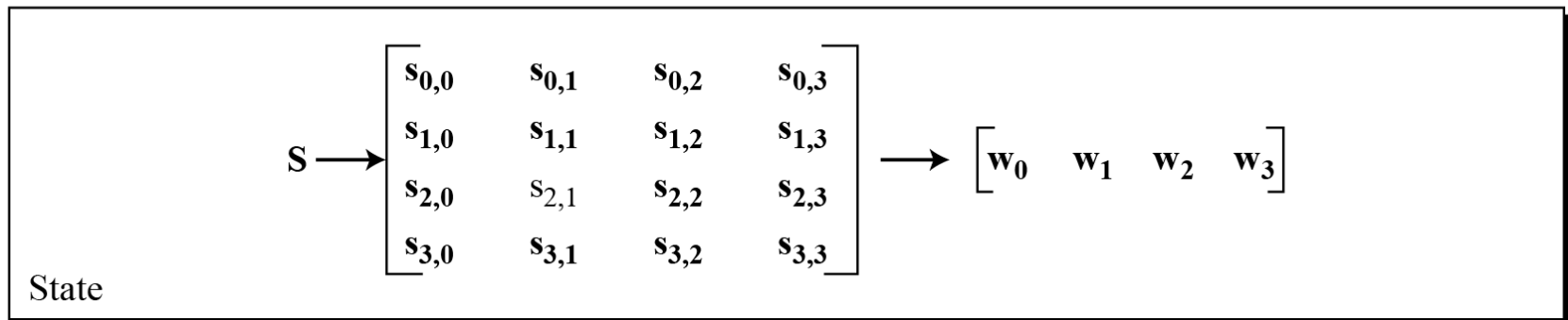
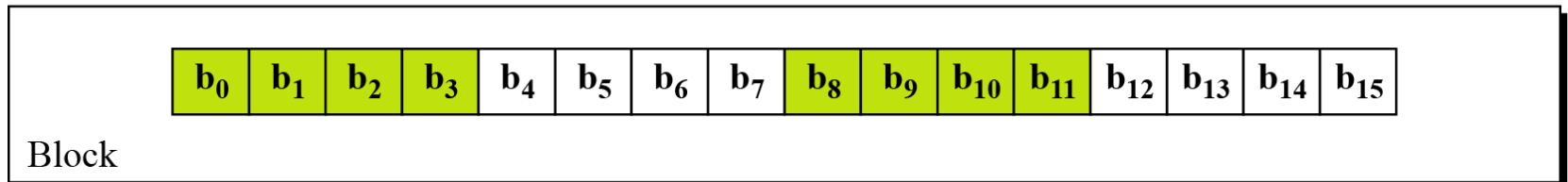
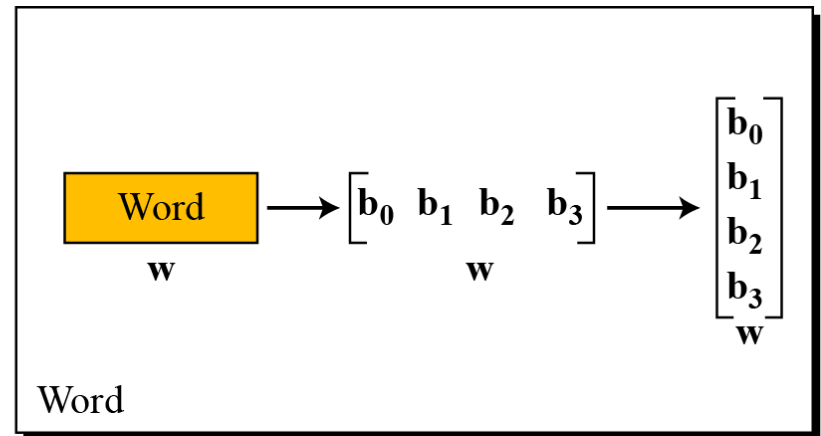
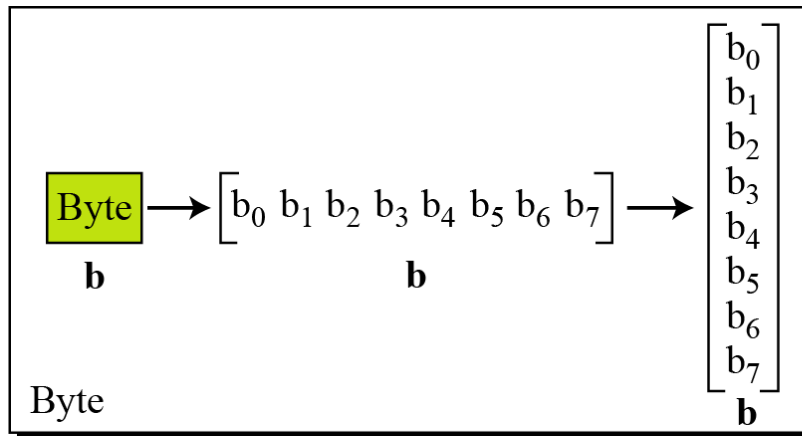
1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

Operation in Each Round – Alternative View

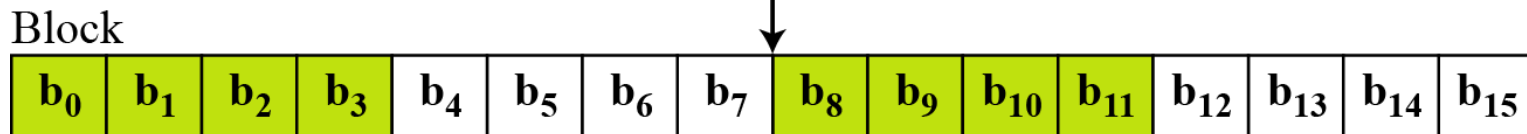
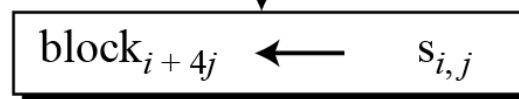
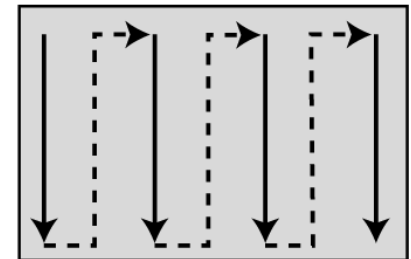
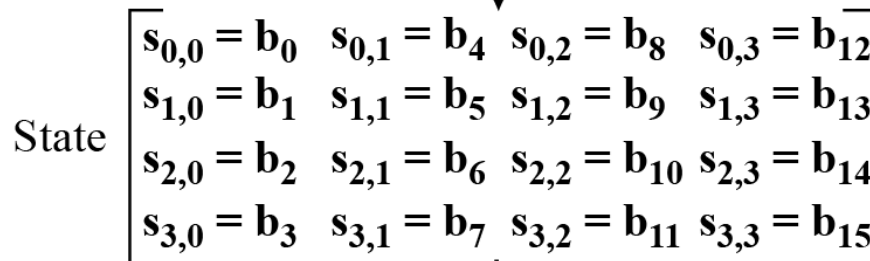
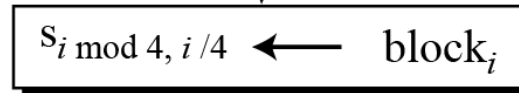


Terminology:

Byte, word, block, state



Block to state & State to block



Changing plaintext to state

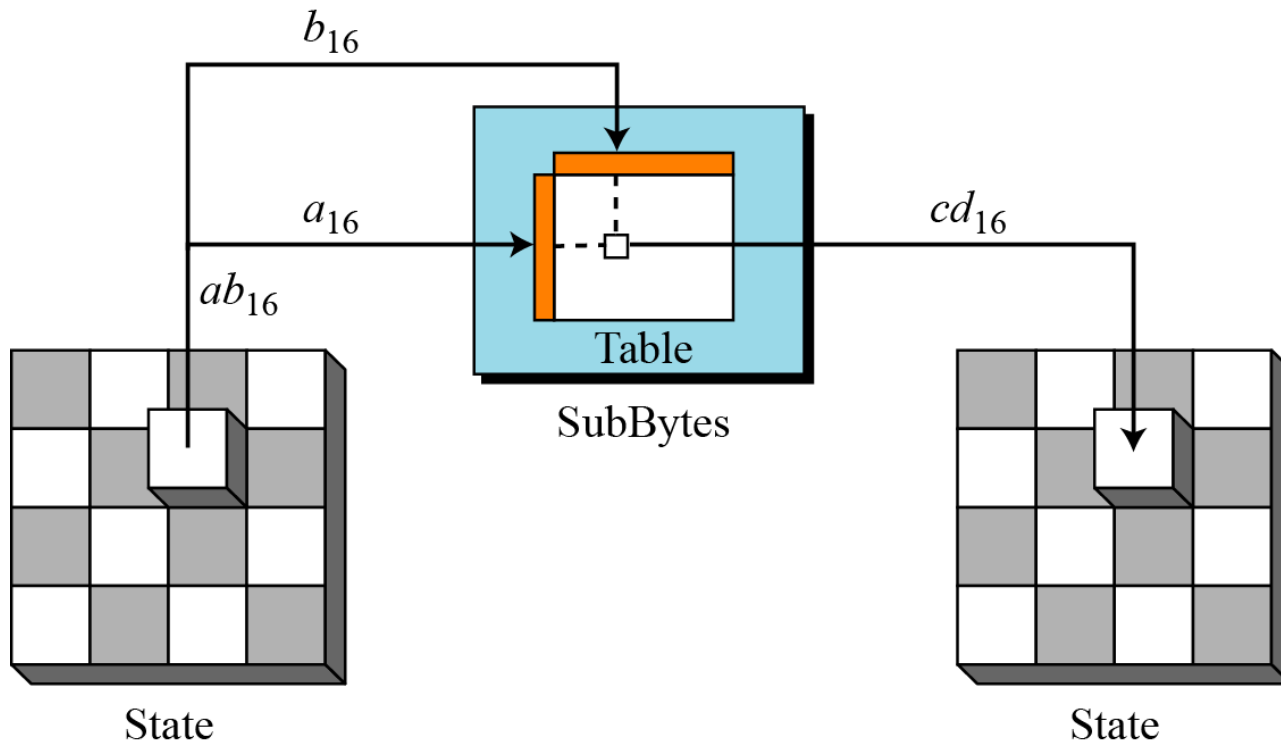
- 128-bit (16 byte) data block is treated as 4x4 byte array
 - Padding at the end (ZZ) to fit the block size (128 bits = 16 ASCII characters)
- Example:
 - Map A to 00_H, B to 01_H, ..., Z to 19_H (A = 0, Z = 25)

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

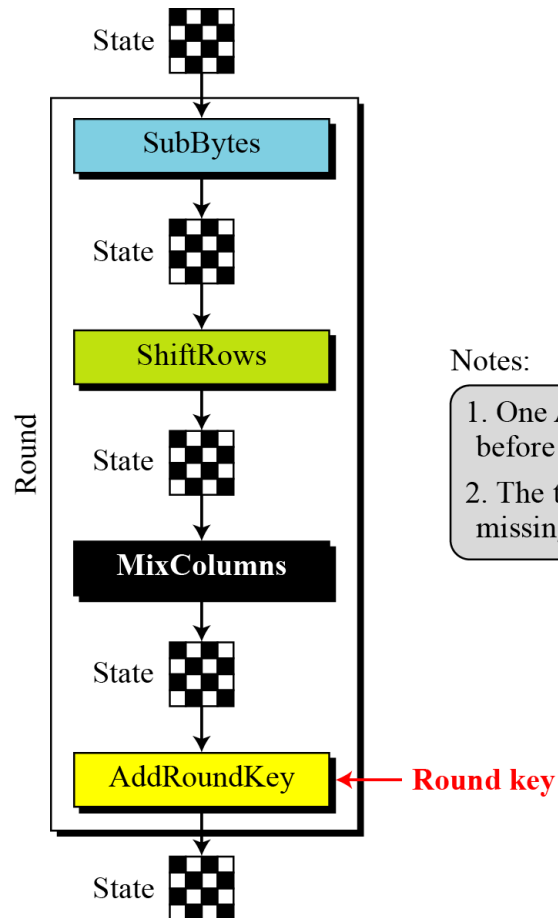
$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \text{State}$$

SubBytes (or S-box)

- The SubBytes operation involves 16 independent byte-to-byte transformations
 - To substitute a byte, interpret the byte as two hexadecimal digits



Operation in Each Round



Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

SubBytes Transformation Table

- The S-box used is derived from the multiplicative inverse over $GF(2^8)$ with an invertible affine transformation

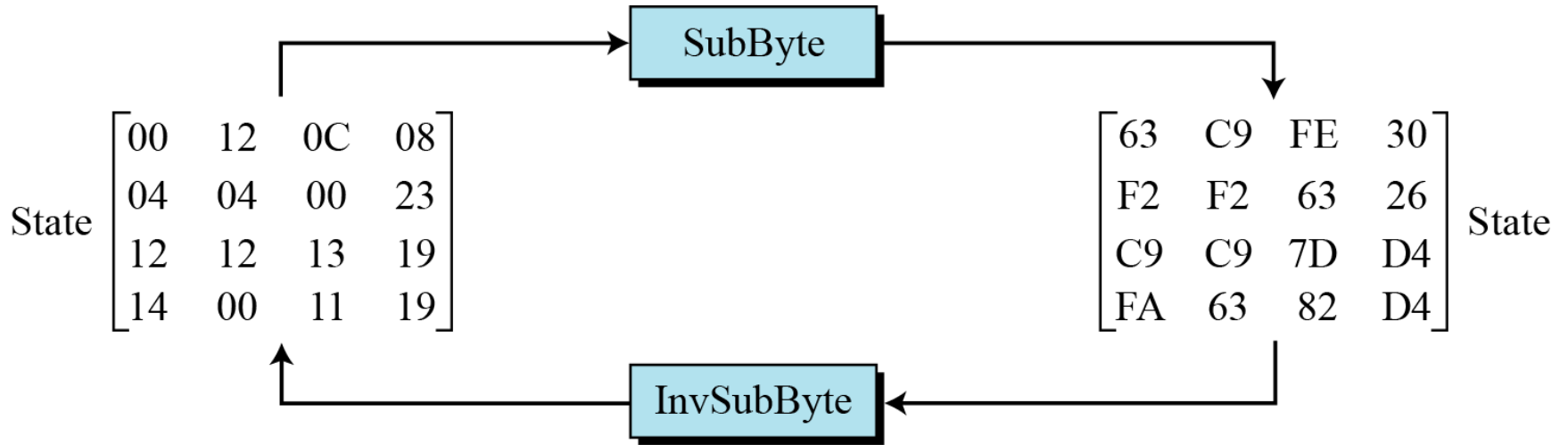
		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	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
	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

Inverse SubBytes Transformation Table

- Inverse S-Box values:

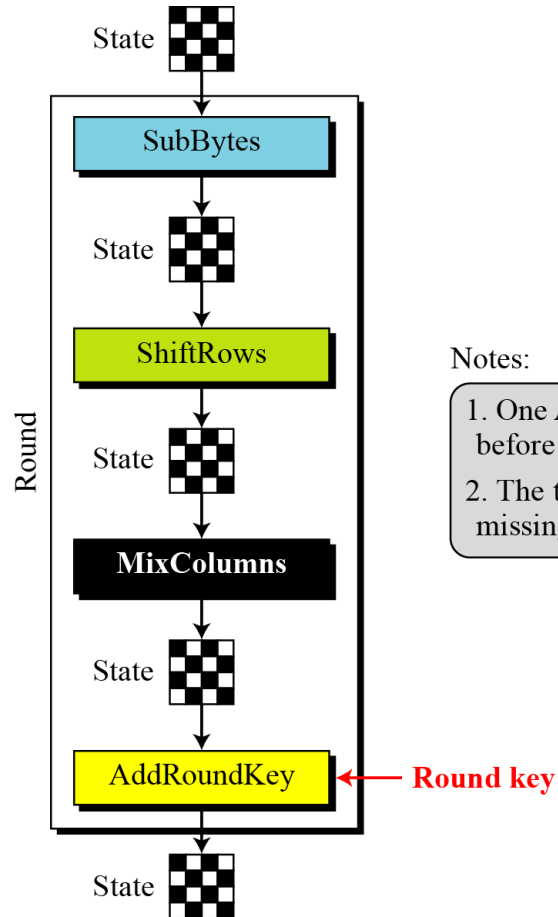
		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	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

Example



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	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
	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

Operation in Each Round

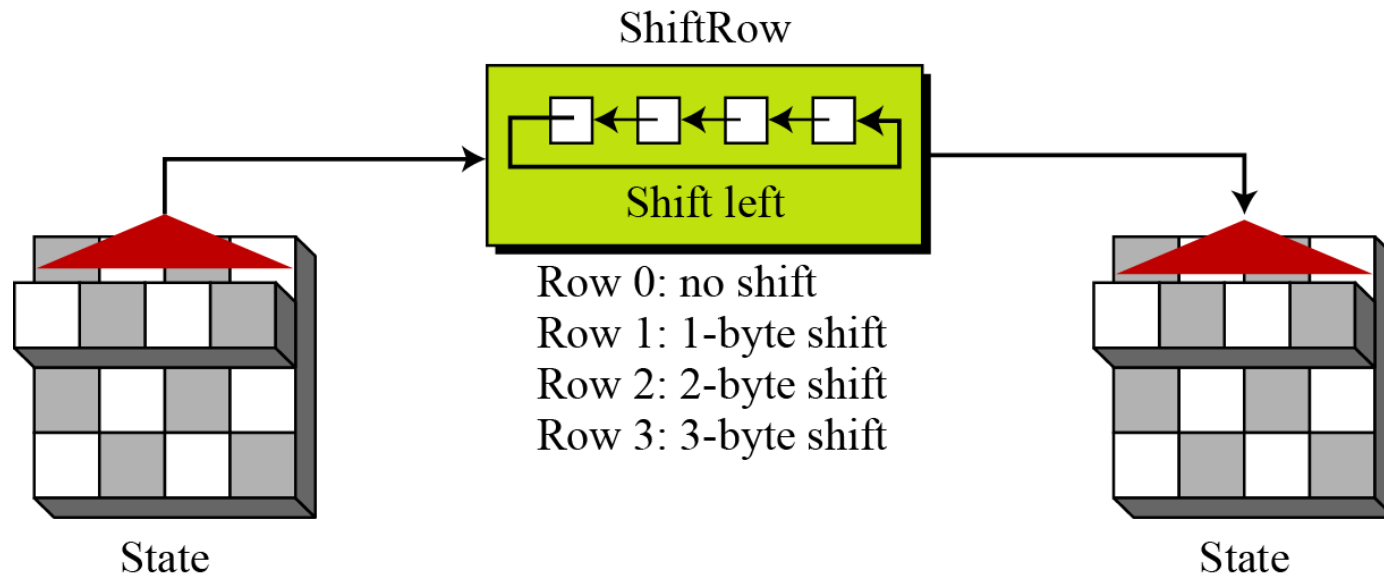


Notes:

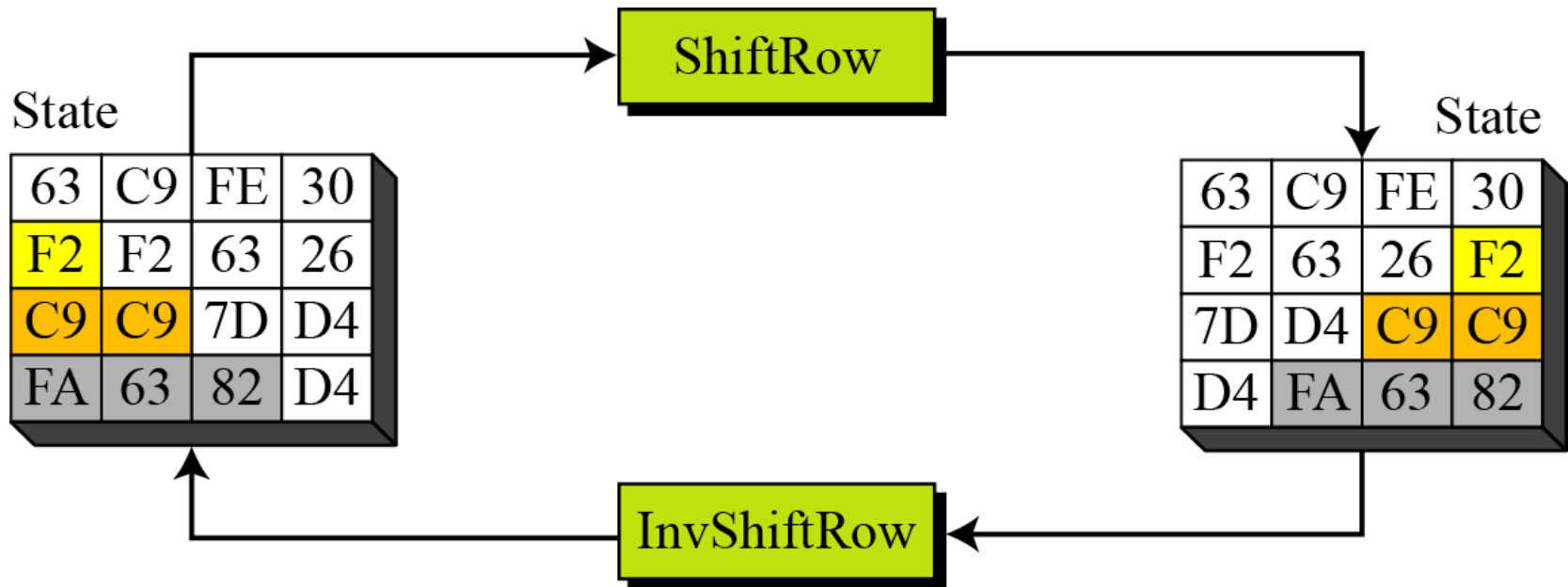
1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

ShiftRows

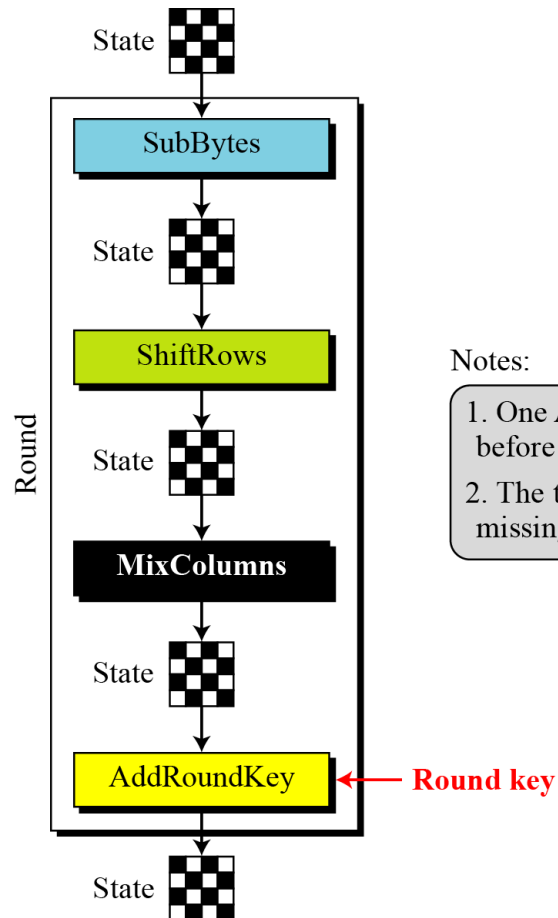
- This operation permutes the bytes



Example



Operation in Each Round



Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

Mixing

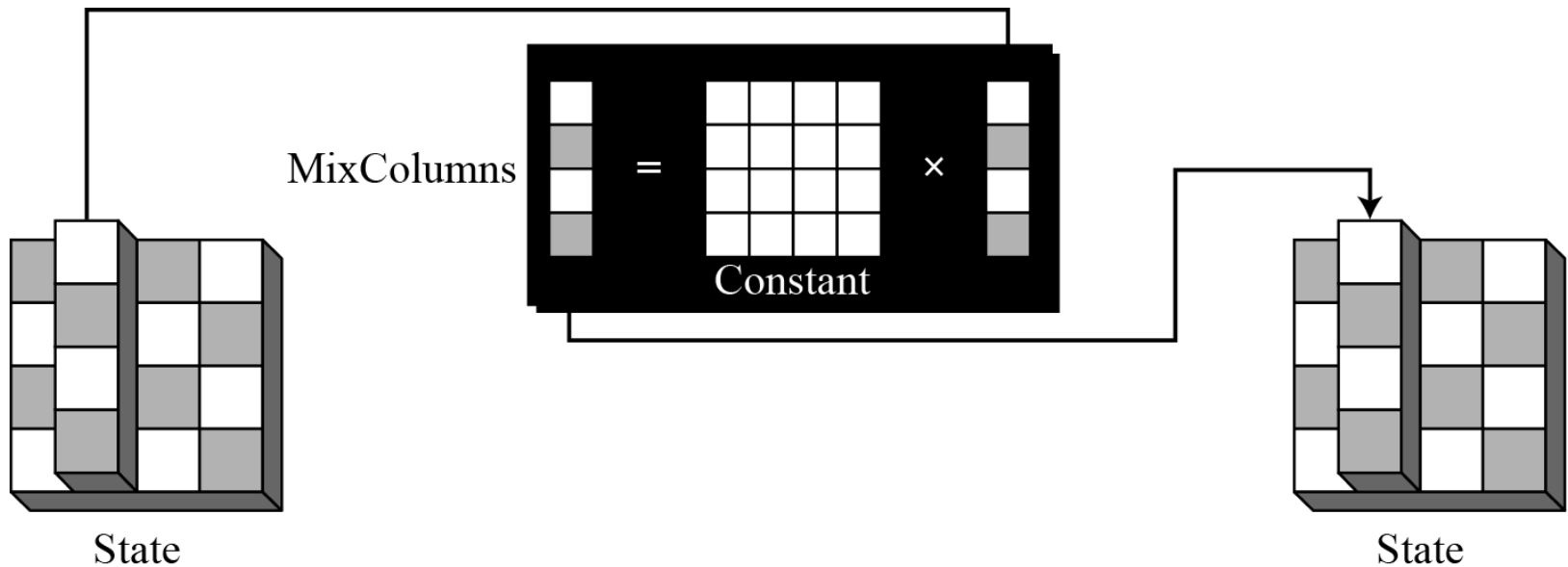
- mix bytes to provide diffusion at the bit level

$$\begin{array}{l} ax + by + cz + dt \\ ex + fy + gz + ht \\ ix + jy + kz + lt \\ mx + ny + oz + pt \end{array} \begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}$$

New matrix **Constant matrix** Old matrix

AES MixColumns

- operates at the column level
- transforms each column of the state to a new column via arithmetic on $\text{GF}(2^8)$
 - $\text{GF}(2^8)$ has 30 irreducible polynomials, Rijndael uses the first one $x^8+x^4+x^3+x+1$

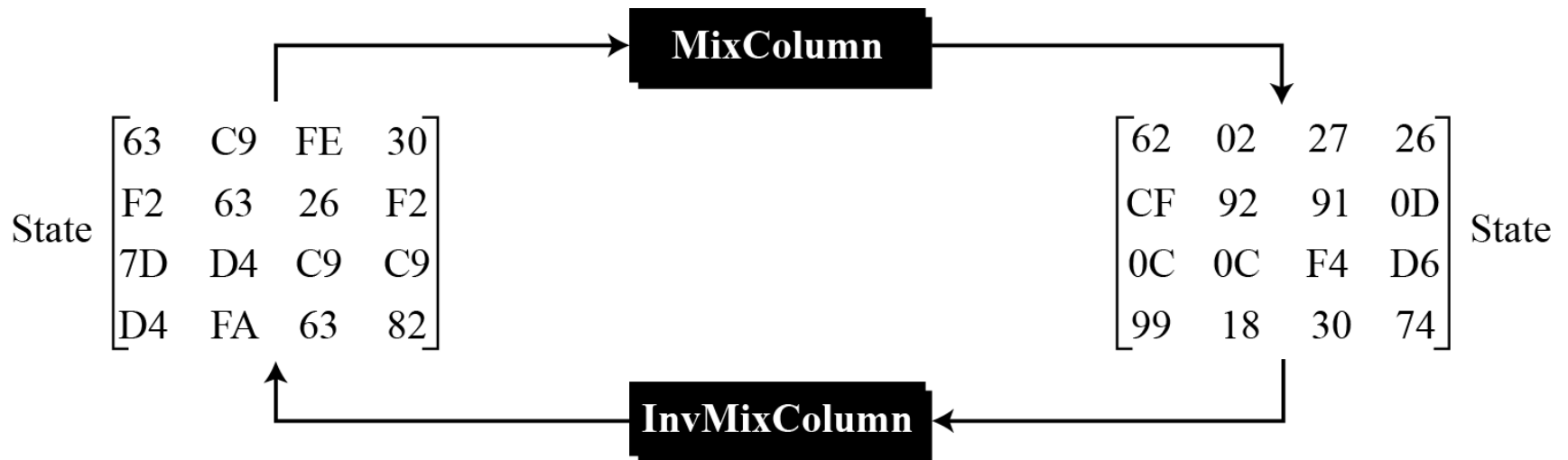


MixColumns & InvMixColumns

- The constant C and its inverse used in AES:

$$\begin{array}{ccc} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} & \xleftrightarrow{\text{Inverse}} & \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \\ C & & C^{-1} \end{array}$$

Example



Example

- First entry in result (all in Hex):

$$\begin{aligned} & [02\ 03\ 01\ 01] * [63\ F2\ 7D\ D4]' \\ &= 02*63 \oplus 03*F2 \oplus 01*7D \oplus 01*D4 \end{aligned}$$

- Multiplication by 2:

- Left shift by 1 bit, padding the least significant bit (LSB) with 0
- If the most significant bit was 1 before shift, XOR the result with $1B_{16}$ or 00011011_2

- Multiplication by 3:

- Write $3*x = (10 \oplus 01) * x = 2*x \oplus x$

- Therefore, the above formula becomes:

$$02*63 \oplus (02*F2 \oplus F2) \oplus 01*7D \oplus 01*D4$$

Example

- $02 * 63$
 - $63_{16} = 0110\ 0011_2$
 - Left shift by 1 bit: $1100\ 0110_2 = C6_{16}$
- $02 * F2$
 - $F2_{16} = 1111\ 0010_2$
 - MSB = 1, need to add $1B_{16}$ to the left shift result
 - Left shift by 1 bit: $1110\ 0100_2 = E4_{16}$
 - Result will be $E4_{16} \oplus 1B_{16}$
- Therefore
$$\begin{aligned} & 02 * 63 \oplus (02 * F2 \oplus F2) \oplus 01 * 7D \oplus 01 * D4 \\ &= C6 \oplus (E4 \oplus 1B \oplus F2) \oplus 7D \oplus D4 \\ &= 62 \end{aligned}$$

Online tool: <http://www.jdejong.net/tools/bitwisecalculator.php>

Alternative Method

- All the matrix entries are elements of the Galois field $GF(2^8)$, using irreducible polynomial $x^8+x^4+x^3+x+1$
- $02*63$
 - $02 = 0000\ 0010 \rightarrow x$
 - $63 = 0110\ 0011 \rightarrow x^6 + x^5 + x + 1$
 - $02*63 \rightarrow x * (x^6 + x^5 + x + 1) = x^7 + x^6 + x^2 + x$
 - It corresponds to **1100 0110 = C6**
- $03*F2$
 - $03 = 0000\ 0011 \rightarrow x + 1$
 - $F2 = 1111\ 0010 \rightarrow x^7 + x^6 + x^5 + x^4 + x$
 - $03*F2 = (x + 1)*(x^7 + x^6 + x^5 + x^4 + x)$
 $= (x^8 + x^7 + x^6 + x^5 + x^2) + (x^7 + x^6 + x^5 + x^4 + x)$
 $= x^8 + x^4 + x^2 + x$

Alternative Method

- Divide the result by $x^8+x^4+x^3+x+1$

$$\begin{array}{r} \frac{1}{x^8+x^4+x^3+x+1} \sqrt{x^8+x^4+x^2+x} \quad (Q) \\ \underline{x^8+x^4+x^3+x+1} \\ x^3+x^2+1 \quad (R) \end{array}$$

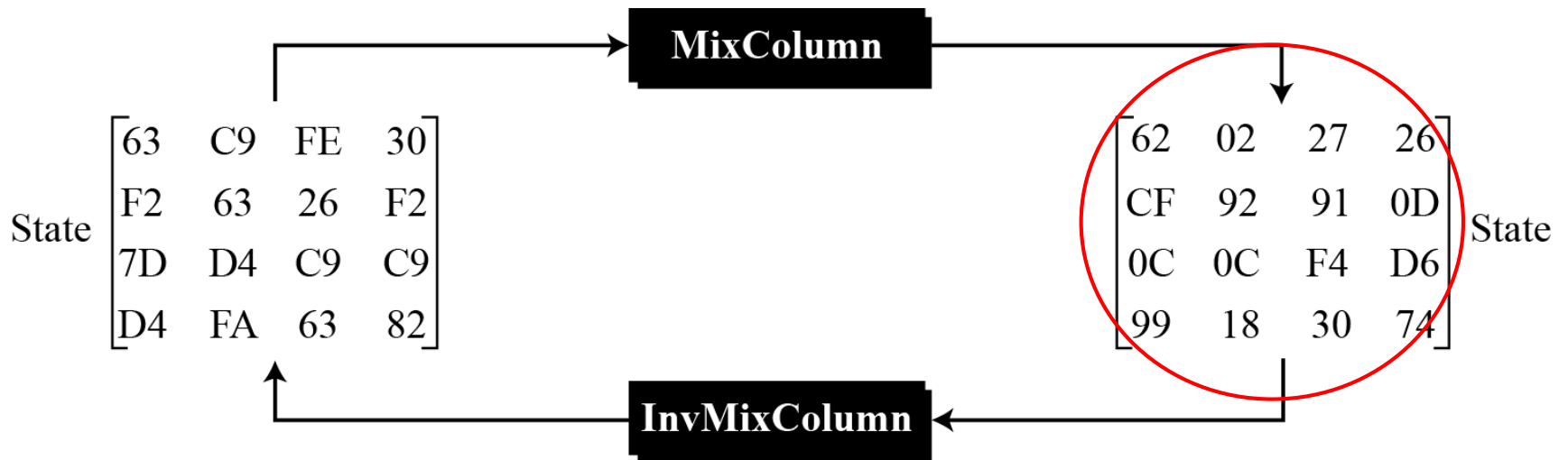
- Remainder is the result:
 - $03 * F2 = 0000\ 1101 = 0D$

- Therefore

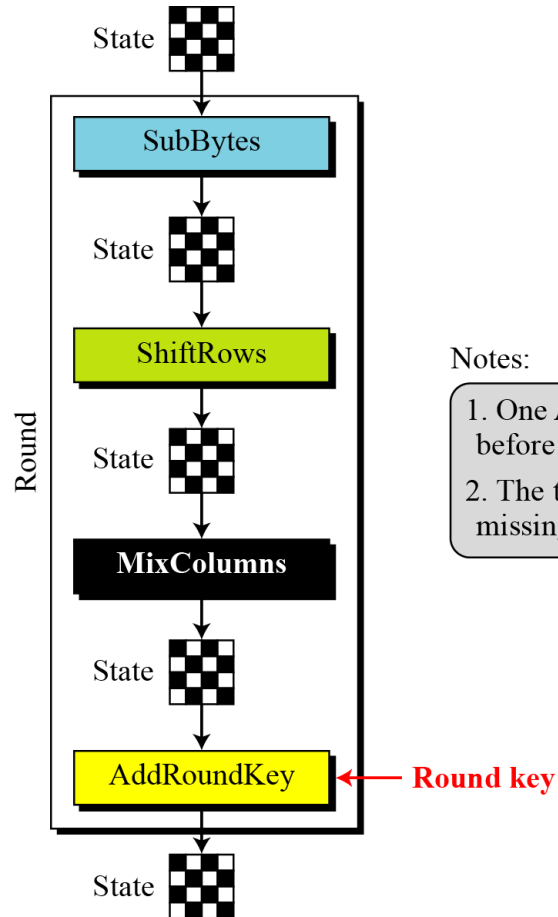
$$\begin{aligned} & 02*63 \oplus 03*F2 \oplus 01*7D \oplus 01*D4 \\ = & \text{C6} \quad \oplus \text{0D} \quad \oplus 7D \quad \oplus D4 \\ = & 62 \end{aligned}$$

Exercise

- Verify the MixColumn result



Operation in Each Round

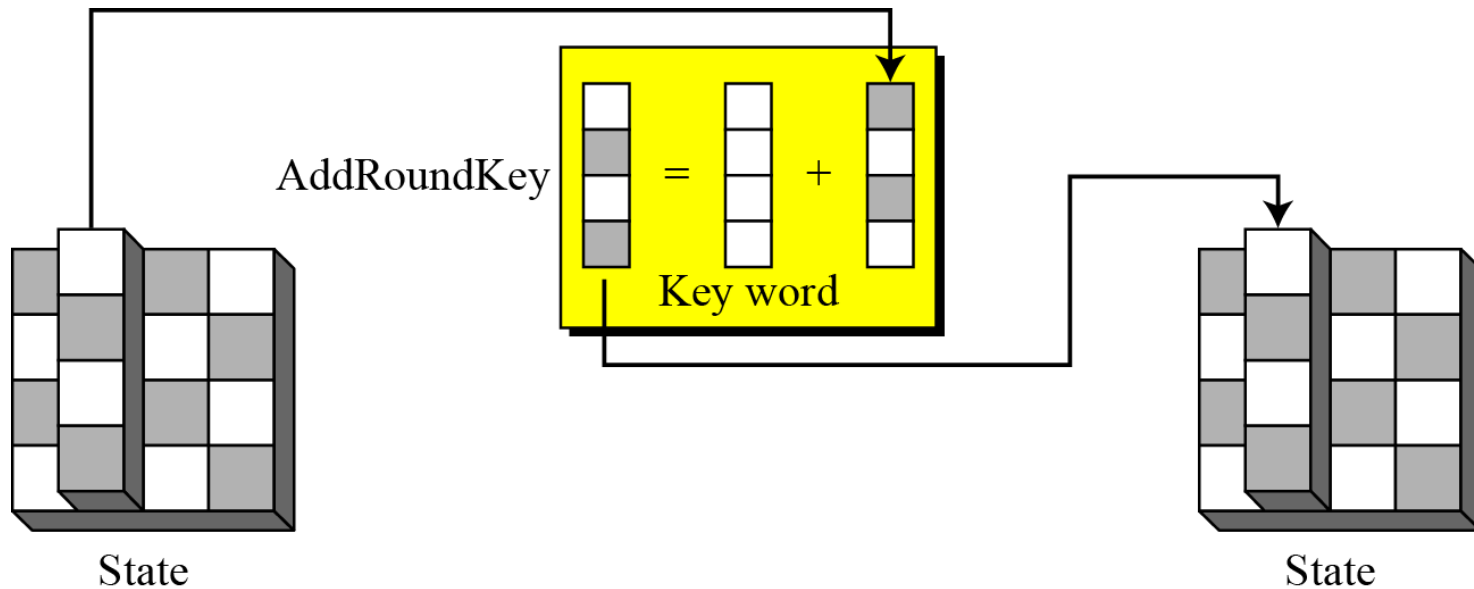


Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

AddRoundKey

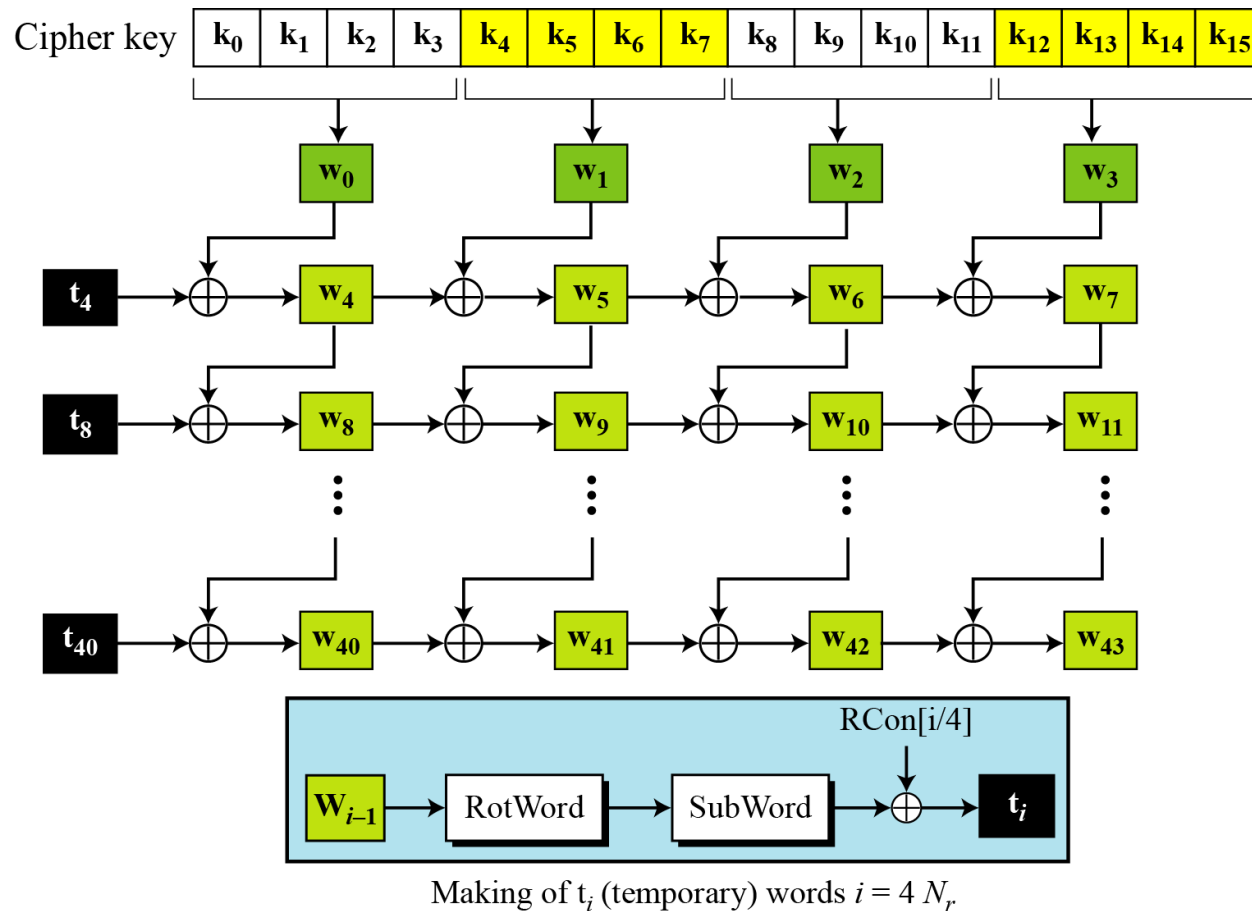
- adds (i.e. xor) a round key word with **each state column** matrix



How can we get the round key?

- AES uses a key-expansion process
- If the number of rounds is N_r , the key-expansion routine creates $N_r + 1$ 128-bit round keys from one single 128-bit cipher key
 - 1 word = 4 bytes = 32 bits
 - 1 round key = 4 words

AES-128 bit Key Expansion



AES-128 bit Key Expansion

- Words for each round

<i>Round</i>	<i>Words</i>			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
...	...			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

RotWord & SubWord

- RotWord: Perform simple cyclic permutation of a word

$$[a_0, a_1, a_2, a_3] \rightarrow [a_1, a_2, a_3, a_0]$$

- SubWord: applies the S-box value used in SubBytes to each of the 4 bytes in the argument

Rcon Constants

<i>Round</i>	<i>Constant (RCon)</i>	<i>Round</i>	<i>Constant (RCon)</i>
1	(<u>01</u> 00 00 00) ₁₆	6	(<u>20</u> 00 00 00) ₁₆
2	(<u>02</u> 00 00 00) ₁₆	7	(<u>40</u> 00 00 00) ₁₆
3	(<u>04</u> 00 00 00) ₁₆	8	(<u>80</u> 00 00 00) ₁₆
4	(<u>08</u> 00 00 00) ₁₆	9	(<u>1B</u> 00 00 00) ₁₆
5	(<u>10</u> 00 00 00) ₁₆	10	(<u>36</u> 00 00 00) ₁₆

SubBytes Transformation Table

- The S-box used is derived from the multiplicative inverse over $GF(2^8)$ with an invertible affine transformation

		<i>y</i>															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
<i>x</i>	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
	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

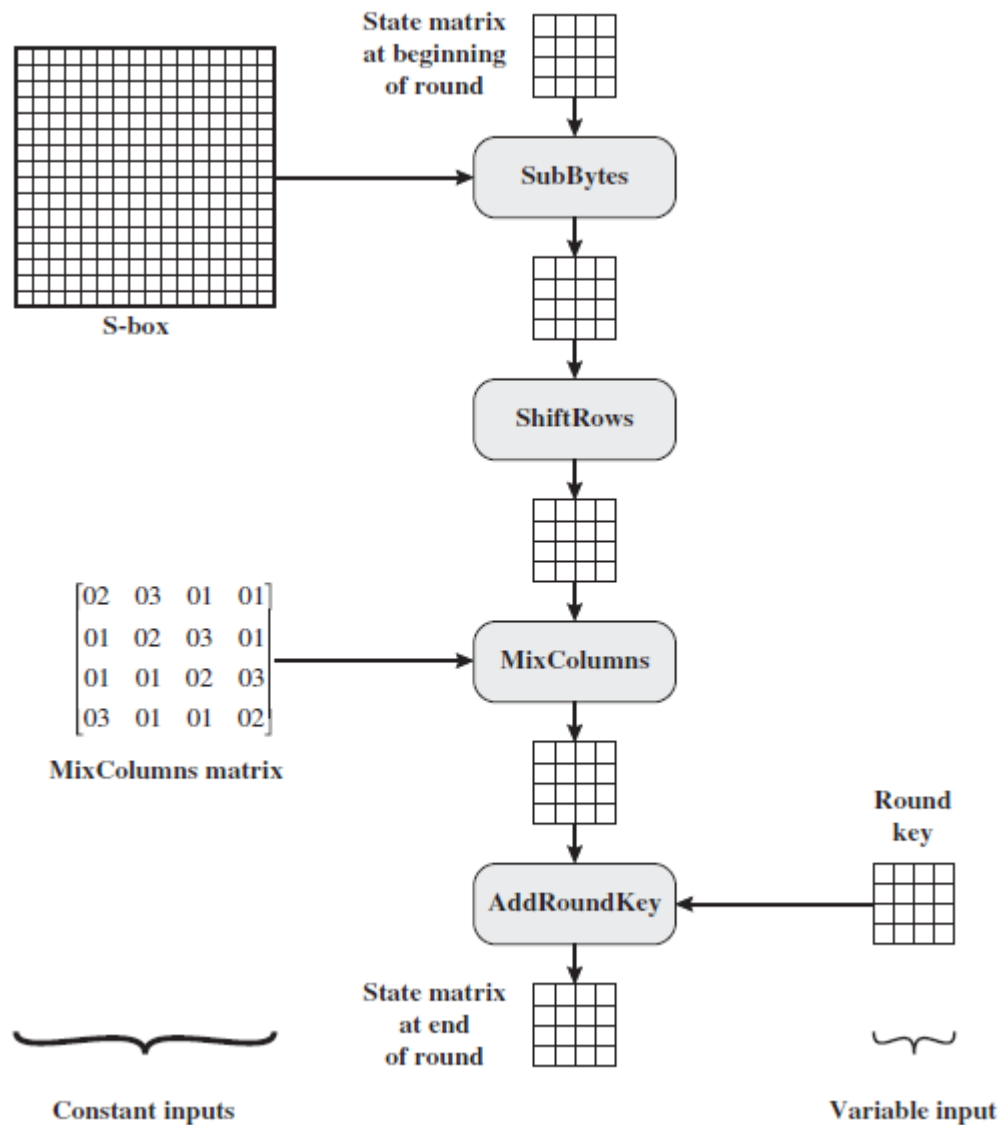
Example

- Given the cipher key (24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87)₁₆
 - $w_0 = 24\ 75\ A2\ B3$
 - $w_1 = 34\ 75\ 56\ 88$
 - $w_2 = 31\ E2\ 12\ 00$
 - $w_3 = 13\ AA\ 54\ 87$
- Calculate t_4
 - Input: $w_3 = 13\ AA\ 54\ 87$
 - RotWord $\rightarrow AA\ 54\ 87\ 13$
 - SubWord: Subbytes (AA 54 87 13) $\rightarrow$ (AC 20 17 7D)
 - Rcon (4/4) = Rcon(1) = 01 00 00 00
 - (AC 20 17 7D) $\oplus$ (01 00 00 00) = AD 20 17 7D

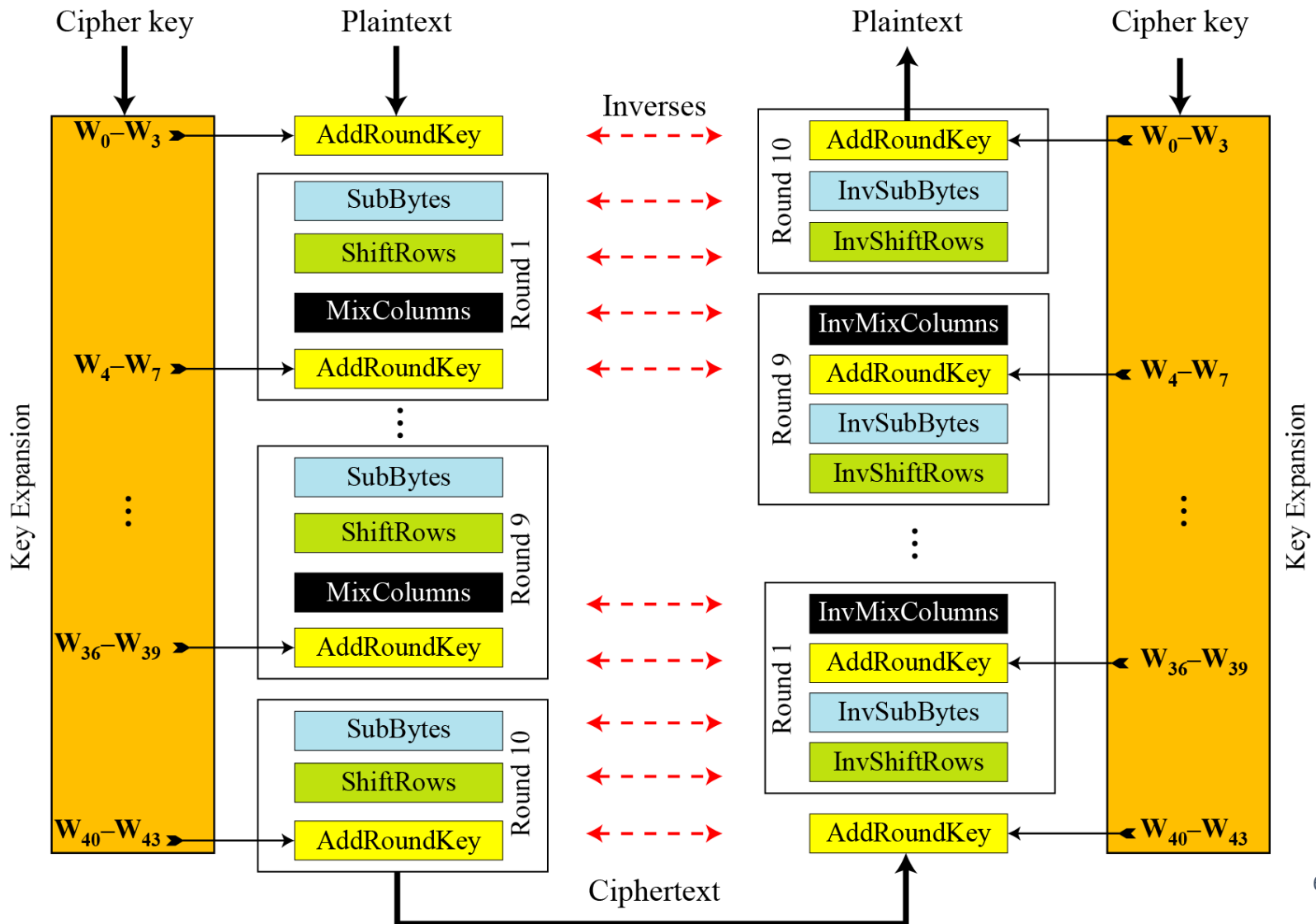
Example

- ▶ $w_4 = t_4 \oplus w_0$
= (AD 20 17 7D) $\oplus$ (24 75 A2 B3)
= (89 55 B5 CE)
- ▶ $w_5 = w_4 \oplus w_1$
= (89 55 B5 CE) $\oplus$ (34 75 56 88)
= (BD 20 E3 46)
- ▶ $w_6 = w_5 \oplus w_2$
= (BD 20 E3 46) $\oplus$ (31 E2 12 00)
= (8C C2 F1 46)
- ▶ $w_7 = w_6 \oplus w_3$
= (8C C2 F1 46) $\oplus$ (13 AA 54 87)
= (9F 68 A5 C1)
- ▶ ...

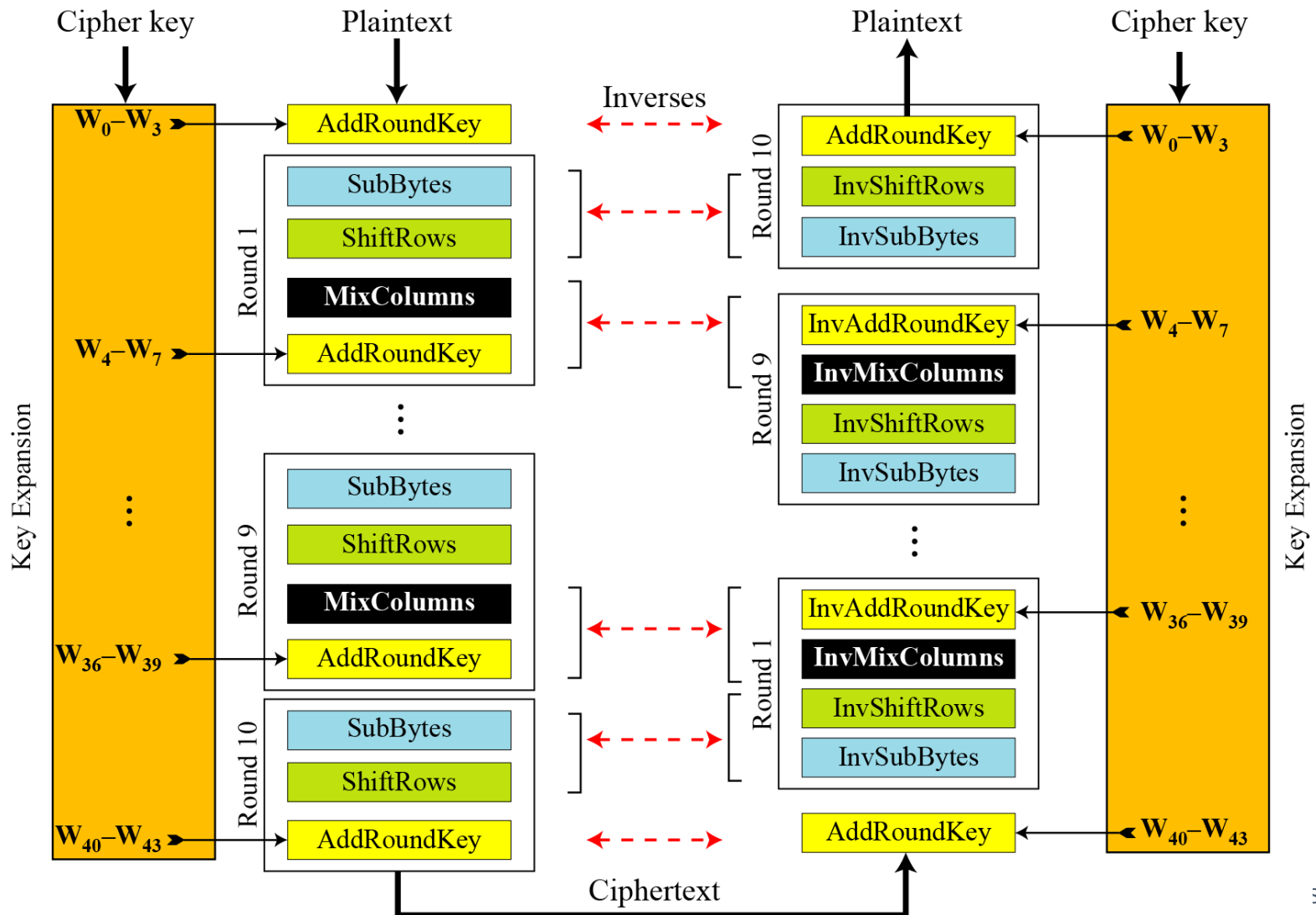
Summary of Round Operation



Overall flow – Original Design



Overall flow – Alternate Design



Rijndael in C

```
/* BC byte count   rk   round key   S   sbox*/
/* plaintext in a */
KeyAddition(a,rk[0],BC);

/* ROUNDS-1 ordinary rounds */
for(r = 1; r < ROUNDS; r++) {
    Substitution(a,S,BC);
    ShiftRow(a,0,BC);
    MixColumn(a,BC);
    KeyAddition(a,rk[r],BC);
}

/* Last round is special: there is no MixColumn */
Substitution(a,S,BC);
ShiftRow(a,0,BC);
KeyAddition(a,rk[ROUNDS],BC);
```

Rijndael in hardware

- 8-bit processor (smartcard)
 - addroundkey is a bitwise XOR
 - shiftrows is byte rotates
 - subbytes is table lookup (256 byte table)
 - mixcolumns is XORs and a table lookup (256 byte table)
- 32-bit processor
 - Operate on 32-bit words rather than bytes
 - 4 table lookups and four XORs per column per round (fast)
 - Need 4 256-word (1024 byte) tables
- Can trade off memory space for computation time

Exercise

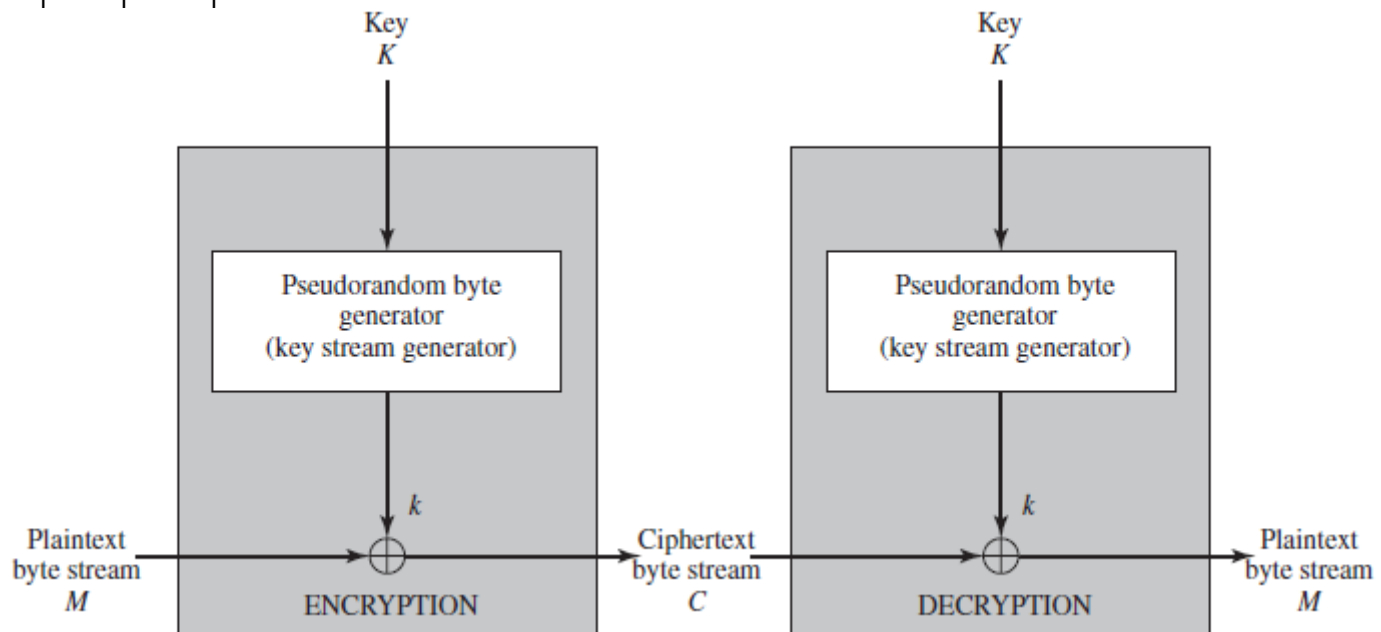
- Google an AES applet / flash to follow the operation of AES encryption and decryption
 - E.g.
http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf
- Write / Google a source code of AES in your favorite programming language

Lecture 4: Symmetric Key Ciphers

- DES
- AES
- Stream Cipher
- Mode of Operations
- Considerations

Stream ciphers

- A symmetric key cipher where plaintext (in unit of byte / bits at a time) xor with a pseudorandom key stream
 - Encrypt:
 - XOR plaintext with keystream $P_i \oplus K_i \rightarrow C_i$
 - Decrypt
 - $C_i \oplus K_i \rightarrow P_i$



Why Stream ciphers

- efficient in hardware
- much theoretical analysis (LFSR's)
- faster than block ciphers

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

- examples
 - RC4 (used in 802.11, PPTP, etc)
 - A5/1 (used in GSM cell phone etc)
 - E0 for Bluetooth protocol

RC4

- RC4 is a byte-oriented stream cipher
 - based on the concept of a state
 - a state begins as an array containing the values 0-255 in order, swaps the elements around based on the key value
 - after the initialization, the original key can then be discarded
 - because RC4 only uses the state to create a pseudo-random keystream
 - as a result, the original key doesn't need to be kept in memory
- RC4 is a synchronous stream cipher
 - the stream of pseudo-random digits are generated independently of the plaintext and ciphertext
- 1 byte (8 bits) of a plaintext xor's with 1 byte of key to produce 1 byte of the ciphertext

Video Example

- <https://www.youtube.com/watch?v=KM-xZYZXEIk>

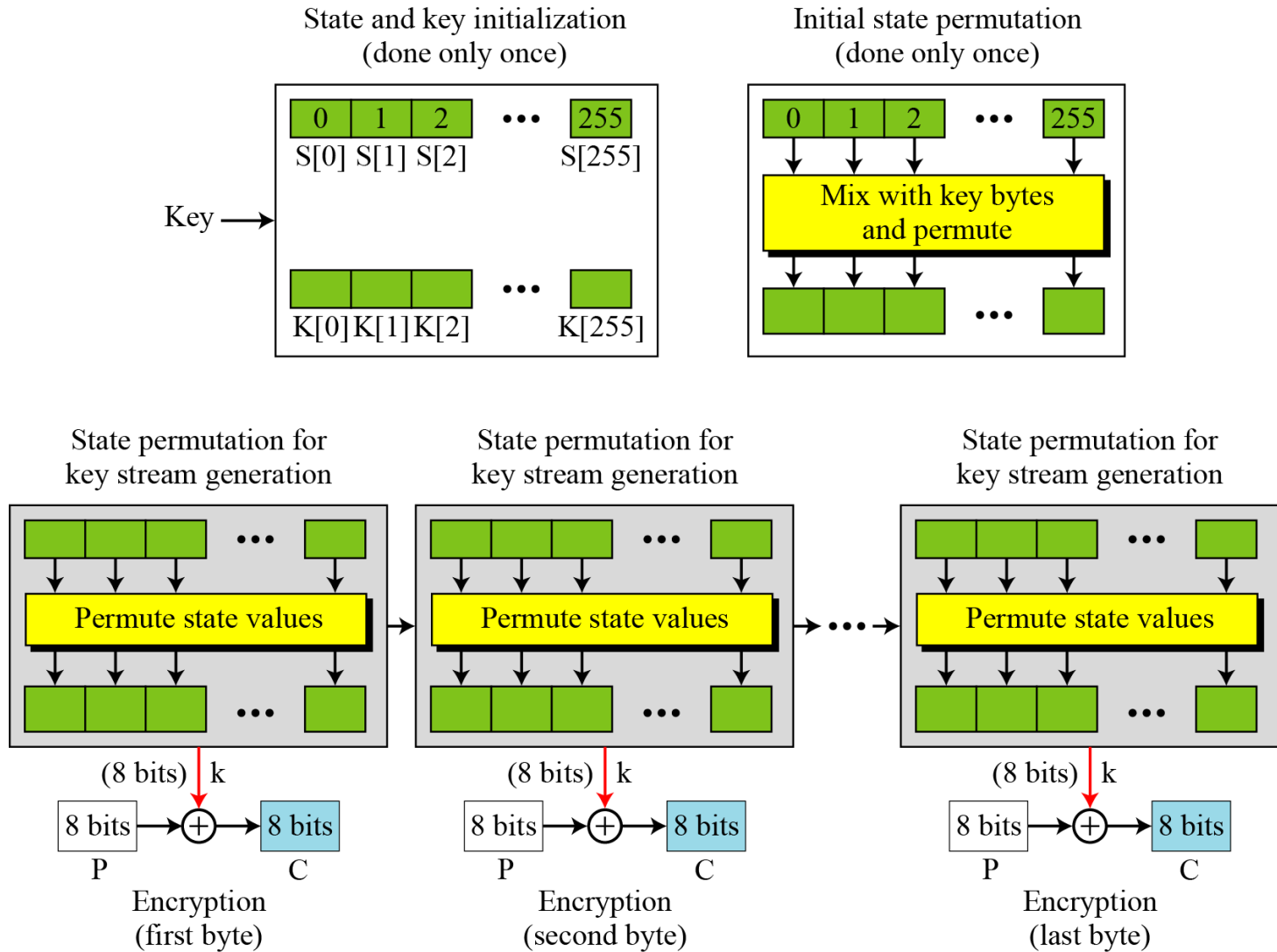
The screenshot displays a video player interface with a presentation on the RC4 algorithm. The presentation is titled "PRGA" and "RC4". It shows a "Permuted State Vector, S" with values: 28, 81, 30, 60, 31, 225, ..., 108. Below it, a "Pseudo-random keystream, K" is shown with values: 149, 96, 252, ..., 131. A red arrow points to the value 31 in the state vector, and a black box labeled "SWAP" highlights the values 31 and 131 in the keystream. The pseudocode section shows the following code:

```
Pseudocode
i = 0
j = 0
plen = length(Plaintext)
while (plen > 0)
  i = (i + 1) mod 256
  j = (j + S[i]) mod 256
  Swap(S[i], S[j])
  k = S[S[i] + S[j]] mod 256
  Output k
  plen = plen - 1
endwhile
```

Below the pseudocode, the state of the algorithm is shown: index pointer, i = 4; index pointer, j = 83; counter, plen = 9.

The video player interface includes a progress bar at the bottom showing 2:52 / 4:46, a page indicator 9 / 11, and a duration of 142 Seconds.

RC4



RC4

- trademark of RSA
- fast, and used in several products (WEP, SSL, etc)
- Source code: <http://blog.markloiseau.com/2012/07/rc4-tutorial/>

```
//fill states with 0 to 255 and take key K
j=0
for i = 0 to 255      // rearrange states according to key
    j = (j + Si + Ki) mod 256
    swap Si and Sj

//keystream generation
i,j=0
while (true)
    i = (i+1) mod 256
    j = (j+ Si) mod 256
    swap Si and Sj      // rearrange states
    t = (Si + Sj) mod 256
    b = St
```

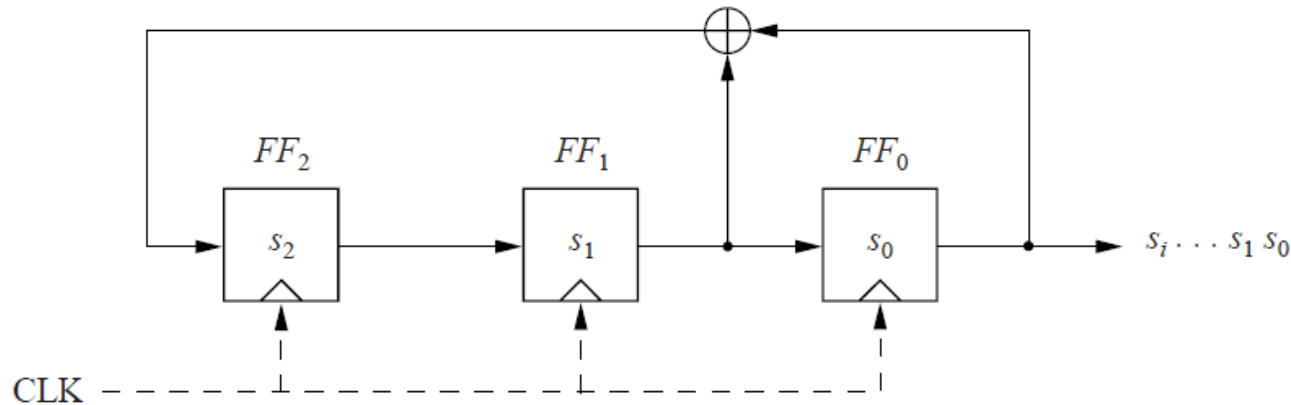
XOR plain/cipher text byte with *b*

LFSR

- Linear feedback shift registers (LFSRs)
 - consists of clocked storage elements (flip-flops) and a feedback path
 - LFSR with m flip-flops is said to be of degree m
- A common building blocks for stream cipher
 - efficient in hardware (shift XOR)
 - readily analysed mathematically

Polynomial arithmetic over $GF(2^n)$ is used in AES (Rijndael) and LFSR's

Example



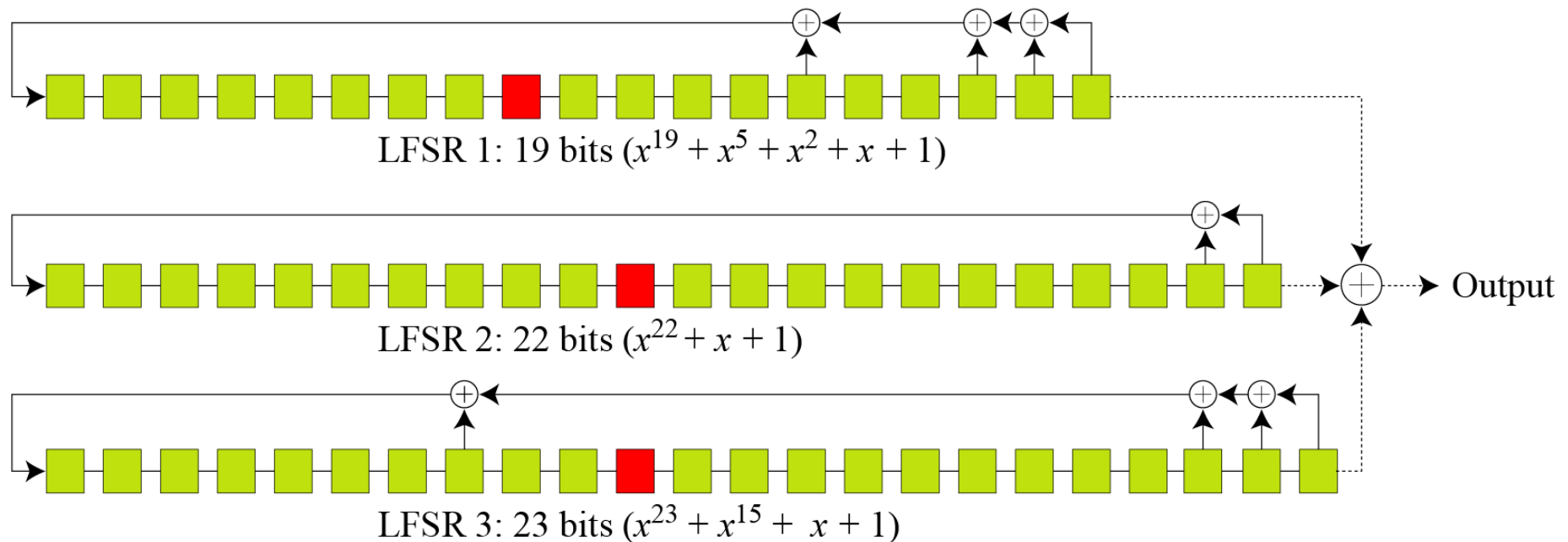
- repeat after clock cycle 6 → the LFSR output has period of length 7 and has the form: **0010111** 0010111 0010111

clk	FF_2	FF_1	$FF_0 = s_i$
0	<u>1</u>	<u>0</u>	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	<u>1</u>	<u>0</u>	0
8	0	1	0

A5/1

- A5/1 uses three LFSRs with 19, 22, and 23 bits
 - Clocking bit: bit 8, bit 10 and bit 10 (marked in RED below)

Note: The three red boxes are used in the majority function



A5/1

- Set all bits in the three LFSRs to 0
- The bit streams created from the key generator are buffered to form a 228-bit key that is xor'ed with the plaintext frame to create the ciphertext frame
- Encryption/decryption is done one frame at a time.

Problems with Stream Cipher

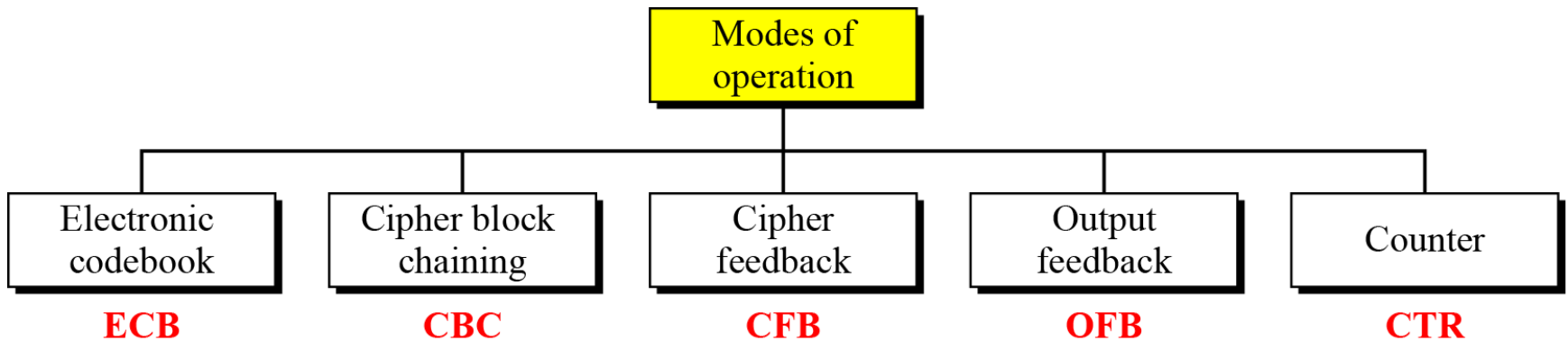
- If keystreams in stream cipher is re-used, Known-plaintext attack is possible
 - $(M_1 \oplus \text{keystream}) \oplus (M_2 \oplus \text{keystream}) = M_1 \oplus M_2$
 - If attacker knows M_1 , then M_2 can also be recovered
- Short key length might make brute-force attack possible
 - E.g. The 40-bit exportable version of RC4 has been broken by brute force!

Lecture 4: Symmetric Key Ciphers

- DES
- AES
- Stream Cipher
- Mode of Operations
- Considerations

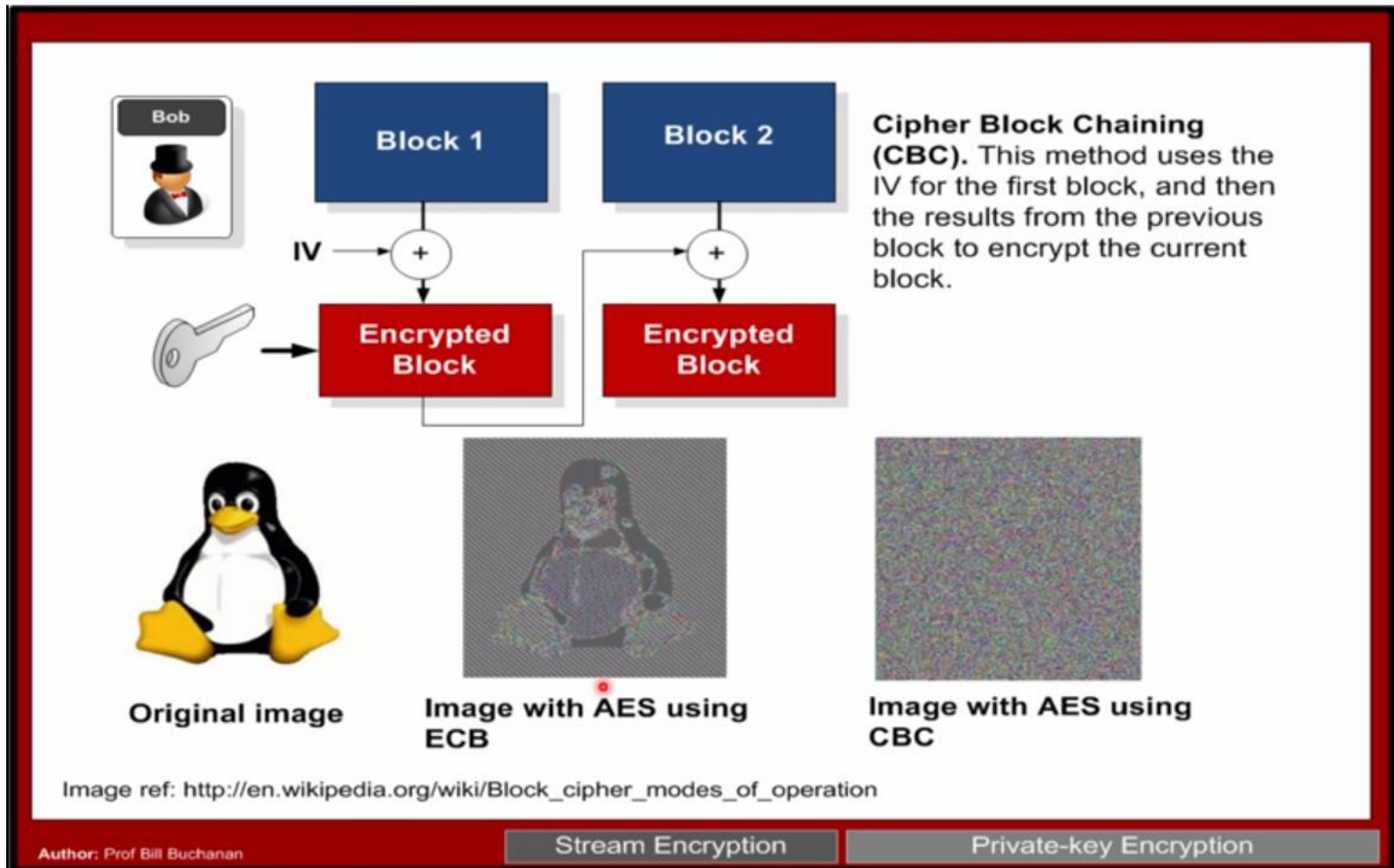
Modes of Operation

- Electronic Codebook (ECB) Mode
- Cipher Block Chaining (CBC) Mode
- Cipher Feedback (CFB) Mode
- Output Feedback (OFB) Mode
- Counter (CTR) Mode



Example

- <https://www.youtube.com/watch?v=uPiqyQOMH1E>



General Description

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Comparison

<i>Operation Mode</i>	<i>Description</i>	<i>Type of Result</i>	<i>Data Unit Size</i>
ECB	Each n -bit block is encrypted independently with the same cipher key.	Block cipher	n
CBC	Same as ECB, but each block is first exclusive-ored with the previous ciphertext.	Block cipher	n
CFB	Each r -bit block is exclusive-ored with an r -bit key, which is part of previous cipher text	Stream cipher	$r \leq n$
OFB	Same as CFB, but the shift register is updated by the previous r -bit key.	Stream cipher	$r \leq n$
CTR	Same as OFB, but a counter is used instead of a shift register.	Stream cipher	n

ECB

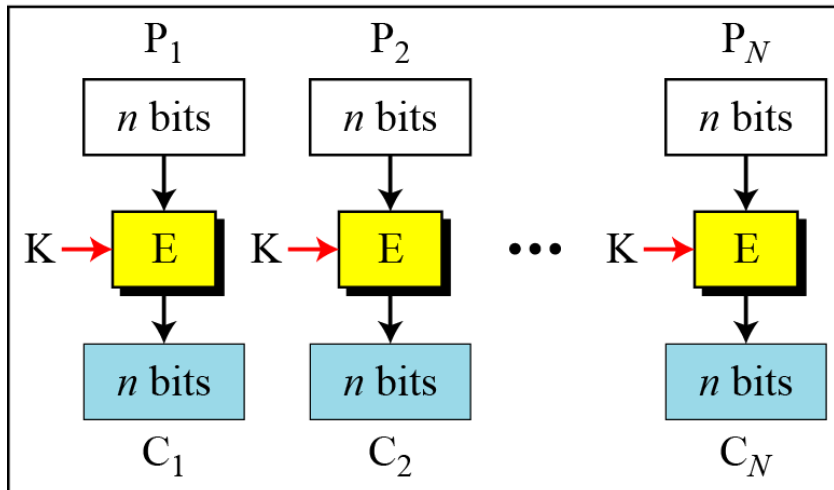
- Message are divided into chunks of independent blocks
 - If the same n -bit block of plaintext appears more than once in the message, it always produces the same ciphertext.
 - Appropriate mode to transmit a DES or AES key securely
 - For lengthy messages, the ECB mode may not be secure.

E: Encryption

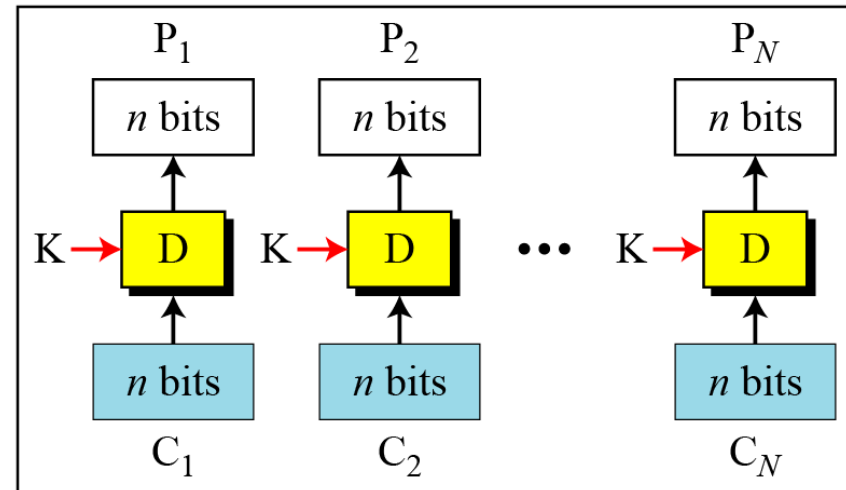
D: Decryption

P_i : Plaintext block i C_i : Ciphertext block i

K: Secret key



Encryption



Decryption

CBC

- Each plaintext block is xor'ed with the previous ciphertext block before being encrypted
 - IV should be known by sender and receiver

E: Encryption

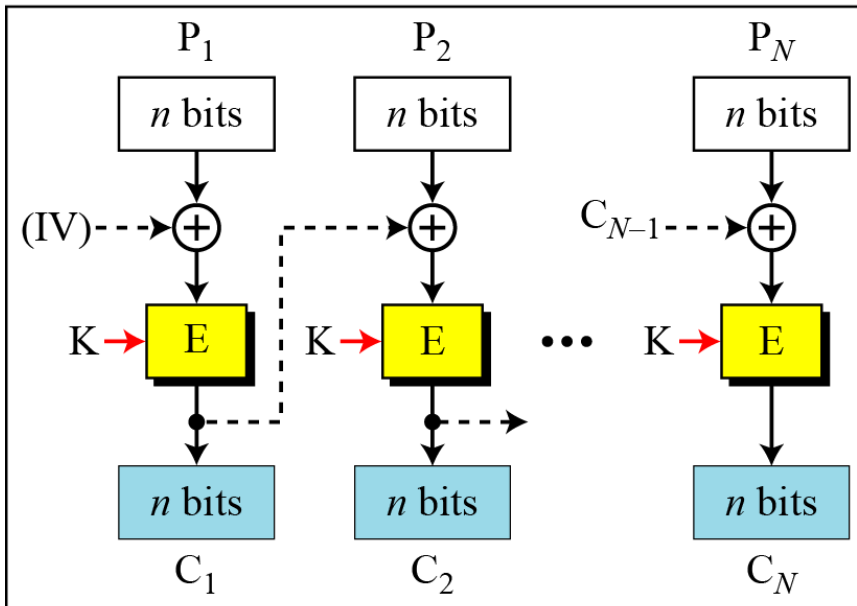
D: Decryption

P_i : Plaintext block i

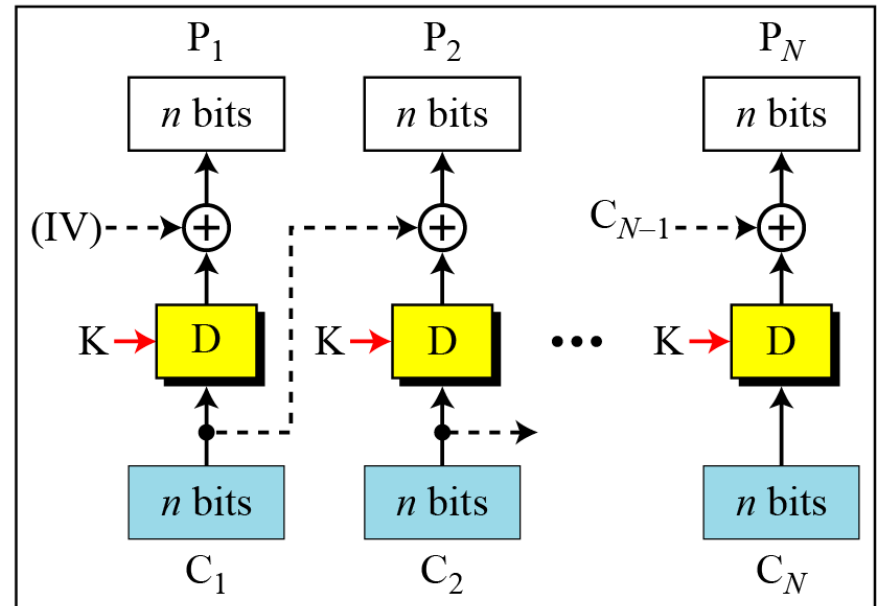
C_i : Ciphertext block i

K: Secret key

IV: Initial vector (C_0)



Encryption

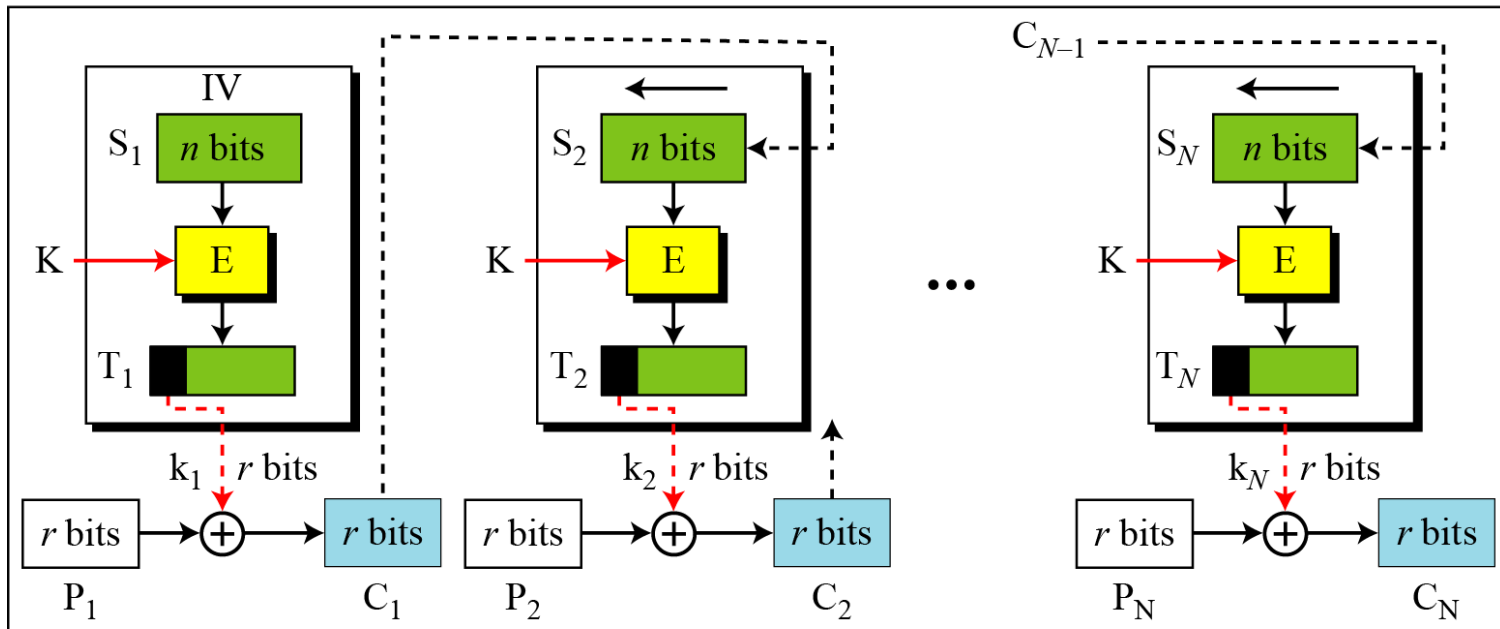


Decryption

CFB

- makes a block cipher into an asynchronous stream cipher (in which the keystream depends on both the key and the ciphertext)

E : Encryption D : Decryption S_i : Shift register
 P_i : Plaintext block i C_i : Ciphertext block i T_i : Temporary register
 K: Secret key IV: Initial vector (S_1)



Encryption

OFB

- Makes a block cipher into a synchronous stream cipher
 - Each bit in the ciphertext is independent of the previous bit or bits

E : Encryption

P_i : Plaintext block i

K: Secret key

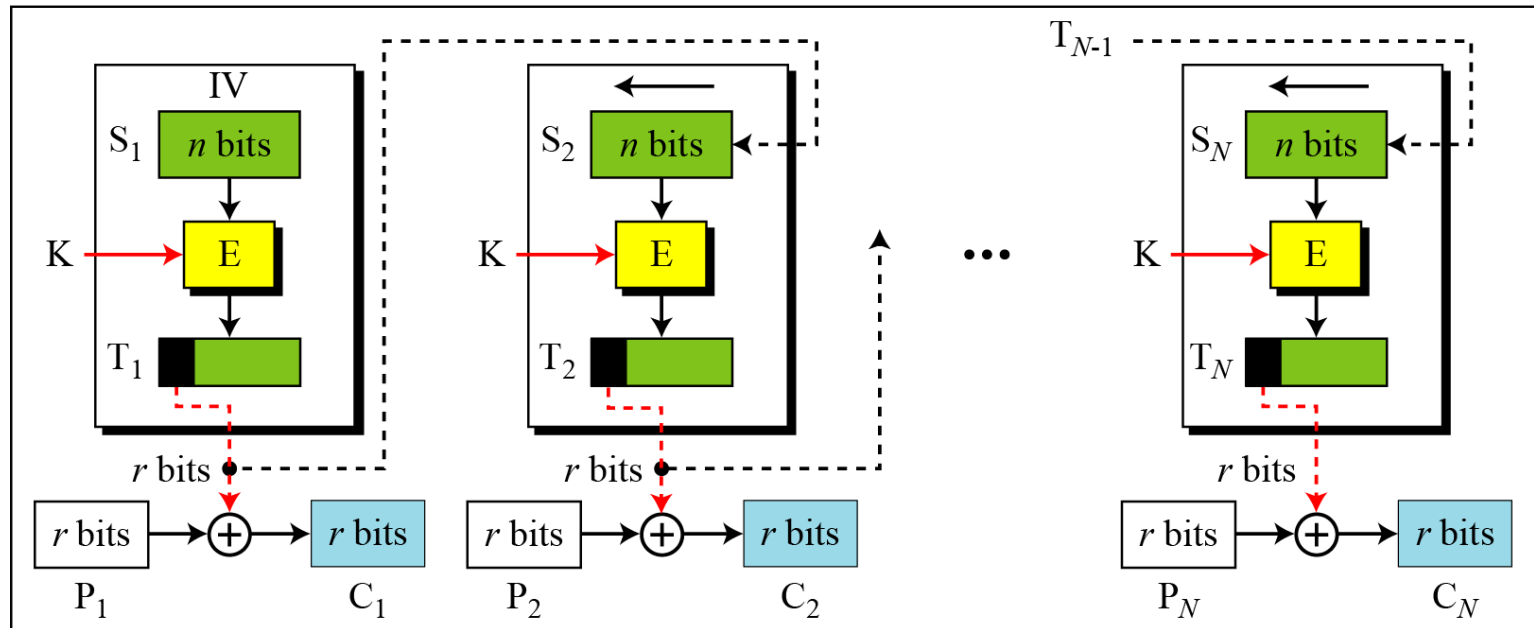
D : Decryption

C_i : Ciphertext block i

IV: Initial vector (S_1)

S_i : Shift register

T_i : Temporary register



Encryption

CTR

- No feedback, and the pseudo-randomness in the key stream is achieved using a counter

E : Encryption

P_i : Plaintext block i

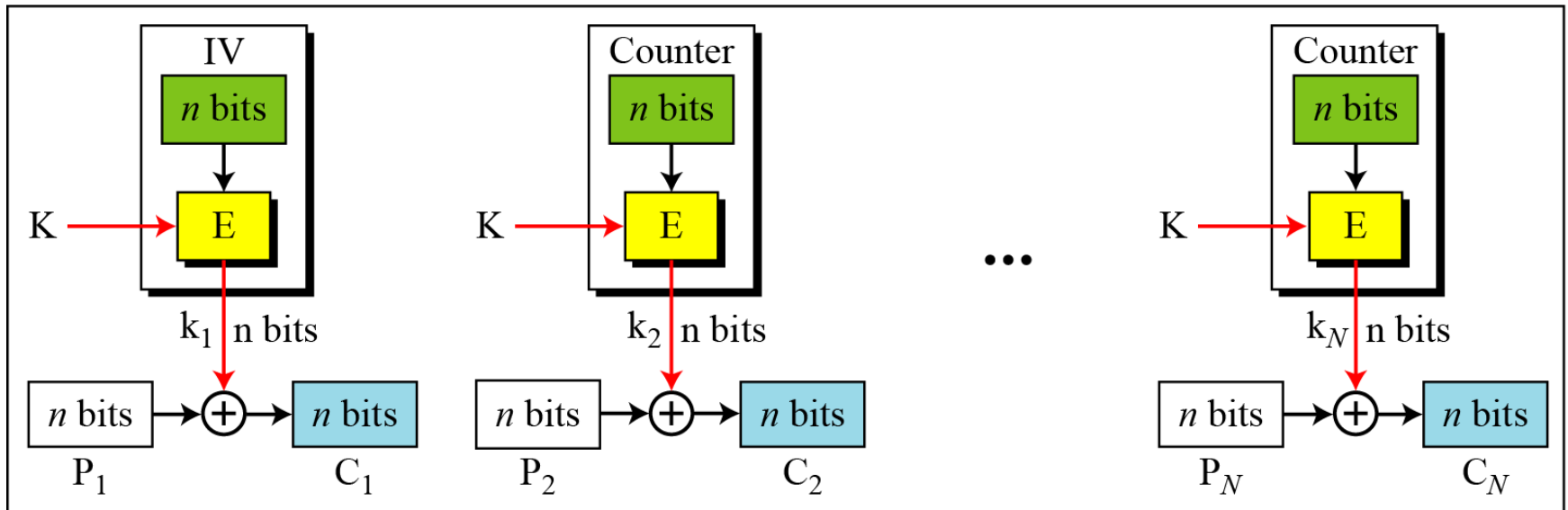
K : Secret key

IV: Initialization vector

C_i : Ciphertext block i

k_i : Encryption key i

The counter is incremented for each block.



Encryption

Lecture 4: Symmetric Key Ciphers

- DES
- AES
- Mode of Operations
- Stream Cipher
- Considerations

Security Management

- Focus on establishing policy for using approved symmetric algorithms in an organisation?
 - DES? AES?
 - Justification? E.g. follow the best practices in the industry, recommendation from NIST, ...
 - Can we use the AES features in WinZip?
 - ...
- Also need to develop a practice to safeguard the exchange of the secret key used in symmetric algorithms
 - Can key be exchanged in the same email containing the encrypted data and the key?
 - Separate channel to exchange the keys, e.g. SMS, Whatsapp,
 - ...

Engineers / Programmers

- Write programs to implement the algorithms
- Build hardware to implement the algorithms
- Re-use reliable crypto packages (e.g. openssl) in applications or scripts
- Build your own toolbox
- ...

Mathematicians

- Understand the mathematics behind the algorithms (e.g. $\text{GF}(2^8)$)
- Study possible ways to do cryptanalysis on these algorithms
- Any improvements to the algorithms?
- ...

Limitations

Does symmetric key encryption provide:

- Privacy?
- Authenticity?
- Integrity?
- Non-repudiation?
- Availability?
- Virus protection?

Lecture 4: What you have learnt?

- DES
- AES
- Stream Cipher
- Mode of Operations
- Considerations

Your mission



- Understand the symmetric key (or secret-key) cryptography
- Build your crypto toolkits in your favorite programming language

