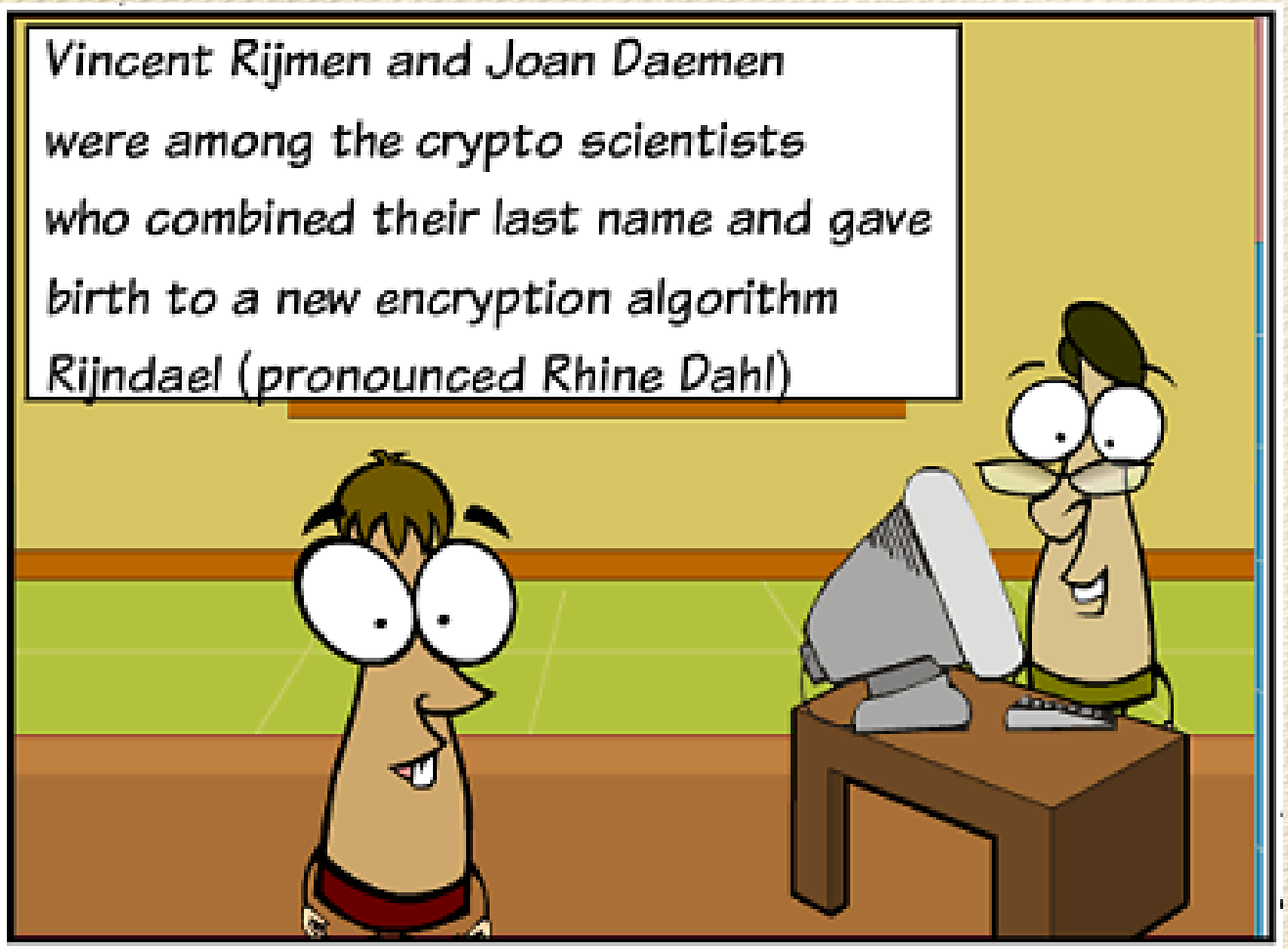
Advanced Encryption Standard (AES)



Mukesh Chinta
Assistant Professor, CSE
VRSEC

Development of AES

- In 1997, NIST issued a call for proposals for a new block cipher standard, to be referred to as the **Advanced Encryption Standard** or **AES**. A new standard was needed primarily because DES has a relatively small 56-bit key which was becoming vulnerable to brute force attacks. In addition, the DES was designed primarily for hardware and is relatively slow when implemented in software.

➢ *The selection would be a public process and the chosen algorithm and design details would be made freely available for public use.*

➢ *The block size should be 128 bits.*

➢ *The block cipher would be designed to offer variable key lengths of 128, 192 and 256 bits, to allow for future developments in exhaustive key search efforts.*

➢ *The block cipher had to operate at a faster speed than Triple DES across a number of different platforms.*

Selection Process of AES

- In the nine months that followed, fifteen different designs were created and submitted from several different countries. They were, in alphabetical order: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish.
- There are **three main categories** of criteria used by NIST to evaluate potential candidates.
  - ***Security:*** **Resistance to cryptanalysis, soundness of math, randomness of output, etc**
  - ***Cost:*** **Computational efficiency (speed), Memory requirements**
  - ***Algorithm/Implementation Characteristics:*** **Flexibility, hardware & software suitability, algorithm simplicity**
- In August 1999 they announced that they were narrowing the field from fifteen to five: MARS, RC6, Rijndael, Serpent, and Twofish. All five algorithms, commonly referred to as "AES finalists", were designed by cryptographers considered well-known and respected in the community.
- On October 2, 2000, NIST announced that **Rijndael** had been selected as the proposed AES and On November 26, 2001, AES was approved as FIPS PUB 197.
- The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: ***Dr. Joan Daemen*** and ***Dr. Vincent Rijmen***.

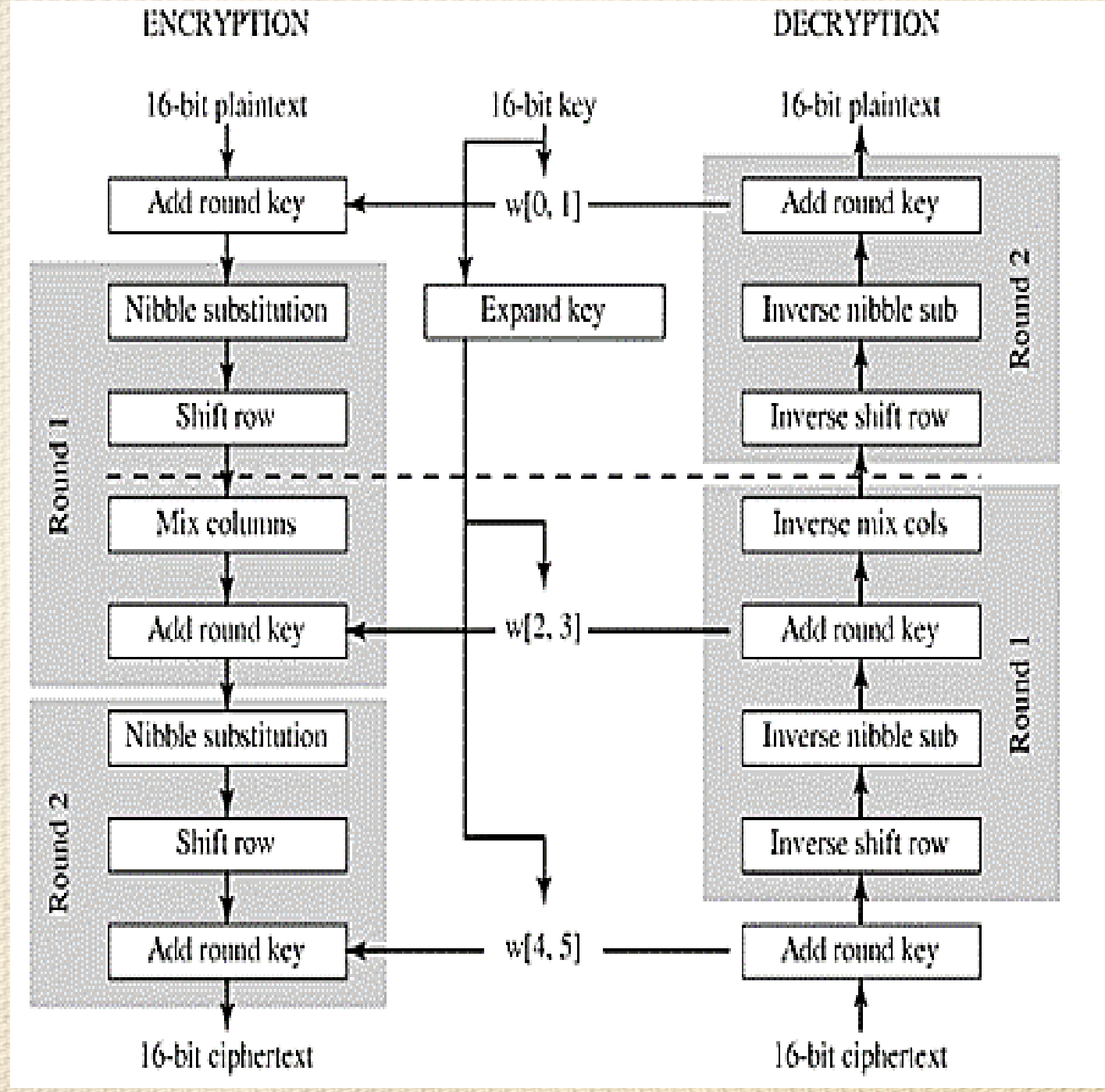
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Rijndael Cipher

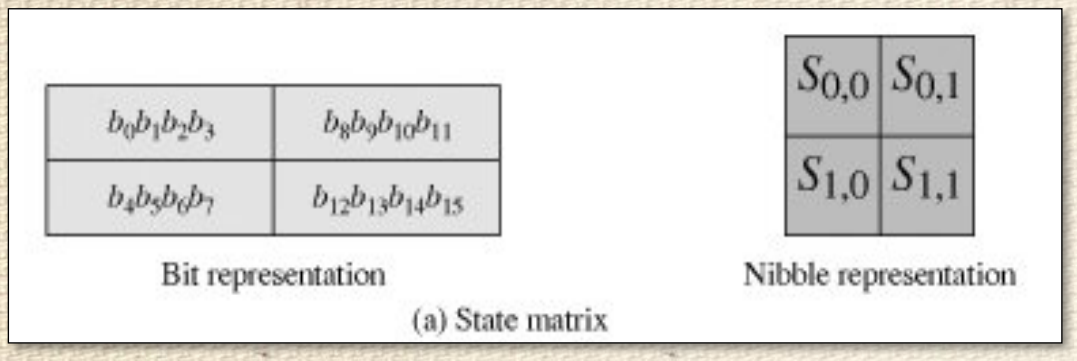
- Like DES, AES is a **symmetric block cipher**. However, AES is quite different from DES in a number of ways.
- The AES standard states that the algorithm can only accept a **block size of 128 bits** and a choice of **three keys - 128, 192, 256 bits**. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES-256 respectively.
- AES is **an iterative** rather than Feistel cipher. Here, the entire data block is processed in parallel during each round using substitutions and permutations.
- AES performs all its computations on **bytes** rather than bits. Hence, AES treats the 128 bits of a plaintext block as **16 bytes**. These 16 bytes are arranged in four columns and four rows for processing as a matrix.
- Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses **10 rounds for 128-bit keys**, **12 rounds for 192-bit keys** and **14 rounds for 256-bit keys**. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

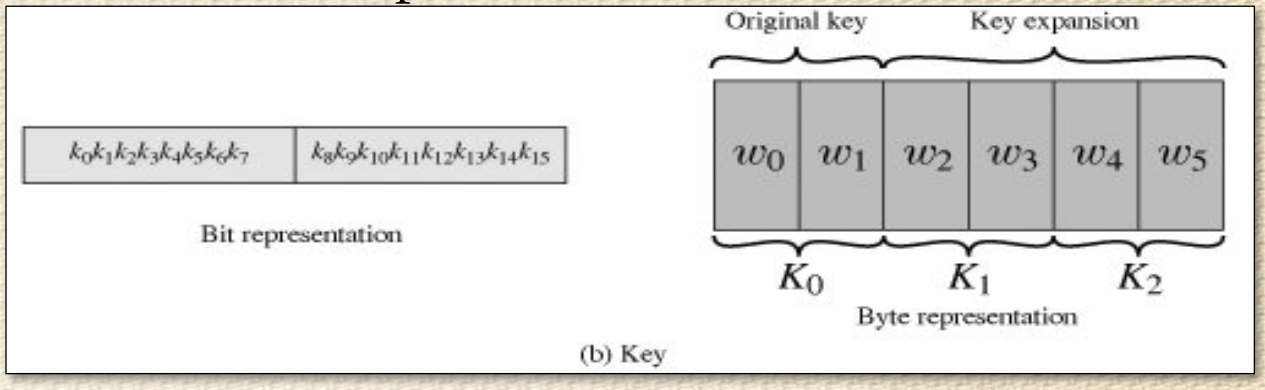
Simplified AES

- S-AES is to AES as S-DES is to DES. In fact, the structure of S-AES is exactly the same as AES. The differences are in the key size (16 bits), the block size (16 bits) and the number of rounds (2 rounds)
- The encryption algorithm involves the use of four different functions, or transformations: add key (A_K) nibble substitution (NS), shift row (SR), and mix column (MC).
- The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of four functions; and round 2 contains only 3 functions
- Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key.



Each function operates on a 16-bit state, treated as a 2 x 2 matrix of nibbles, where one nibble equals 4 bits. The initial value of the state matrix is the 16-bit plaintext.

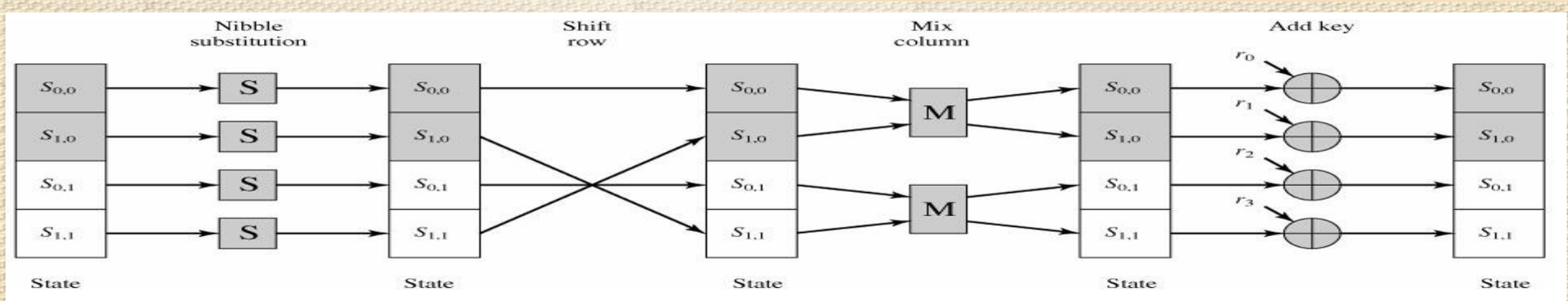




The 16-bit key is similarly organized, but it is somewhat more convenient to view the key as two bytes rather than four nibbles The expanded key of 48 bits is treated as three round keys, whose bits are labelled as follows: K0 = *k0...k15*; K1 = *k16...k31*; K2 = *k32...k47*.

The encryption algorithm can be expressed as: $A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$

The decryption process can be expressed as: $A_{K_0} \circ INS \circ ISR \circ IMC \circ A_{K_1} \circ INS \circ ISR \circ A_{K_2}$



S-AES Encryption Process

Add Key

The add key function consists of the bitwise XOR of the 16-bit state matrix and the 16-bit round key. The inverse of the add key function is identical to the add key function, because the XOR operation is its own inverse.

A	4
7	9

state matrix

$\oplus$

2	5
D	5

key

$=$

8	1
A	C

Nibble Substitution

AES defines a 4 x 4 matrix of nibble values, called an S-box that contains a permutation of all possible 4-bit values. Each individual nibble of the state matrix is mapped into a new nibble in the following way: The leftmost 2 bits of the nibble are used as a row value and the rightmost 2 bits are used as a column value.

i \ j	00	01	10	11
00	9	4	A	B
01	D	1	8	5
10	6	2	0	3
11	C	E	F	7

(a) S-Box

i \ j	00	01	10	11
00	A	5	9	B
01	1	7	8	F
10	6	0	2	3
11	C	4	D	E

(b) Inverse S-Box

After nibble substitution, the output is

8	1
A	C

$\rightarrow$

6	4
0	C

Shift Row

The shift row function performs a one-nibble circular shift of the second row of the state matrix; the first row is not altered. The inverse shift row function is identical to the shift row function, because it shifts the second row back to its original position.

6	4
0	C

→

6	4
C	0

Mix Column

The mix column function operates on each column individually. Each nibble of a column is mapped into a new value that is a function of both nibbles in that column. The transformation can be defined by the following matrix multiplication on the state matrix, Where arithmetic is performed in GF(2^4)

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as follows:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

S-AES Key Expansion

For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. The following figure shows the expansion into 6 words, by the calculation of 4 new words from the initial 2 words. The algorithm is as follows:

$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \text{RCON}(1) \oplus \text{SubNib}(\text{RotNib}(w_1))$$

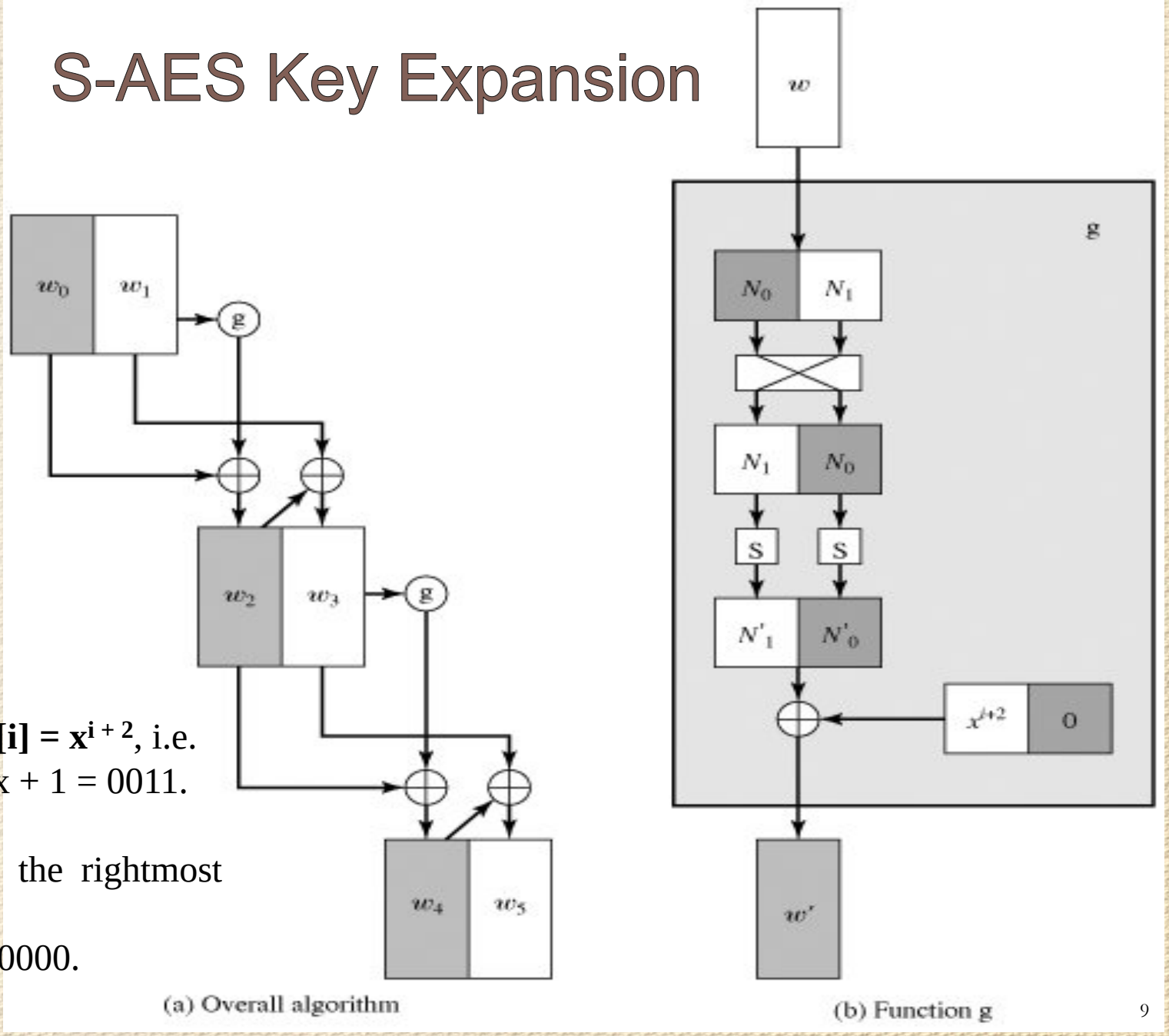
$$w_3 = w_2 \oplus w_1$$

$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus \text{RCON}(2) \oplus \text{SubNib}(\text{RotNib}(w_3))$$

$$w_5 = w_4 \oplus w_3$$

RCON is a round constant, defined as follows: **RC[i] = x^{i+2}**, i.e. RC[1]=x^3=1000 and RC[2]=x^4 mod (x^4 + x + 1) = x + 1 = 0011.

RC[i] forms the leftmost nibble of a byte, with the rightmost nibble being all zeros.
Thus, RCON(1) = 10000000 and RCON(2) = 00110000.



For example, suppose the key is 2D55 = 0010 1101 0101 0101 = w_0w_1. Then

$$w_2 = 00101101 \oplus 10000000 \oplus \text{SubNib}(01010101)$$

$$= 00101101 \oplus 10000000 \oplus 00010001 = 10111100$$

$$w_3 = 10111100 \oplus 01010101 = 11101001$$

$$w_4 = 10111110 \oplus 00110000 \oplus \text{SubNib}(10011110)$$

$$= 10111100 \oplus 00110000 \oplus 00101111 = 10100011$$

$$w_5 = 10100011 \oplus 11101001 = 01001010$$

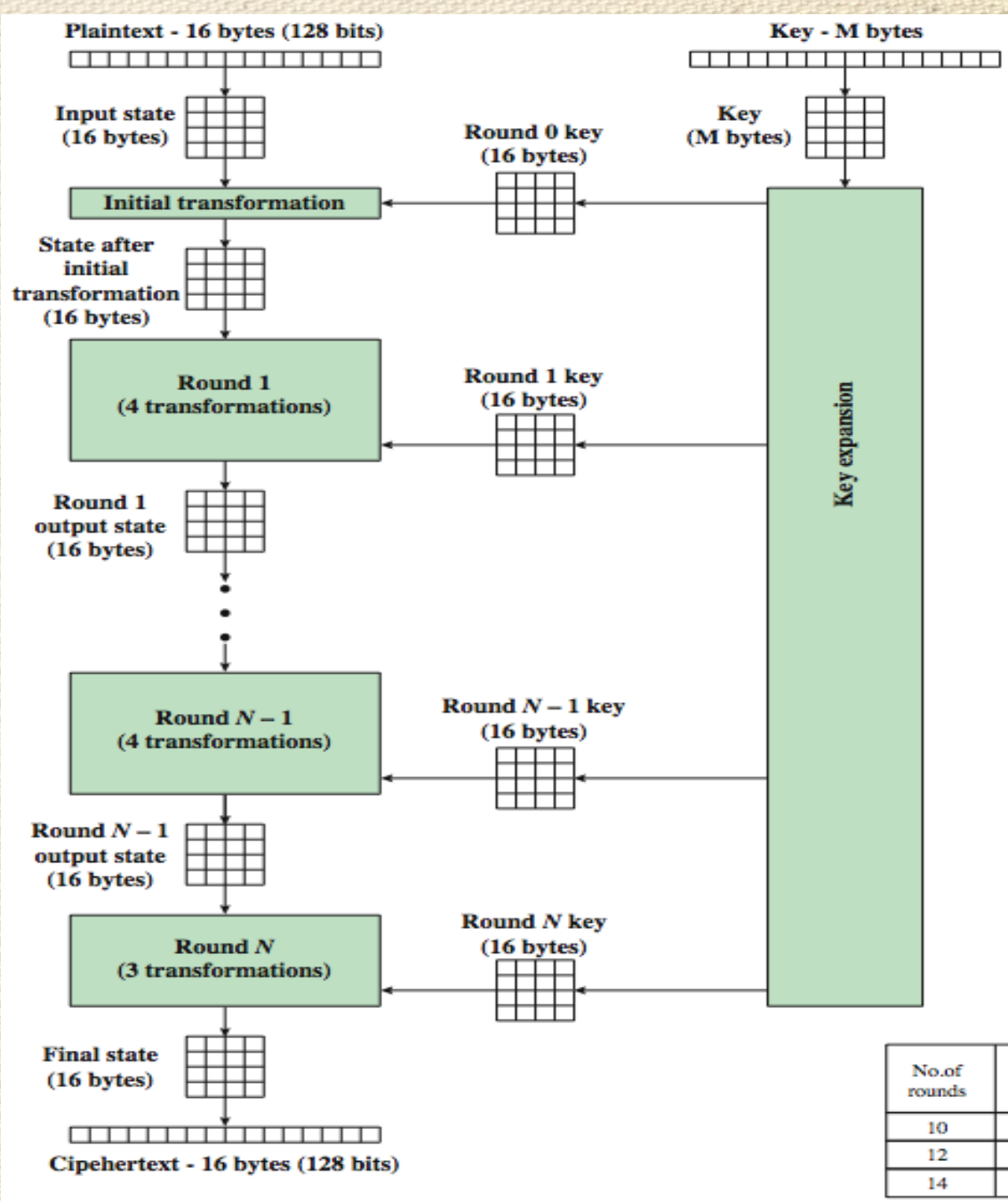
Exercise Use the key 1010 0111 0011 1011 to encrypt the plaintext "ok" as expressed in ASCII, that is 0110 1111 0110 1011. The designers of S-AES got the ciphertext 0000 0111 0011 1000. Do you?

AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to have:
  - resistance against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Joan Daemen

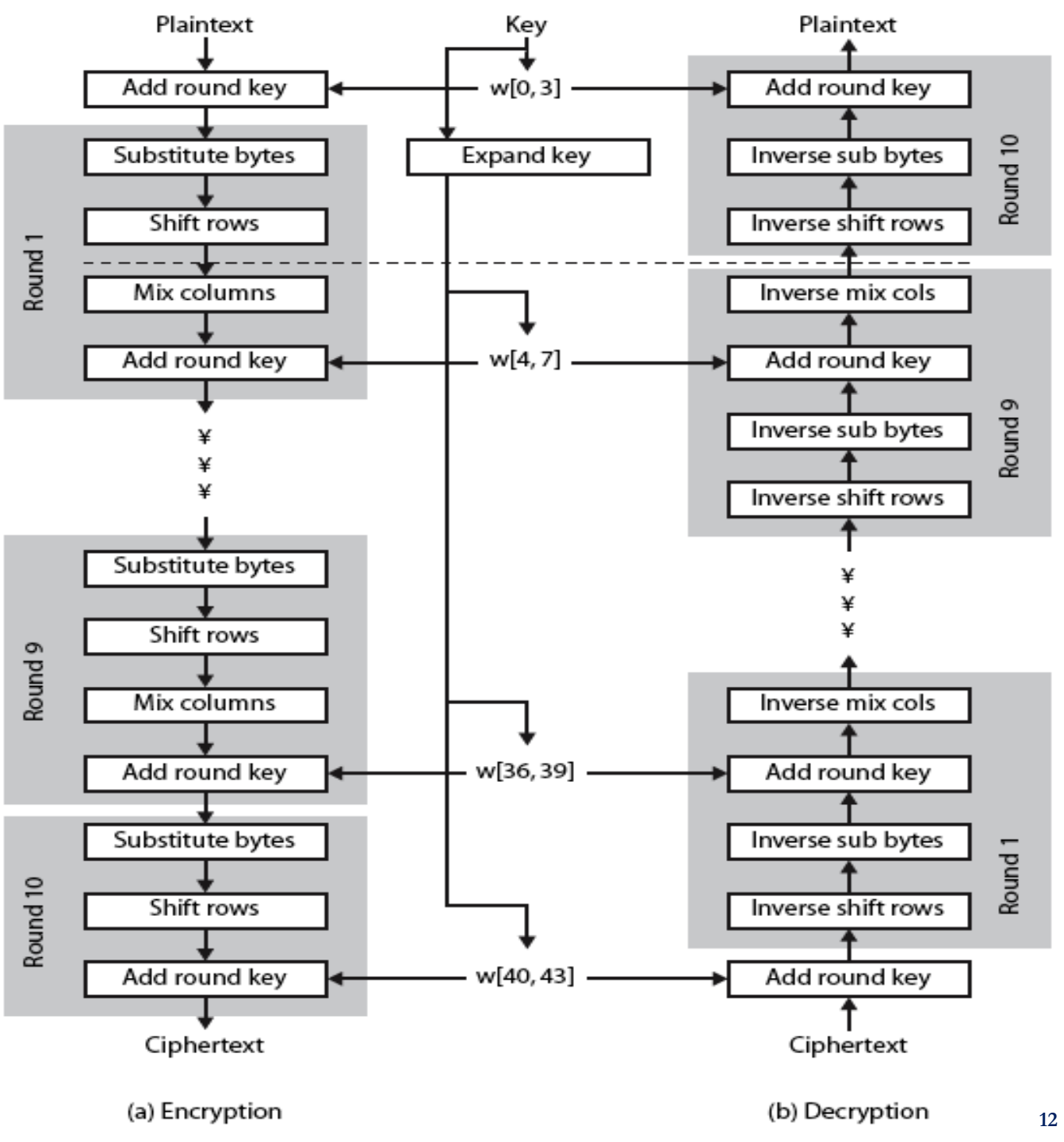
Vincent Rijmen



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

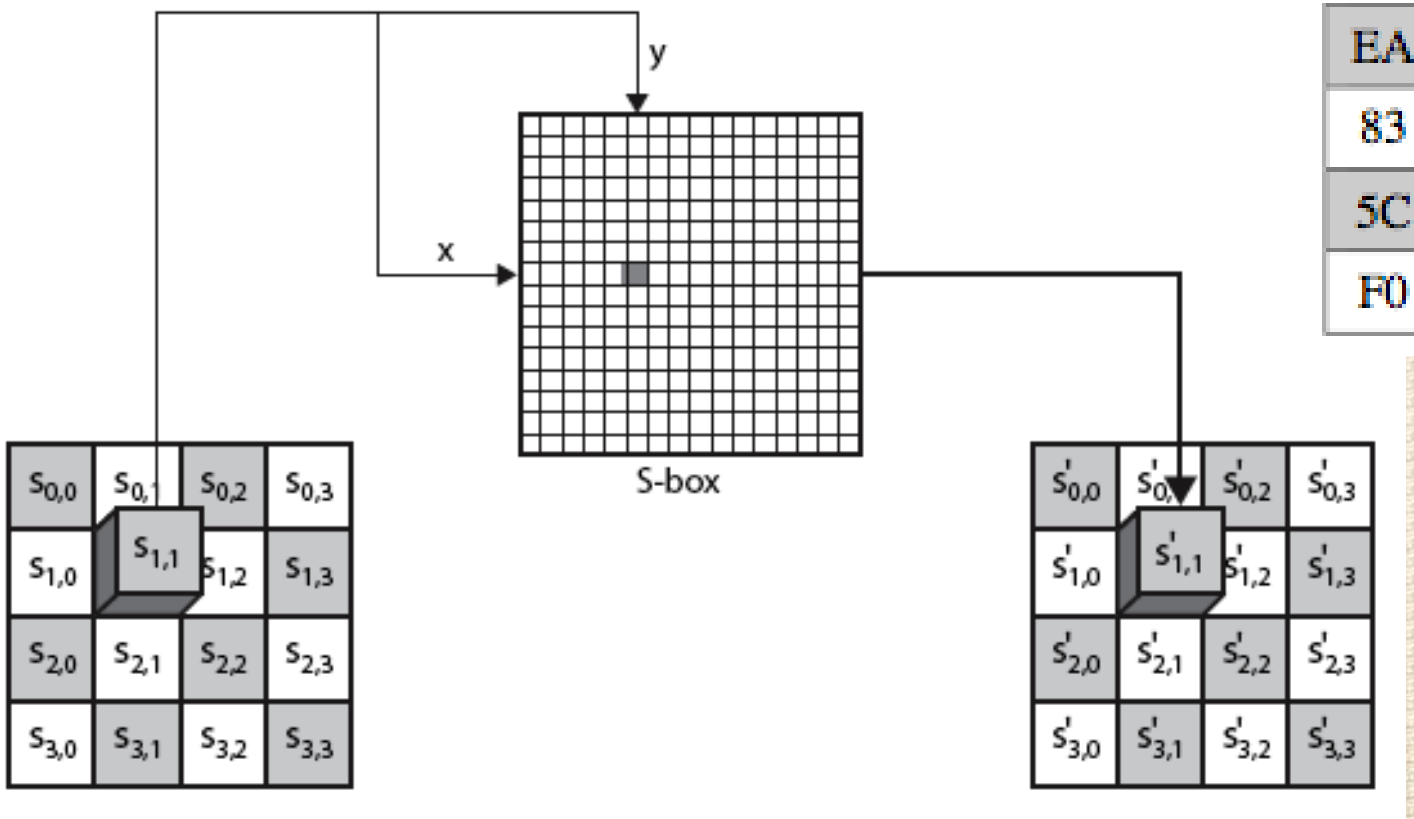
AES Structure

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation



Substitute Bytes

- a simple substitution of each byte
- uses one table of 16x16 bytes called an **s-box** containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
- eg. byte {95} is replaced by byte in row 9 column 5 which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks



EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Substitute Bytes Example

(a) S-box

		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

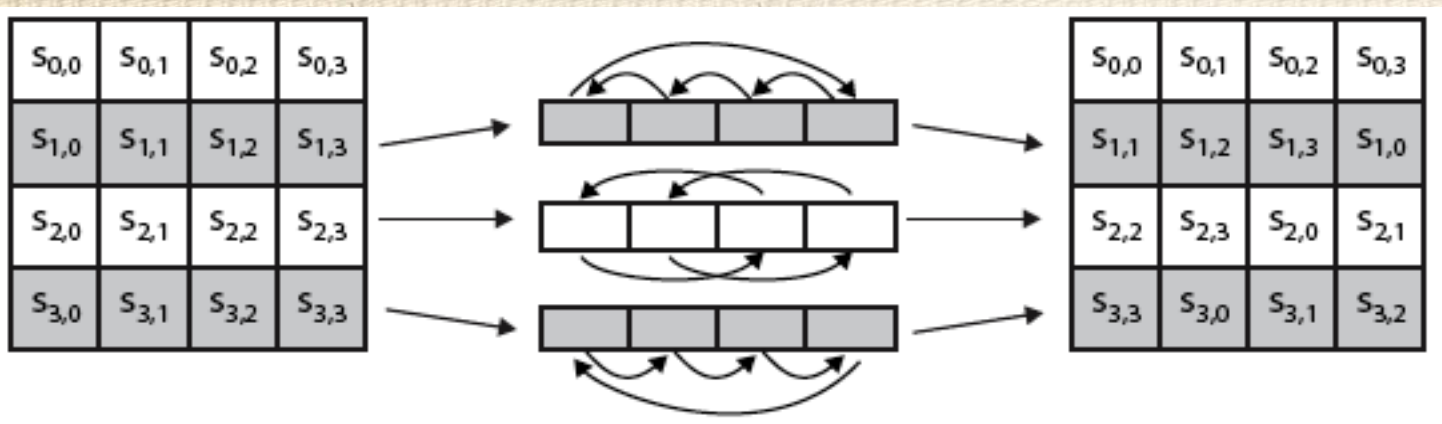
(b) Inverse S-box

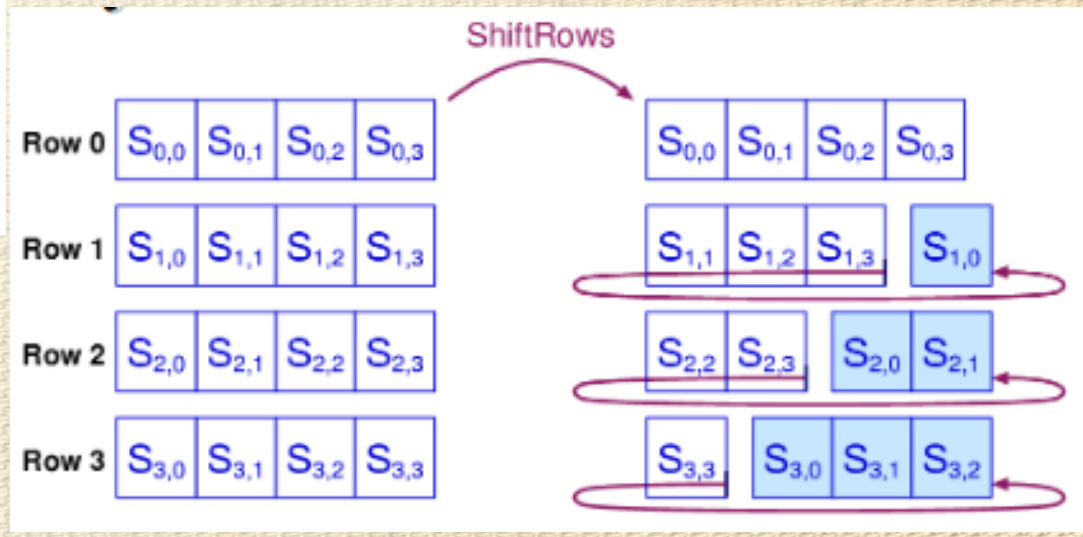
		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65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

- A circular byte shift in each row. This is a simple permutation and nothing more.
  - **1st row is unchanged**
  - **2nd row does 1 byte circular shift to left**
  - **3rd row does 2 byte circular shift to left**
  - **4th row does 3 byte circular shift to left**

Shift Row

- The Inverse Shift Rows transformation (known as InvShiftRows) performs these circular shifts in the opposite direction for each of the last three rows (the first row was unaltered to begin with).





87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

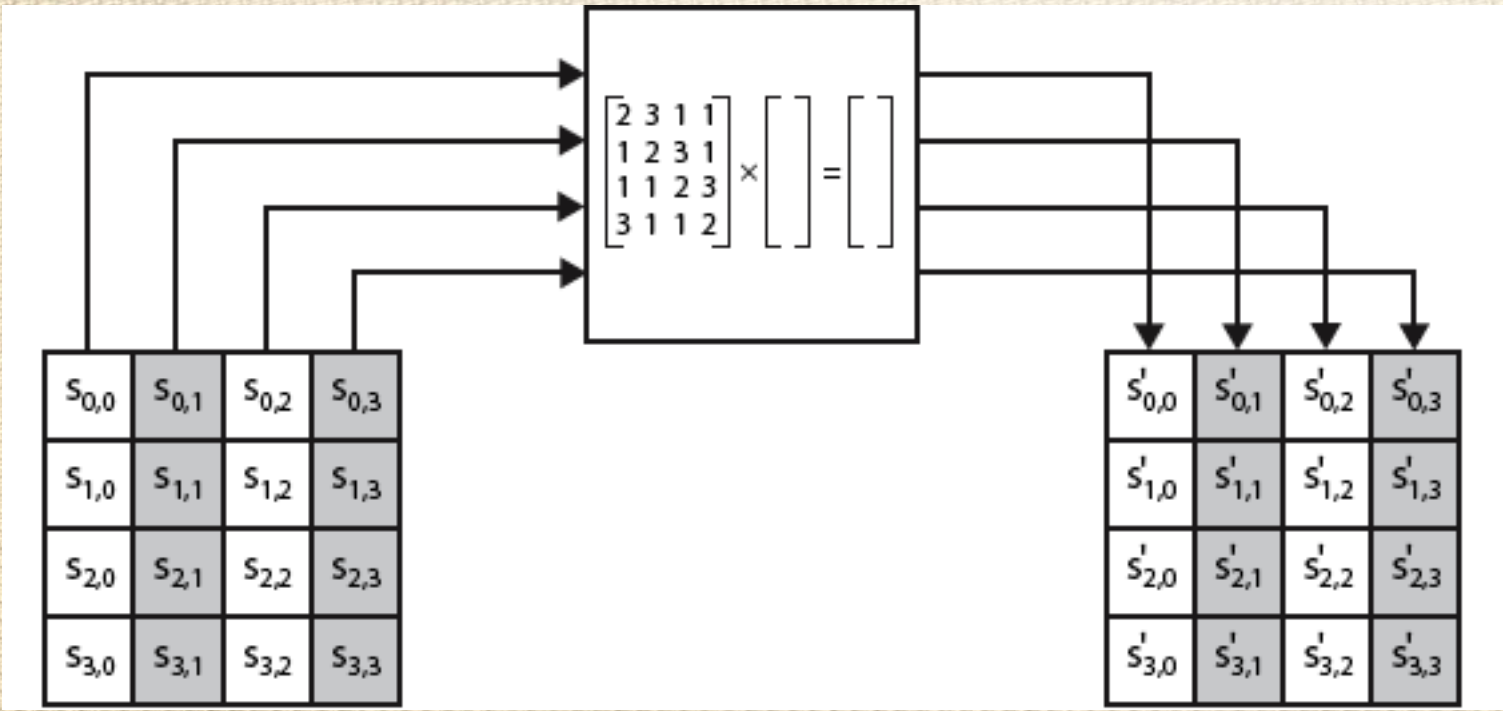
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Shift Row Example

Mix Column

- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Effectively a matrix multiplication in GF(2^8) using prime poly m(x) =$x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$



$$s'_{0,j} = (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j})$$
$$s'_{3,j} = (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})$$

$$
\begin{aligned}
(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\
\{87\} \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} &= \{37\} \\
\{87\} \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{03\} \bullet \{A6\}) &= \{94\} \\
(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) &= \{ED\}
\end{aligned}
$$

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

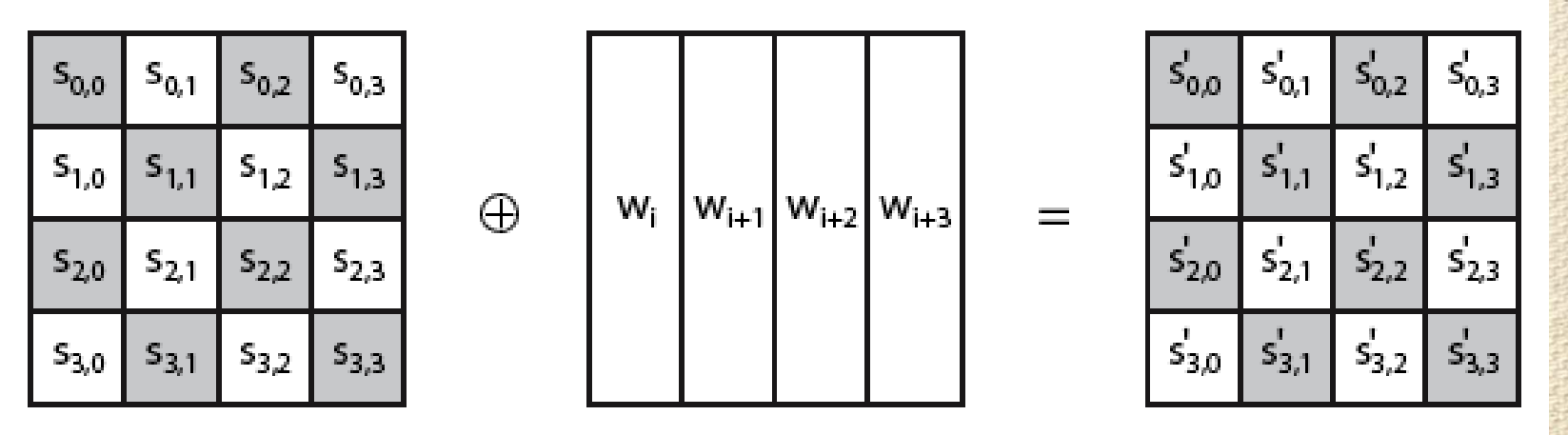
Mix Column Example

The InvMixColumns is defined by the following matrix multiplication:

$$
\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}
\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}
=
\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}
$$

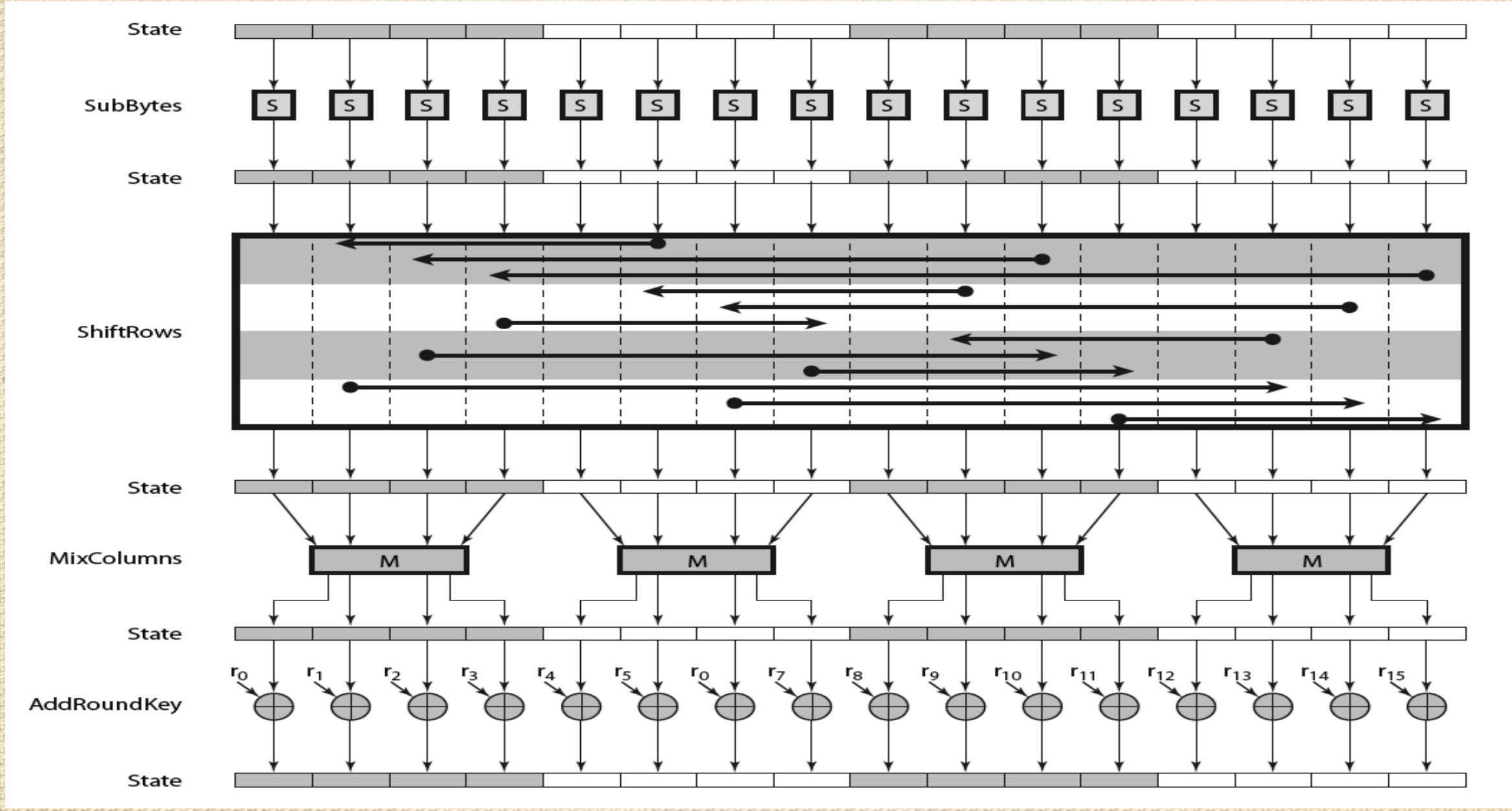
Add Round Key

- In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key.
- The operation is viewed as a columnwise operation between the 4 bytes of a state column and one word of the round key.
- This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.



- Inverse for decryption is identical since XOR is its own inverse, with reversed keys

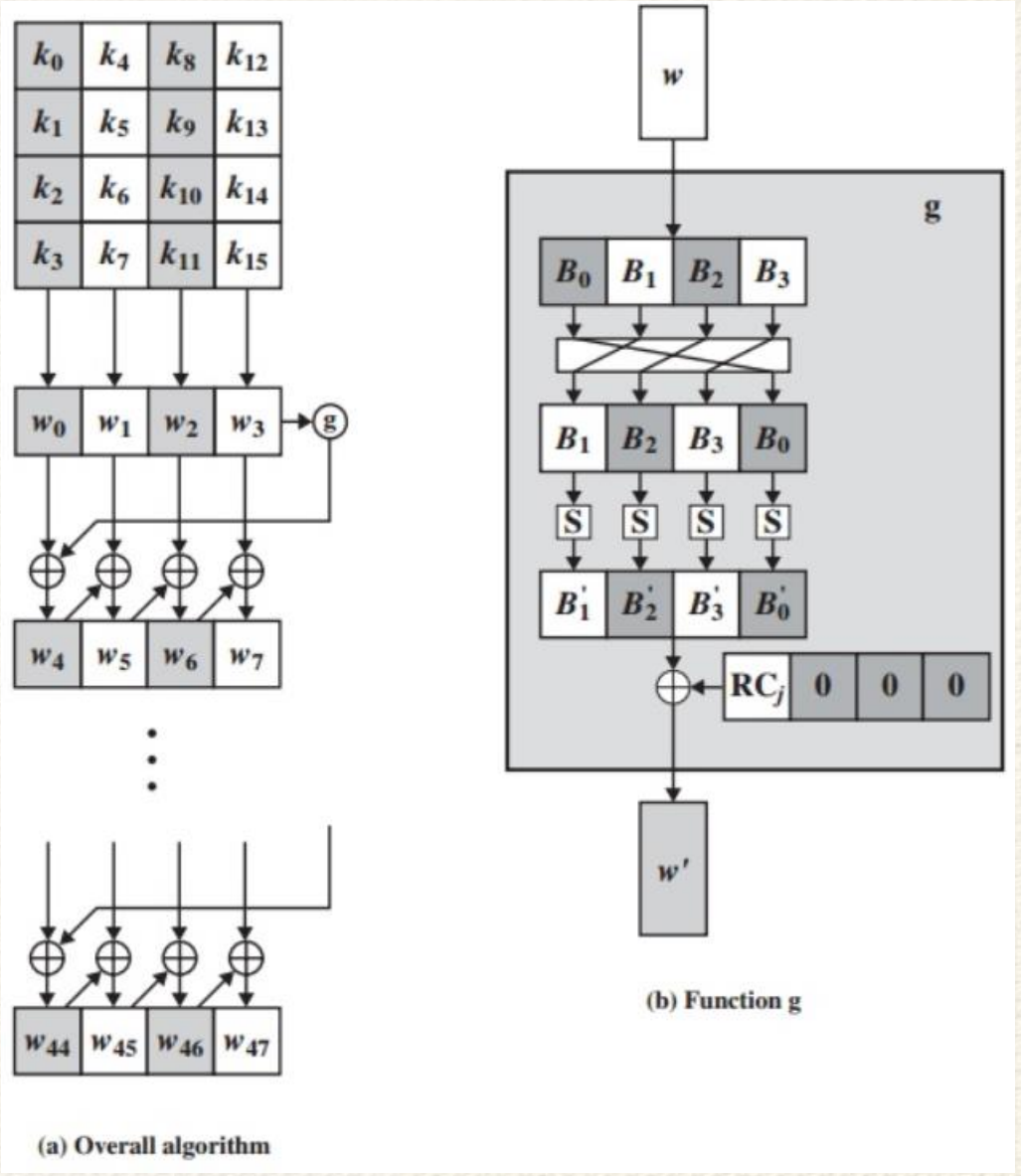
AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r0
r1
r2
r3
r4
r5
r0
r7
r8
r9
r10
r11
r12
r13
r14
r15
State

AES Key Expansion

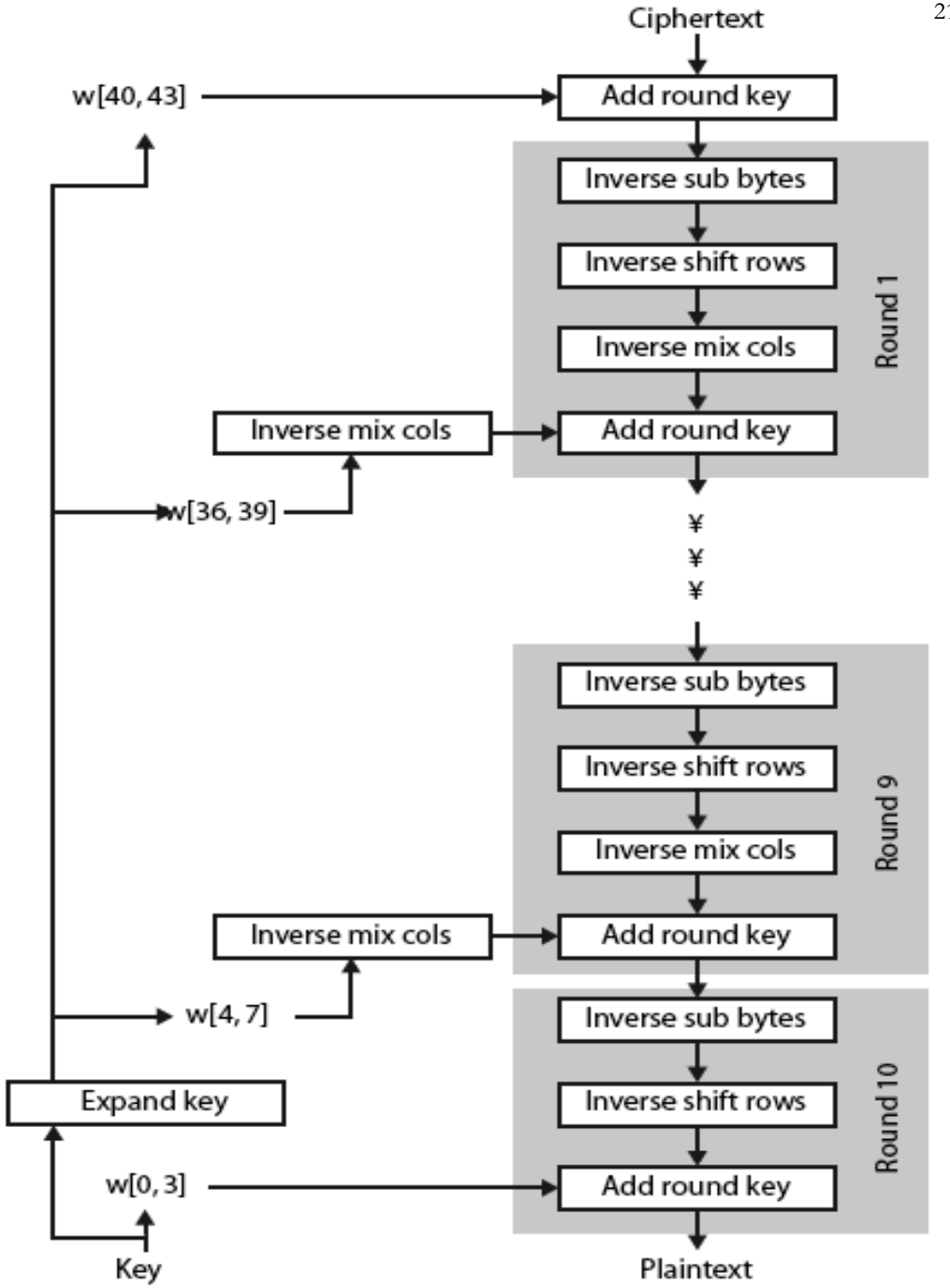
- The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.
- The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time.
- Each added word w[i] depends on the immediately preceding word, w[i-1], and the word four positions back, w[i-4].
- For a word whose position in the w array is a multiple of 4, a more complex function *g* is used. The function g consists of the following subfunctions:
  - *RotWord* performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
  - *SubWord* performs a byte substitution on each byte of its input word, using the S-box.
  - The result of steps 1 and 2 is XORed with round constant, Rcon[j].
- The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
- The round constant is different for each round and is defined as

  Rcon[j] = (RC[J], 0,0,0), with RC[1]= 1, RC[j]= 2• RC[j −1] and with multiplication defined over the field GF(2^8).



AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- But an equivalent inverse cipher with steps as for encryption can be defined
  - but using inverses of each step with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key



AES Advantages

Advantages of AES over 3DES:

- AES is more secure (it is less susceptible to cryptanalysis than 3DES).
- AES supports larger key sizes than 3DES's 112 or 168 bits.
- AES is faster in both hardware and software.
- AES's 128-bit block size makes it less open to attacks via the birthday problem than 3DES with its 64-bit block size.
- AES is required by the latest U.S. and international standards.
- AES is federal information processing standard and there are currently no known non-brute-force direct attacks against AES

Homework

Given the plaintext [0001 0203 0405 0607 0809 0A0B 0C0D 0E0F] and the key [0101 0101 0101 0101 0101 0101 0101 0101]

a) Show the original contents of state, displayed as a 4x4 matrix.

b) Show the value of state after initial AddRoundKey.

c) Show the value of State after SubBytes.

d) Show the value of State after ShiftRows.

e) Show the value of State after MixColumns.

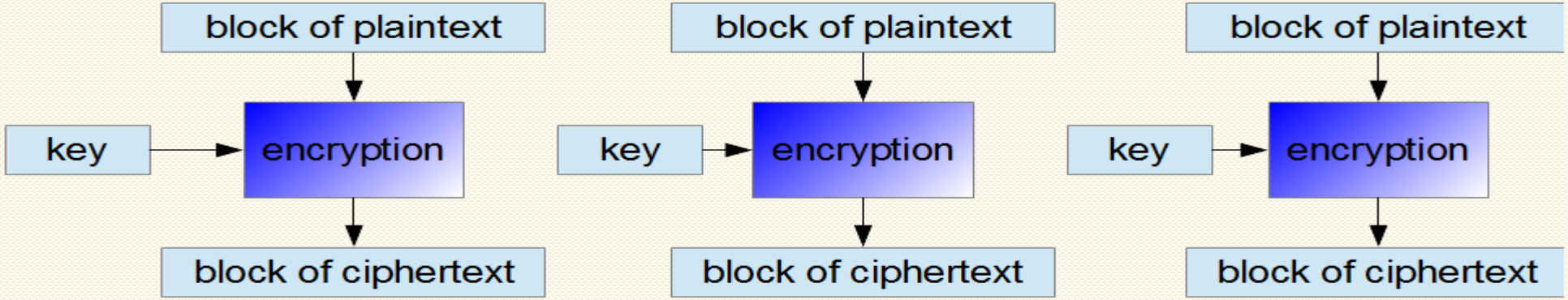
Cipher Block Modes of Operation

Block Cipher Modes of Operation

- A block cipher processes the data blocks of fixed size. Usually, the size of a message is larger than the block size. Hence, the long message is divided into a series of sequential message blocks, and the cipher operates on these blocks one at a time.
- The modes of operation of block ciphers are configuration methods that allow those ciphers to work with large data streams, without the risk of compromising the provided security.
- To apply a block cipher in a variety of applications, **five modes of operation have been defined by NIST (SP 800-38A).**
- The five modes are intended to cover a wide variety of applications of encryption for which a block cipher could be used. These modes are intended for use with any symmetric block cipher, including triple DES and AES.

ECB (electronic codebook) Mode

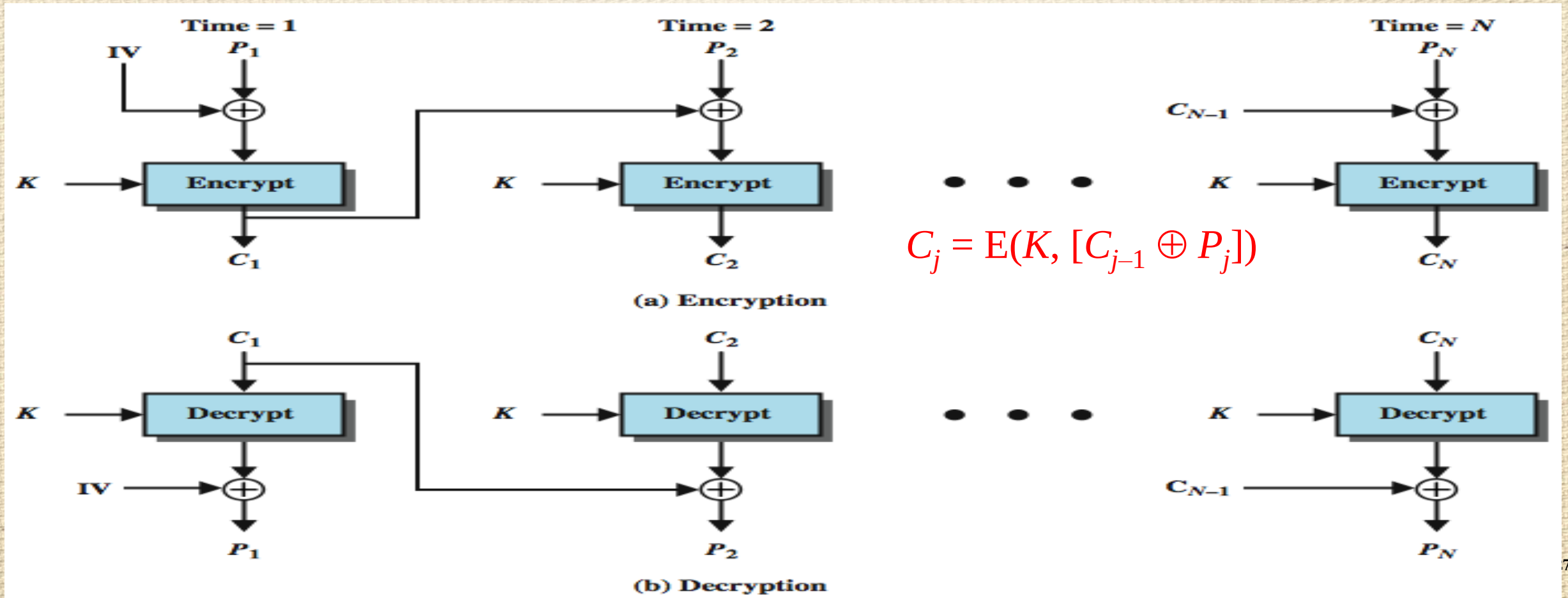
- This is the simplest mode in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key.
- The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.



- The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use.
- The most significant characteristic of ECB is that if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext making it insecure for lengthy messages.
- To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks

CBC (cipher-block chaining) Mode

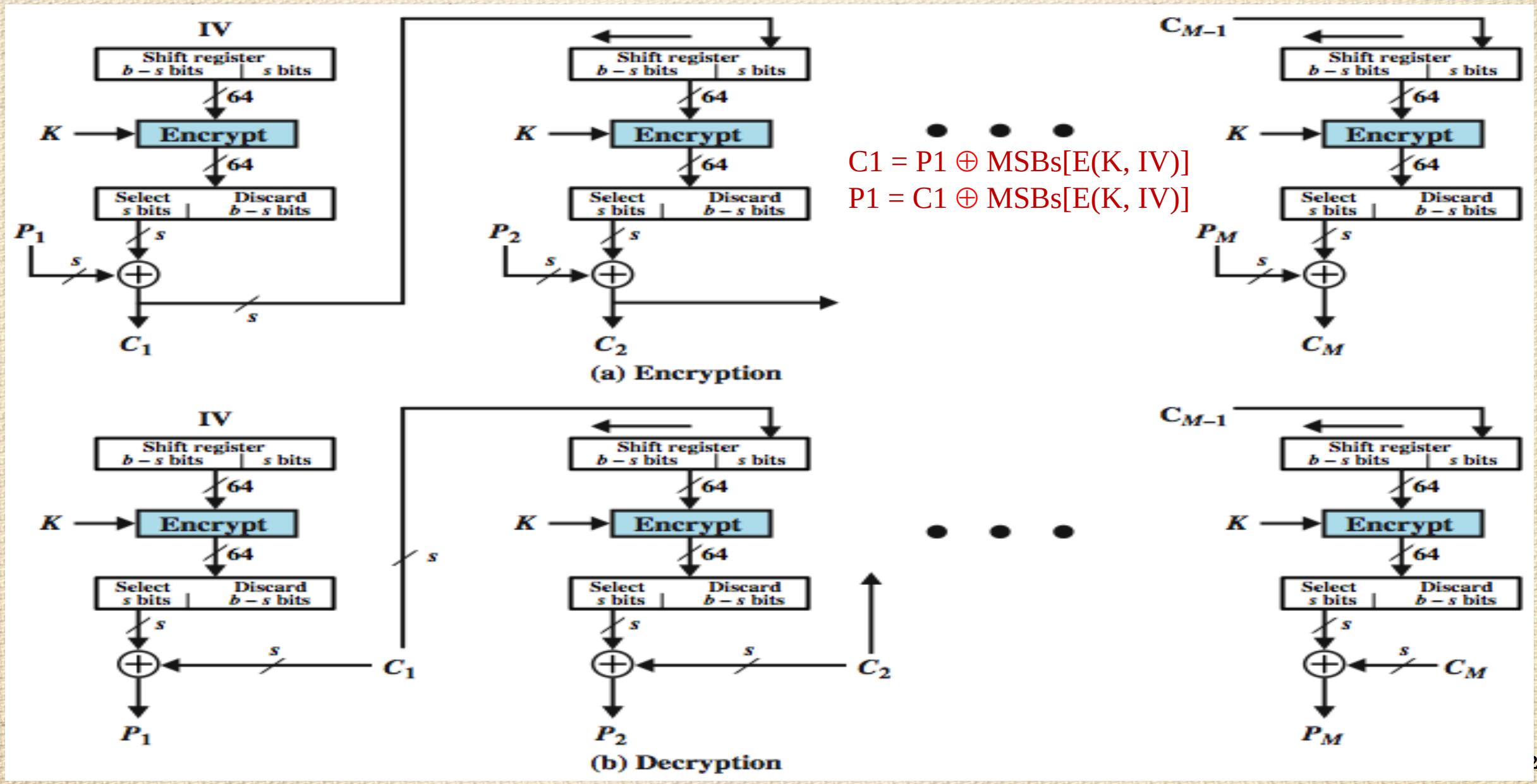
- The CBC mode of encryption was invented by IBM in 1976. Here, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block.
- The processing of the sequence of plaintext block is chained together and the output of the encryption algorithm does not bear any fixed relationship to the plaintext block.
- For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block.



- In CBC mode, repeating patterns of bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be **padded** to a full bits if it is a partial block.
- To produce the first block of ciphertext, an **initialization vector (IV)** is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block.
- The IV must be known to both the sender and receiver but be unpredictable by a third party.
- For maximum security, the IV should be **protected against unauthorized changes**. This could be done by sending the IV using ECB encryption
- If one bit of a plaintext message is damaged (for example because of some earlier transmission error), all subsequent ciphertext blocks will be damaged and it will be never possible to decrypt the ciphertext received from this plaintext. As opposed to that, if one ciphertext bit is damaged, only two received plaintext blocks will be damaged.

CFB (cipher feedback) Mode

It is possible to convert a block cipher into a stream cipher, using one of the next three modes



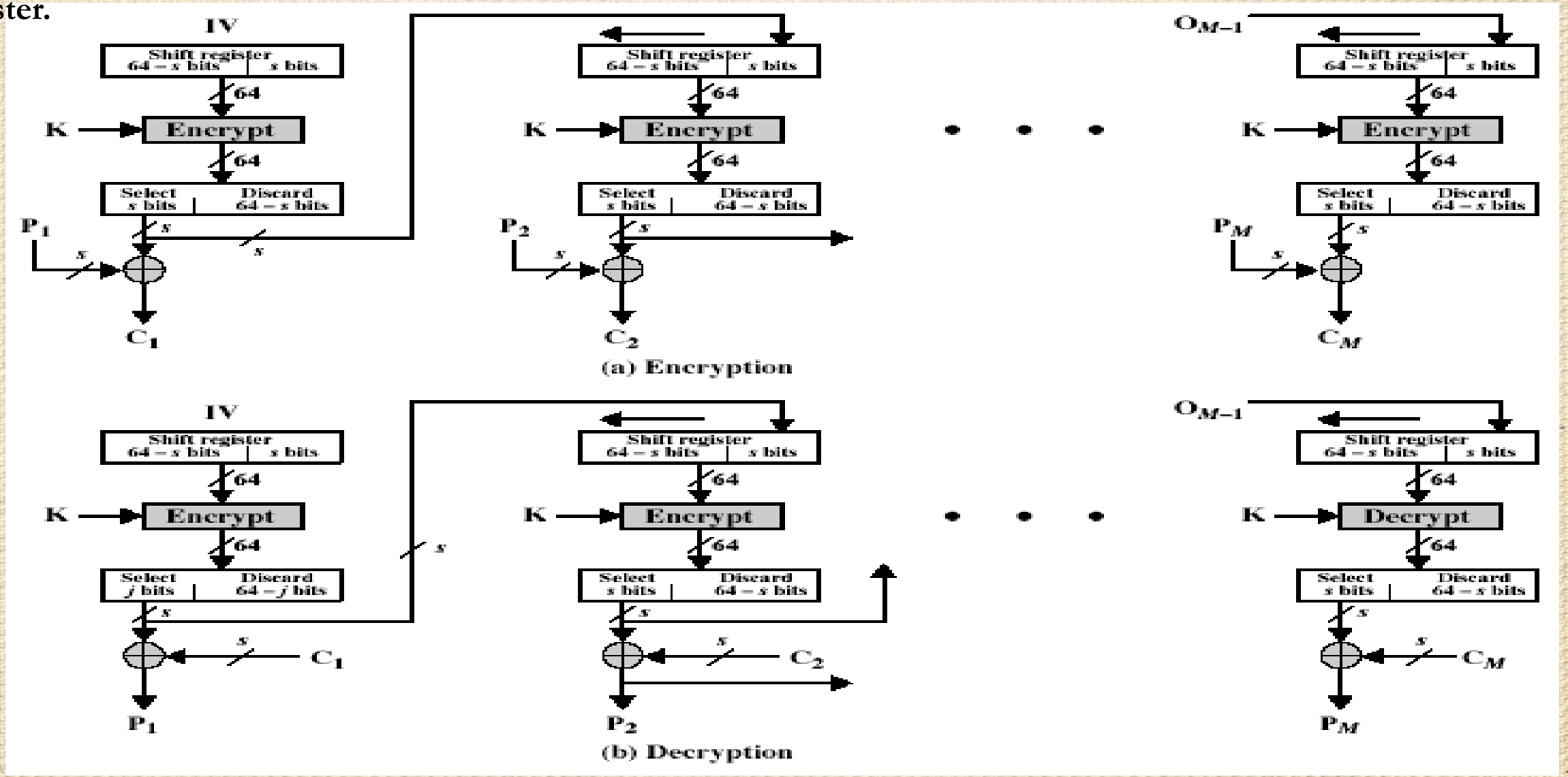
C1 = P1 ⊕ MSBs[E(K, IV)]

P1 = C1 ⊕ MSBs[E(K, IV)]

- A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.
- As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, **rather than blocks of bits, the plaintext is divided into segments of bits**.
- The input to the **encryption function** is a b-bit shift register that is initially set to some initialization vector (IV).The leftmost (most significant) bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C1, which is then transmitted.
- In addition, the contents of the shift register are shifted left by bits, and is placed in the rightmost (least significant) bits of the shift register. This process continues until all plaintext units have been encrypted.
- For **decryption**, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.

OFB (output feedback) Mode

- The output feedback (OFB) mode is similar in structure to that of CFB. It is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.



- In the case of OFB, the IV must be a nonce; that is, the IV must be unique to each execution of the encryption operation.

- Here the generation of the "random" bits is independent of the message being encrypted. The advantage is that firstly, they can be computed in advance, good for bursty traffic, and secondly, any bit error only affects a single bit making it good for noisy links like satellite communication etc.

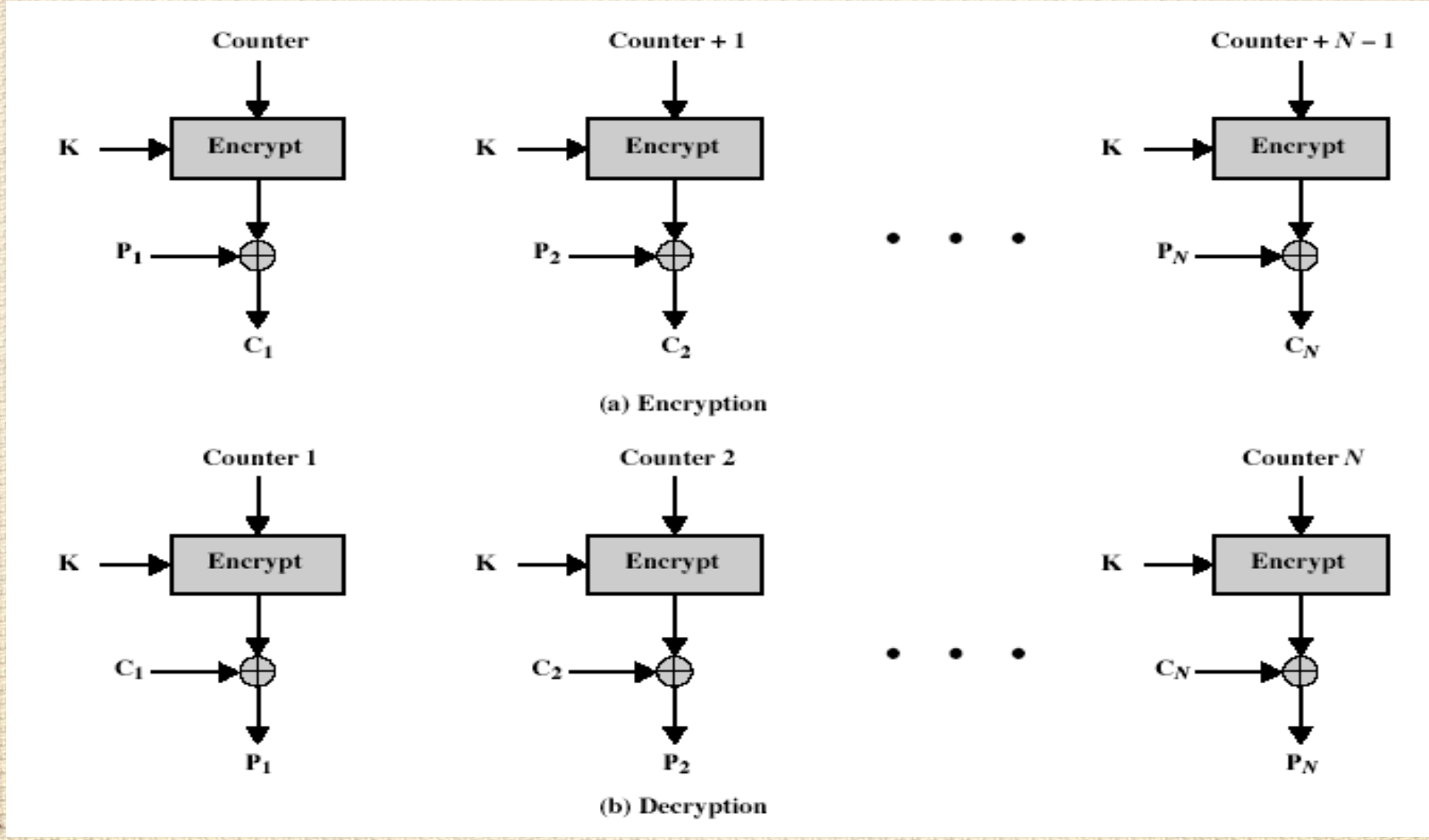
- Advantage of OFB is that the transmission errors do not propagate. The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

$$C_j = P_j \oplus O_j$$
$$P_j = C_j \oplus O_j$$
$$O_j = E_K(I_j)$$
$$I_j = O_{j-1}$$
$$I_0 = \text{IV}$$

CTR (Counter) Modee

- The Counter (CTR) mode is a variant of OFB, but which encrypts a counter value (hence name). Although it was proposed many years before, it has only recently been standardized for use with AES along with the other existing 4 modes.
- All modes of operations except ECB make random access to the file impossible: to access data at the end of the file one has to decrypt everything.
- A counter equal to the plaintext block size is used. A requirement is that the counter value must be different for each plaintext block that is encrypted
- The counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b, where is the block size).
- For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

- CTR mode has a number of advantages in parallel h/w & s/w efficiency, can preprocess the output values in advance of needing to encrypt, can get random access to encrypted data blocks, and is simple. But like OFB have issue of not reusing the same key + counter value



Summary of Block Modes of Operation

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 DES output.	•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 of Cipher Block Modes

Operation Mode	*Description*	*Type of Result*	*Data Unit Size*
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

Comparison of Modes

Mode	Description	Application
ECB	64-bit plaintext block encoded separately	Secure transmission of encryption key
CBC	64-bit plaintext blocks are XORed with preceding 64-bit ciphertext	Commonly used method. Used for authentication
CFB	s bits are processed at a time and used similar to CBC	Primary stream cipher. Used for authentication
OFB	Similar to CFB except that the output is fed back	Stream cipher well suited for transmission over noisy channels
CTR	Key calculated using the nonce and the counter value. Counter is incremented for each block	General purpose block oriented transmission. Used for high-speed communications

Modes and impact of an IV leakage

- An IV has different security requirements than a key
- Generally, an IV will not be reused under the same key
- CBC and CFB
  - reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages
- OFB and CTR
  - reusing an IV completely destroys security