

Advanced Encryption Standard

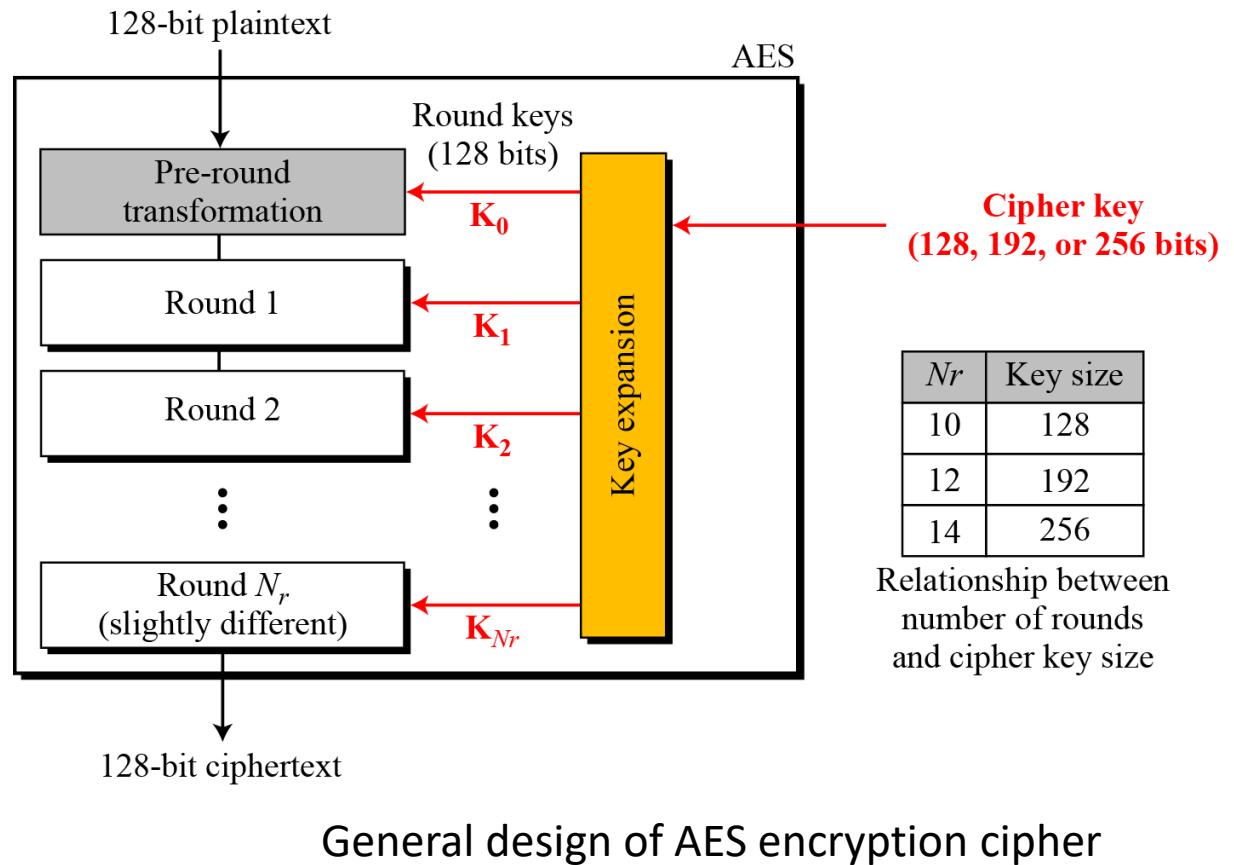
Course Teacher: Dr. Muhammad Usama

AES History

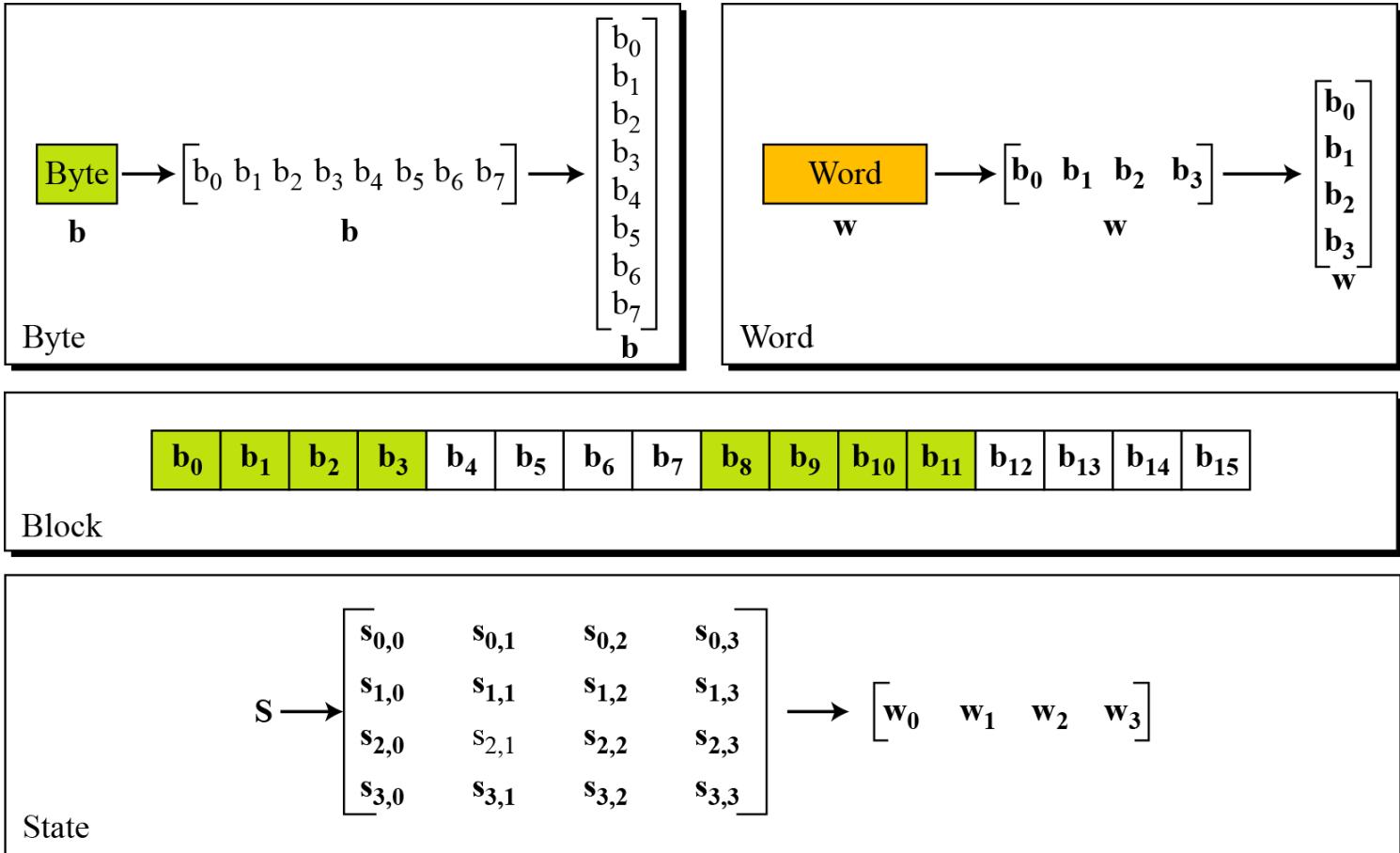
- Advanced Encryption Standard (AES) is a symmetric-key block cipher published by the NIST.
- In February 2001, NIST announced that a draft of the Federal Information Processing Standard (FIPS) was available for public review and comment.
- Finally, AES was published as FIPS 197 in the Federal Register in December 2001.
- The criteria defined by NIST for selecting AES fall into three areas:
 1. Security
 2. Cost
 3. Implementation

AES Rounds

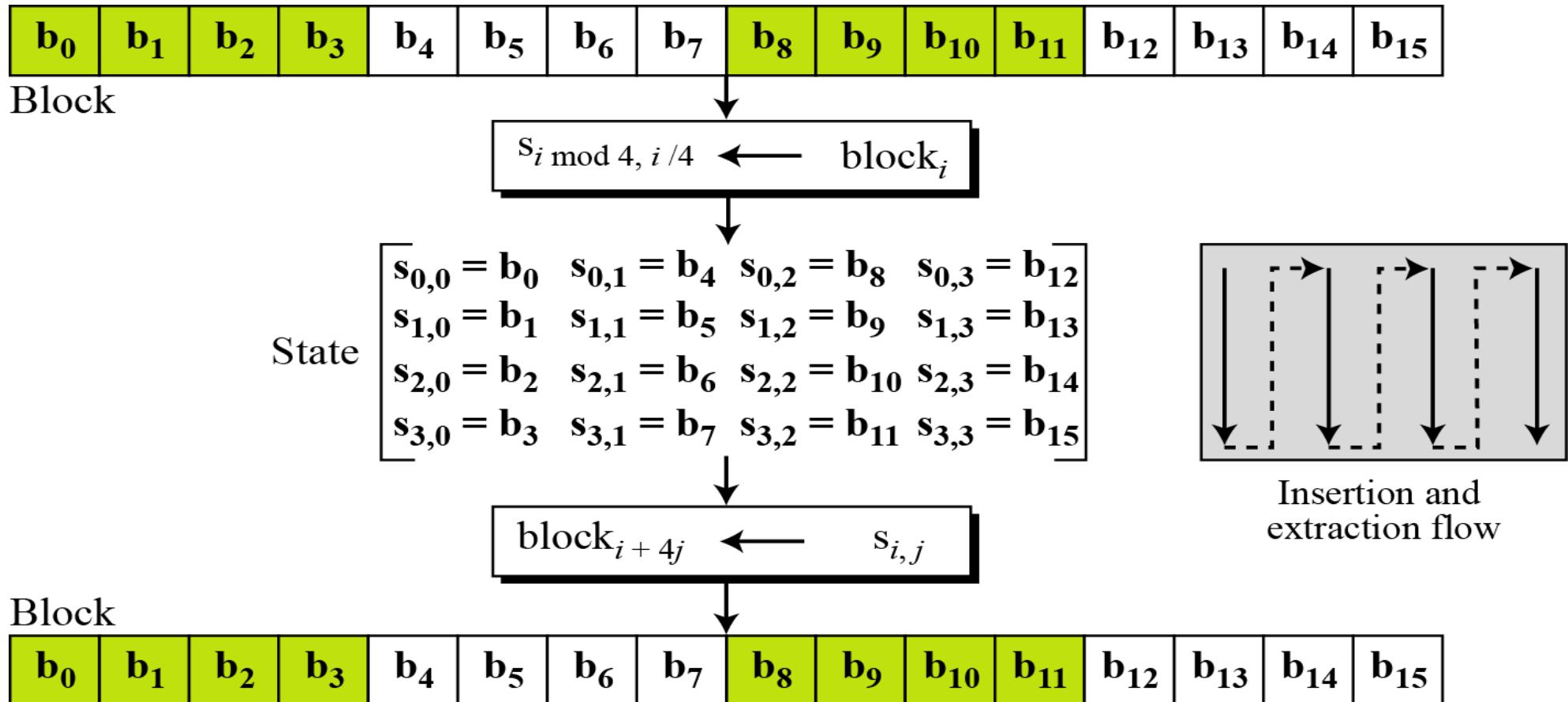
- AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits.
- AES has defined three versions, with 10, 12, and 14 rounds.
- Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.



AES Data Units



AES Data Units (Cont.)



Block-to-state and state-to-block transformation

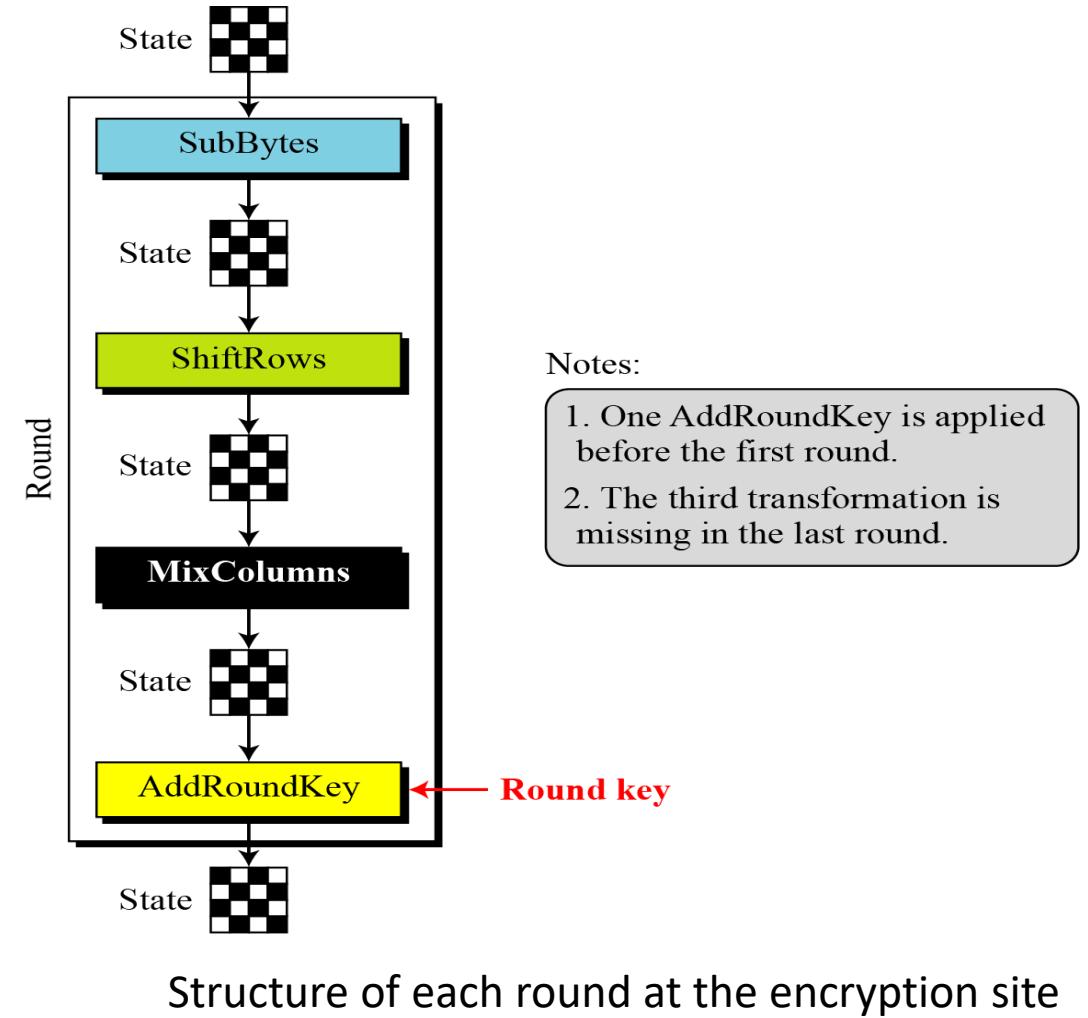
AES Parameters

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Note: number of rounds = words in a key block + 6

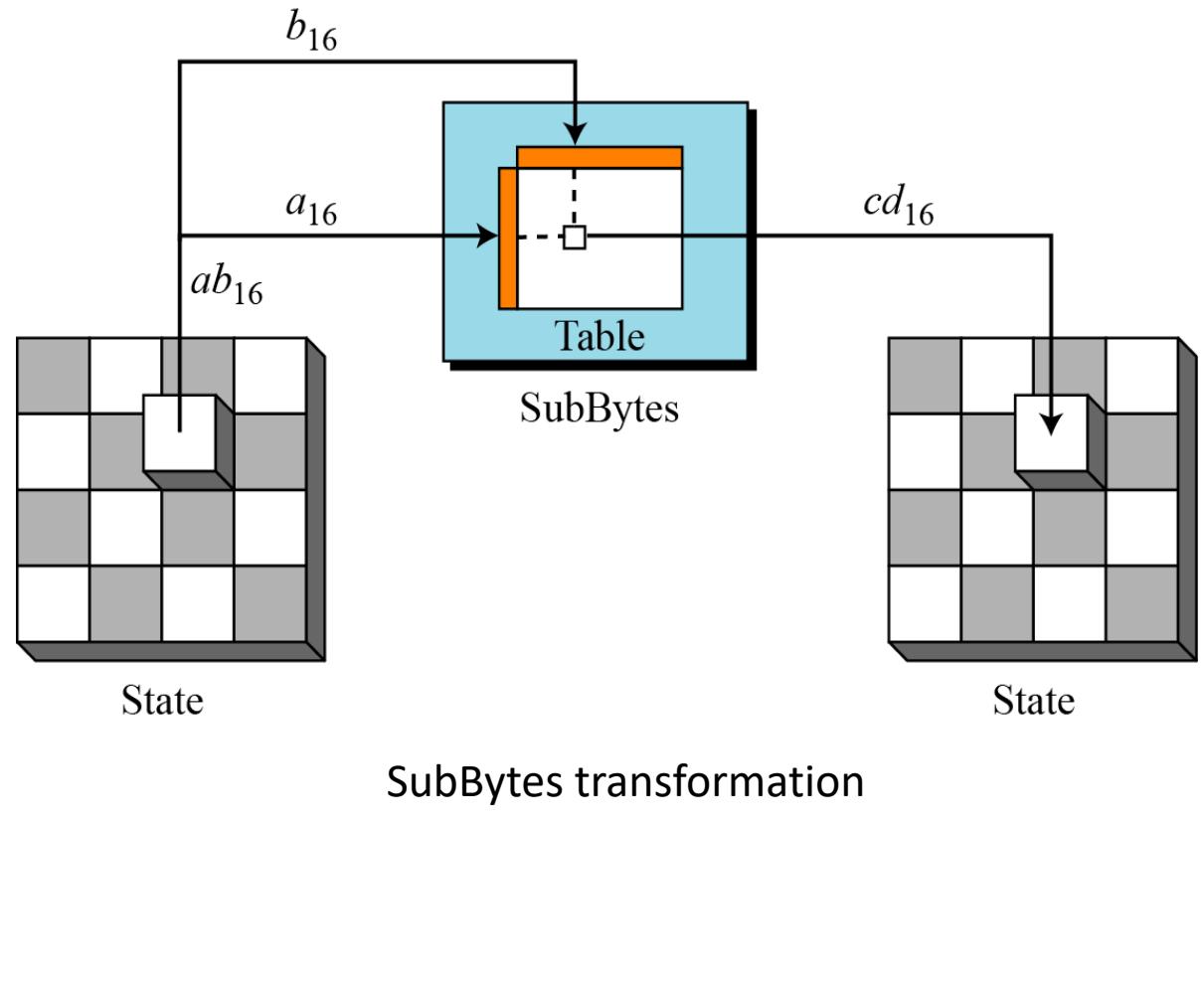
AES Structure of Each Round

- To provide security, AES uses four types of transformations:
 - Substitution (SubBytes)
 - Permutation (ShiftRows)
 - Mixing (MixColumns) and
 - key-adding (AddRoundKey)



AES Substitution (SubBytes)

- AES, like DES, uses substitution.
- The *SubBytes* operation involves **16 independent byte-to-byte** transformations.
- To substitute a byte, we interpret the byte as **two hexadecimal digits**.
- *SubBytes*, is used at the **encryption site**.
- *InvSubBytes*, is used at the **decryption site**.



SubBytes Transformation Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
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	CB	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

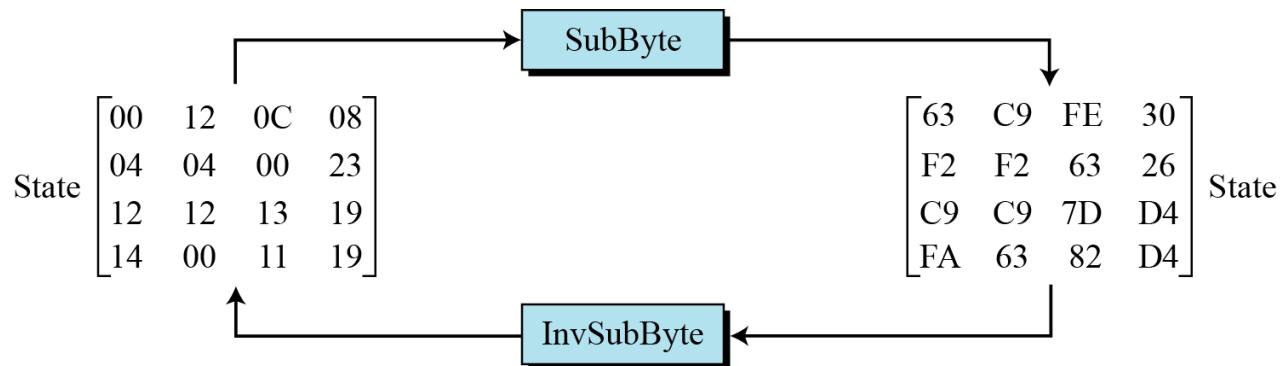
InvSubBytes Transformation Table

	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>
<i>0</i>	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
<i>1</i>	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
<i>2</i>	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
<i>3</i>	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
<i>4</i>	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
<i>5</i>	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
<i>6</i>	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
<i>7</i>	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
<i>8</i>	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
<i>9</i>	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
<i>A</i>	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
<i>B</i>	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
<i>C</i>	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
<i>D</i>	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
<i>E</i>	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
<i>F</i>	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

AES Substitution (SubBytes) (Cont.)

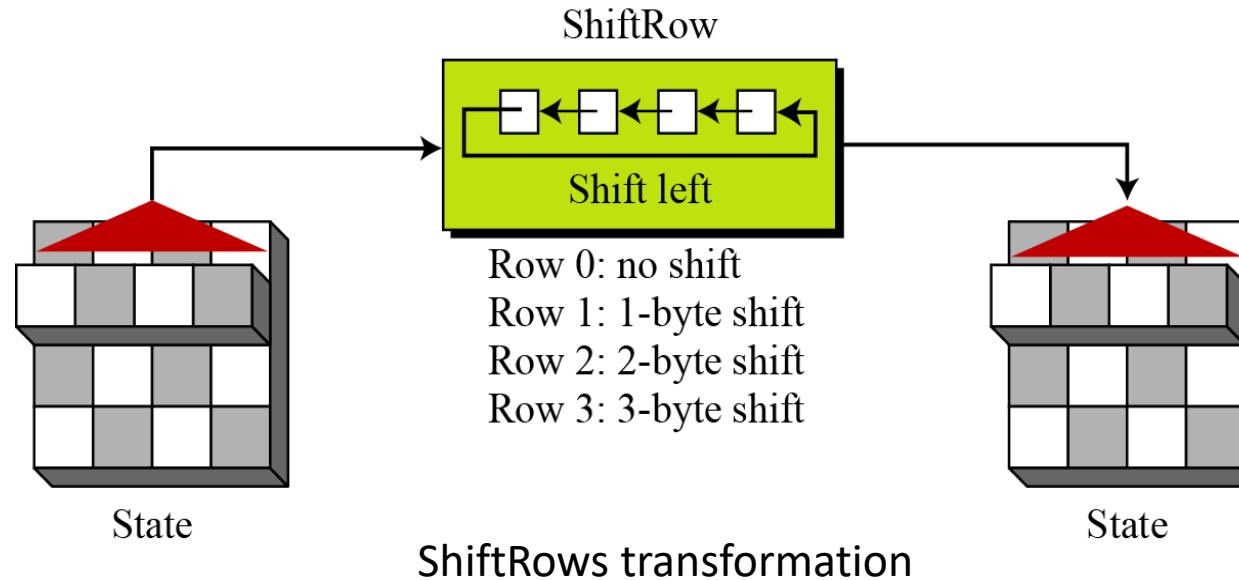
Example

- Figure shows how a state is transformed using the *SubBytes* transformation.
- The figure also shows that the *InvSubBytes* transformation creates the original one.
- **Note** that if the two bytes have the same values, their transformation is also the same.



AES Permutation (ShiftRows)

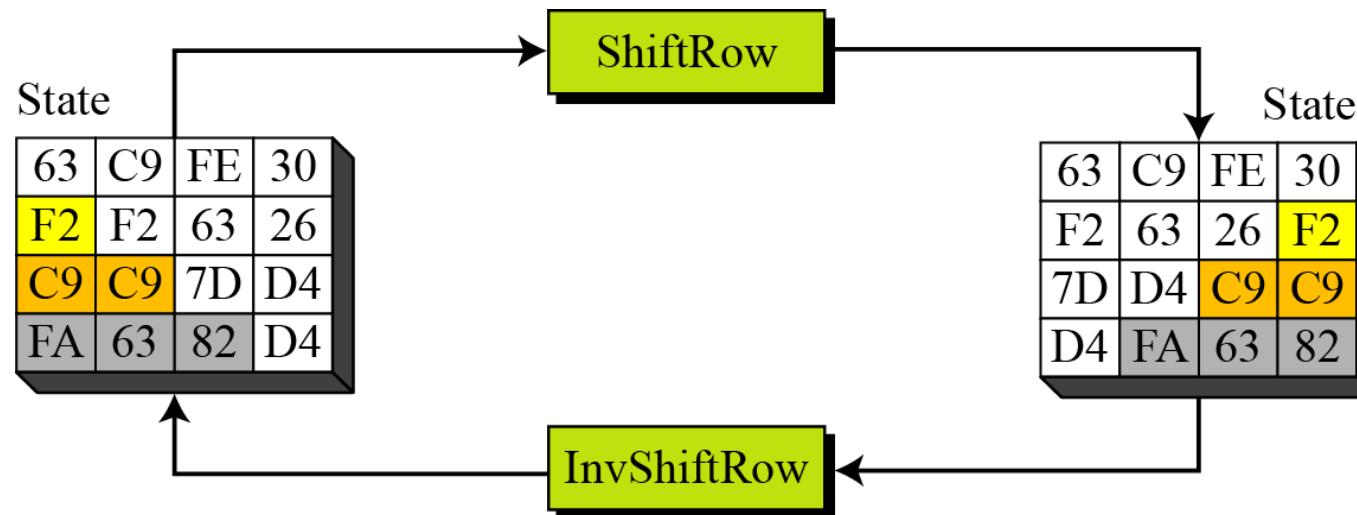
- Another transformation found in a round is **shifting**, which **permutes** the bytes.
- **In the encryption**, the transformation is called **ShiftRows** and the shifting is to the left.
- **In the decryption**, the transformation is called **InvShiftRows** and the shifting is to the right.



AES Permutation (ShiftRows) (Cont.)

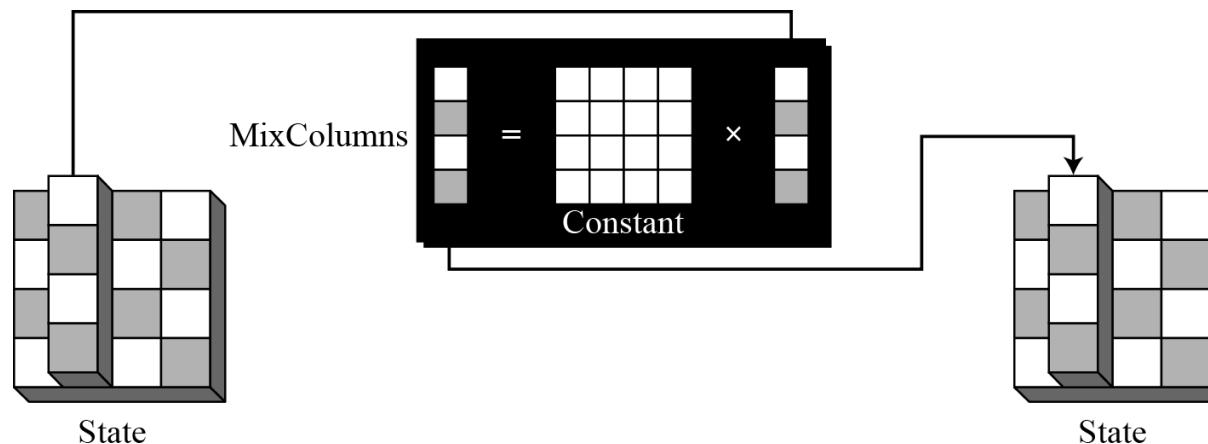
Example

- Figure shows how a state is transformed using ***ShiftRows*** transformation.
- The figure also shows that ***InvShiftRows*** transformation creates the original state.



AES Mixing (MixColumn)

- **MixColumns:** The *MixColumns* transformation operates at the column level; it transforms each column of the state to a new column.
- **InvMixColumns:** The *InvMixColumns* transformation is basically the same as the *MixColumns* transformation.



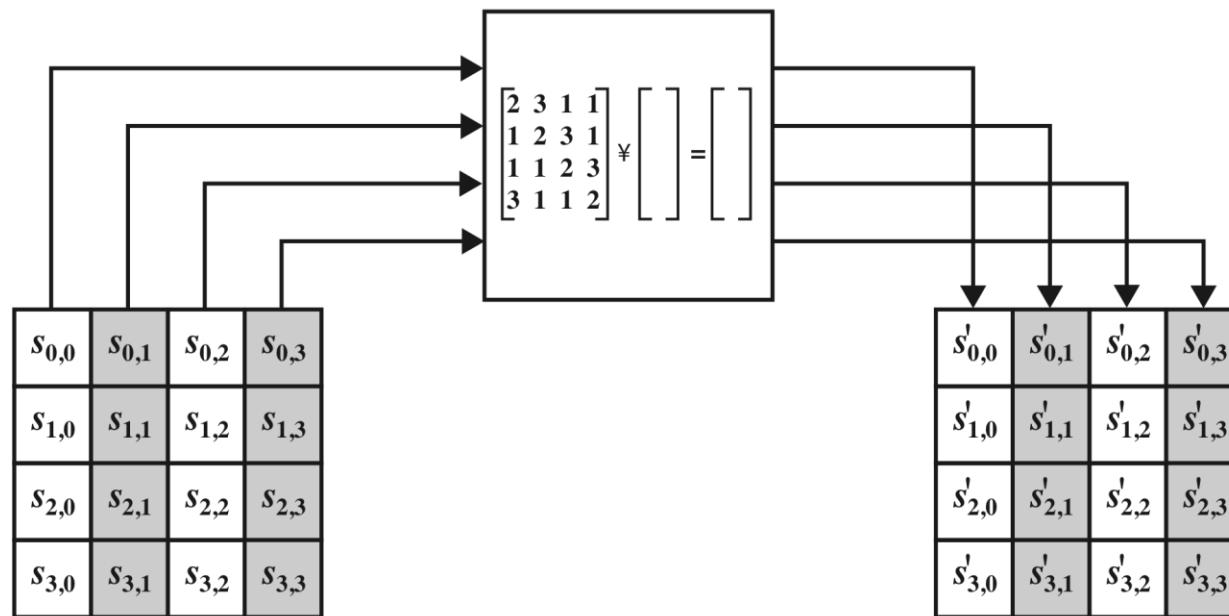
AES Mixing (MixColumn) (Cont.)

- We need an interbyte transformation that changes the bits inside a byte, based on the bits inside the neighboring bytes.
 - We need to mix bytes to provide diffusion at the bit level.
 - Mixing bytes using matrix multiplication

$$\begin{array}{l}
 \begin{matrix}
 ax + by + cz + dt \\
 ex + fy + gz + ht \\
 ix + jy + kz + lt \\
 mx + ny + oz + pt
 \end{matrix}
 \xrightarrow{\text{New matrix}}
 \left[\begin{array}{c} \color{red}{\boxed{\text{}}} \\ \color{red}{\boxed{\text{}}} \\ \color{red}{\boxed{\text{}}} \\ \color{red}{\boxed{\text{}}} \end{array} \right] = \left[\begin{array}{cccc} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{array} \right] \times \left[\begin{array}{c} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{array} \right]
 \end{matrix}$$

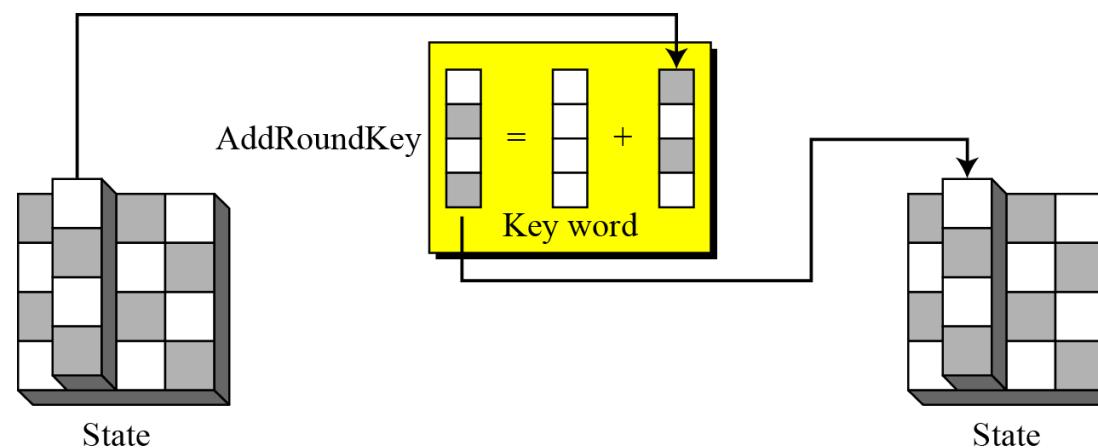
- Constant matrices used by
MixColumns and *InvMixColumns*

AES Mixing (MixColumn) (Cont.)

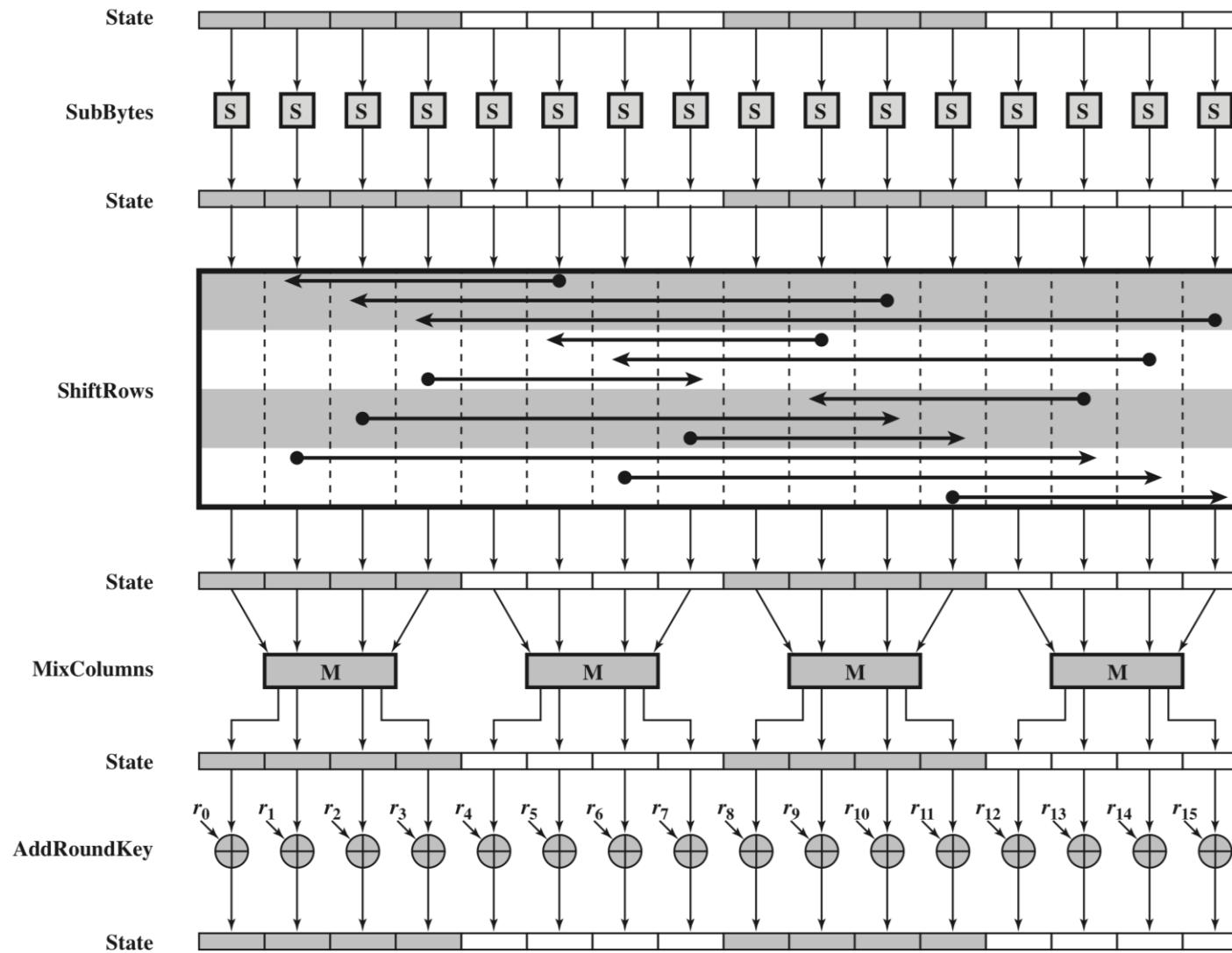


AES Key Adding (AddRoundKey)

- **AddRoundKey** proceeds one column at a time.
- **AddRoundKey** adds a round key word with each state column matrix; the operation in **AddRoundKey** is matrix addition.
- The **AddRoundKey** transformation is the inverse of itself.



AES Encryption Round



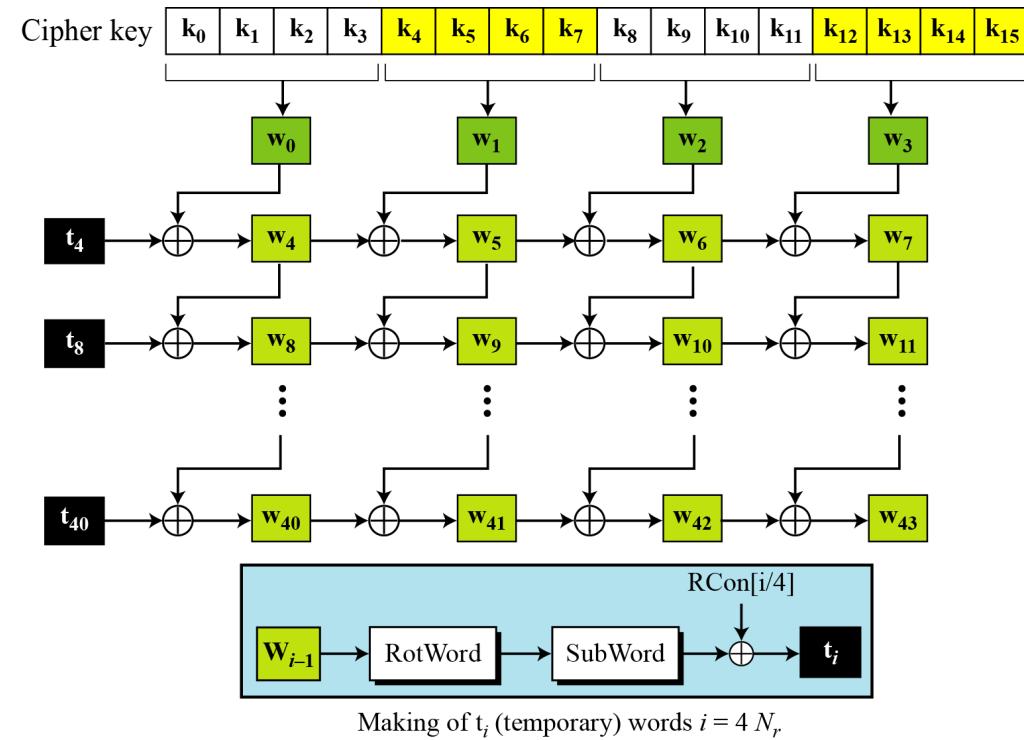
AES Key Expansion

- To create round keys for each round, 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.

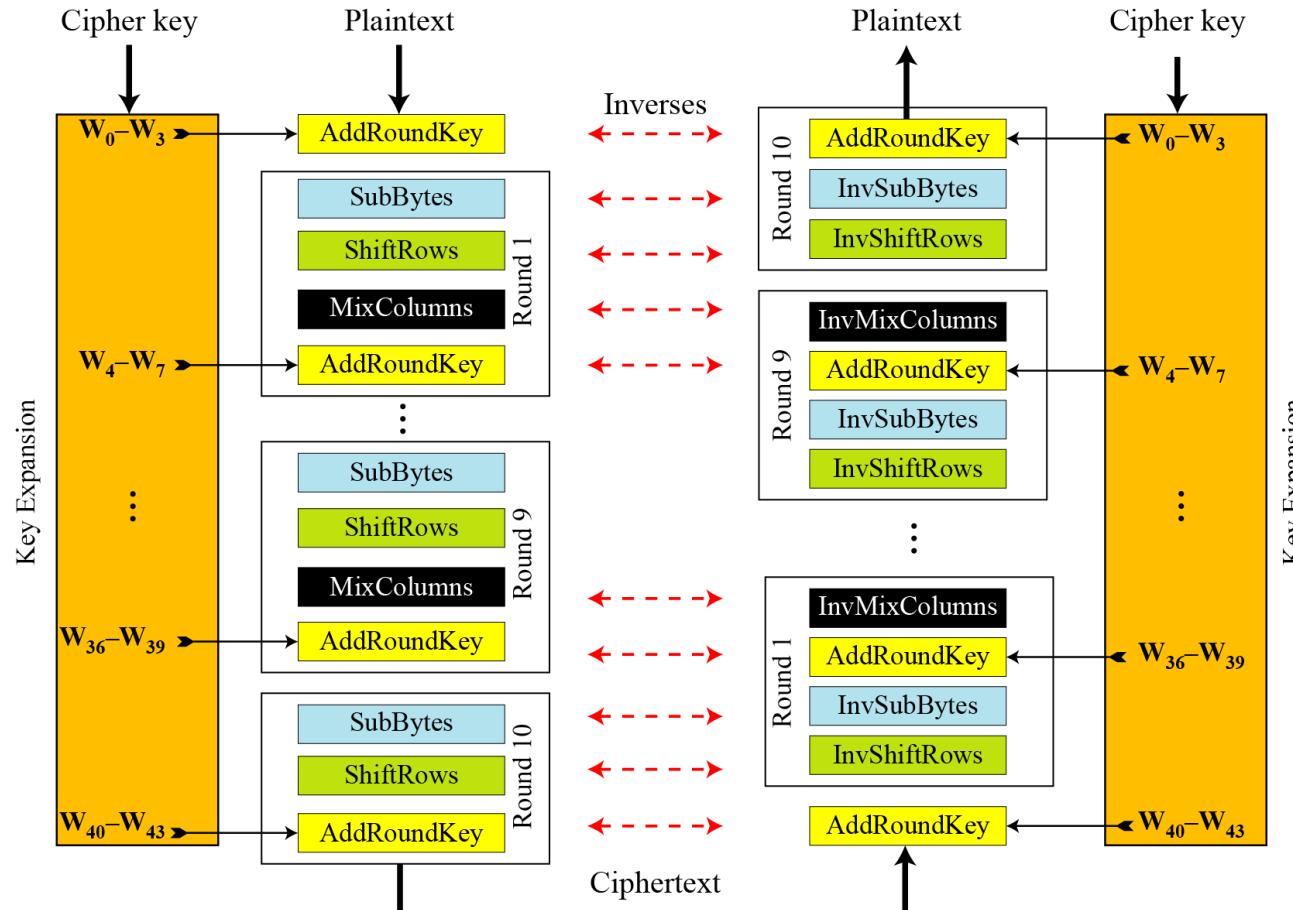
<i>Round</i>	<i>Words</i>			
Pre-round	w_0	w_1	w_2	w_3
1	w_4	w_5	w_6	w_7
2	w_8	w_9	w_{10}	w_{11}
...	...			
N_r	w_{4N_r}	w_{4N_r+1}	w_{4N_r+2}	w_{4N_r+3}

AES Key Expansion (Cont.)

Round	Constant (RCon)	Round	Constant (RCon)
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



AES Ciphers and inverse ciphers of the original design



AES Analysis

- AES was designed after DES. Most of the known attacks on DES were already tested on AES.
- **Brute-Force Attack:** AES is more secure than DES due to the larger-size key.
- **Statistical Attacks:** Numerous tests have failed to do statistical analysis of the ciphertext.
- **Differential and Linear Attacks:** There are no differential and linear attacks on AES yet.
- **Implementation:** AES can be implemented in software, hardware, and firmware. The implementation can use table lookup process or routines that use a well-defined algebraic structure.
- **Simplicity and Cost:** The algorithms used in AES are so simple that they can be easily implemented using cheap processors and a minimum amount of memory.

Standard Block-Cipher Modes of Operations

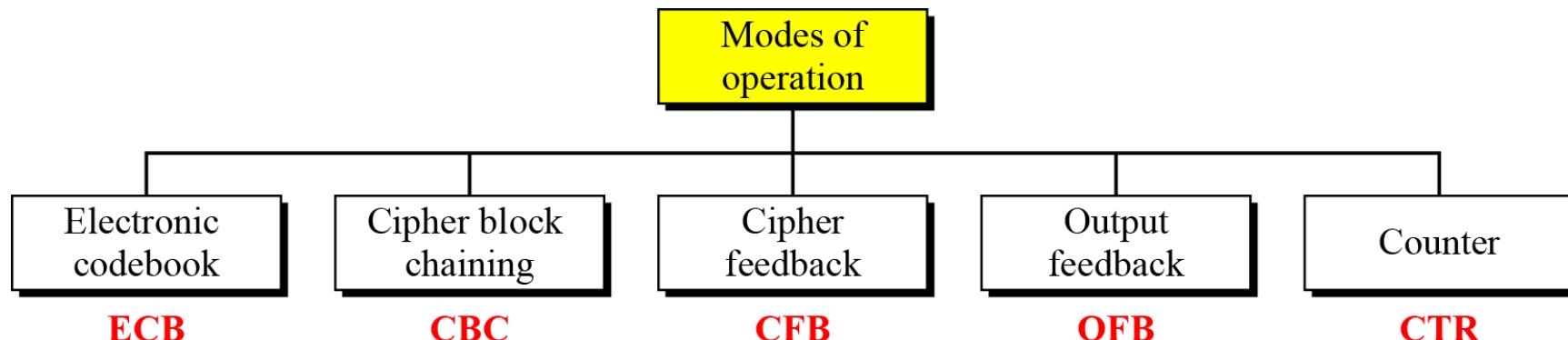
Modes of operation

- Symmetric-key encipherment can be done using modern block ciphers. Modes of operation have been devised to encipher text of any size employing either DES or AES.
- Let I be the block size of a given block cipher ($I = 64$ in DES, $I = 128$ in AES).
- Let M be a plaintext string. Divide M into a sequence of blocks:

$$M = M_1 \ M_2, \dots \ M_k$$

such that the size of each block M_i is I (padding the last block if necessary)

- There are several methods to encrypt M , which are referred to as block-cipher modes of operations



Electronic Codebook (ECB) Mode

- The simplest mode of operation is called the electronic codebook (ECB) mode.

Encryption: $C_i = E_K(P_i)$

Decryption: $P_i = D_K(C_i)$

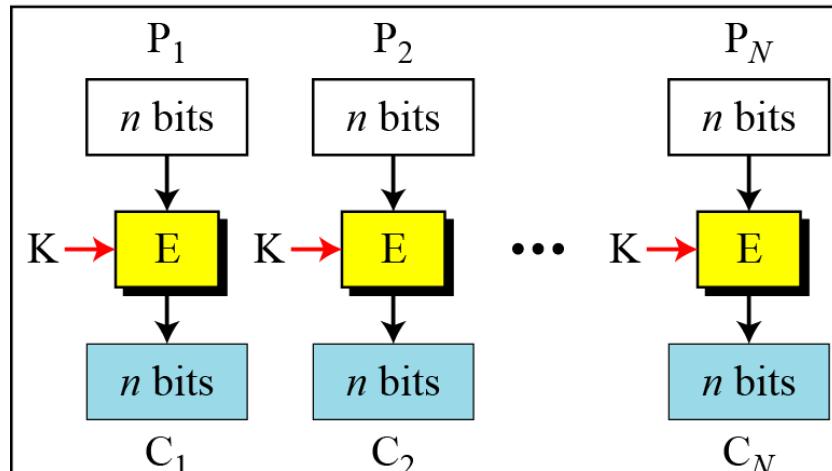
E: Encryption

P_i : Plaintext block i

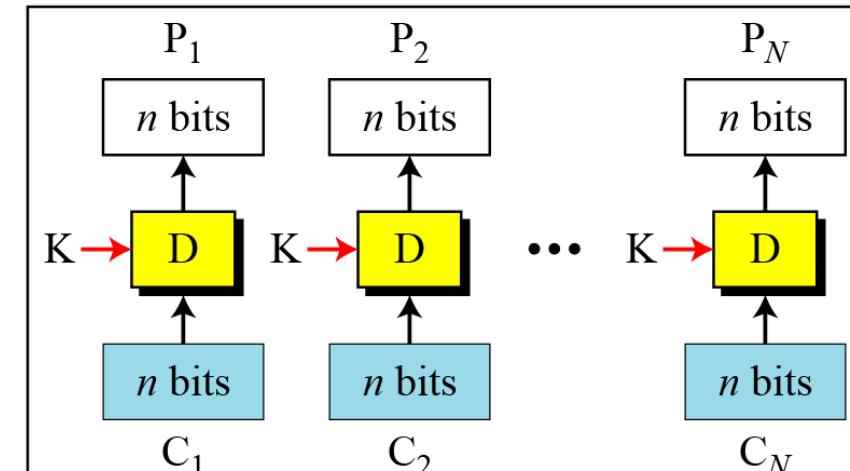
K: Secret key

D: Decryption

C_i : Ciphertext block i



Encryption



Decryption

Electronic Codebook (ECB) Mode (Cont.)

Security Issues

- Patterns at the block level are preserved
- The block independency creates opportunities for Eve to exchange some ciphertext blocks without knowing the key.

Applications

- The ECB mode is ***not recommended*** for encryption of messages ***more than one block***.
- One area where the independency of the ciphertext block is useful is where records need to be encrypted before they are stored in a database or decrypted before they are retrieved....***Access to the database can be random***.
- Another advantage of this mode is that we can use ***parallel processing*** if we need to create a very huge encrypted database.

Cipher Block Chaining (CBC) Mode

- In CBC mode, each *plaintext block* is *XOR* with the *previous ciphertext block* before being encrypted.

Encryption:

$$C_0 = \text{IV}$$

$$C_i = E_K(P_i \oplus C_{i-1})$$

Decryption:

$$C_0 = \text{IV}$$

$$P_i = D_K(C_i) \oplus C_{i-1}$$

E: Encryption

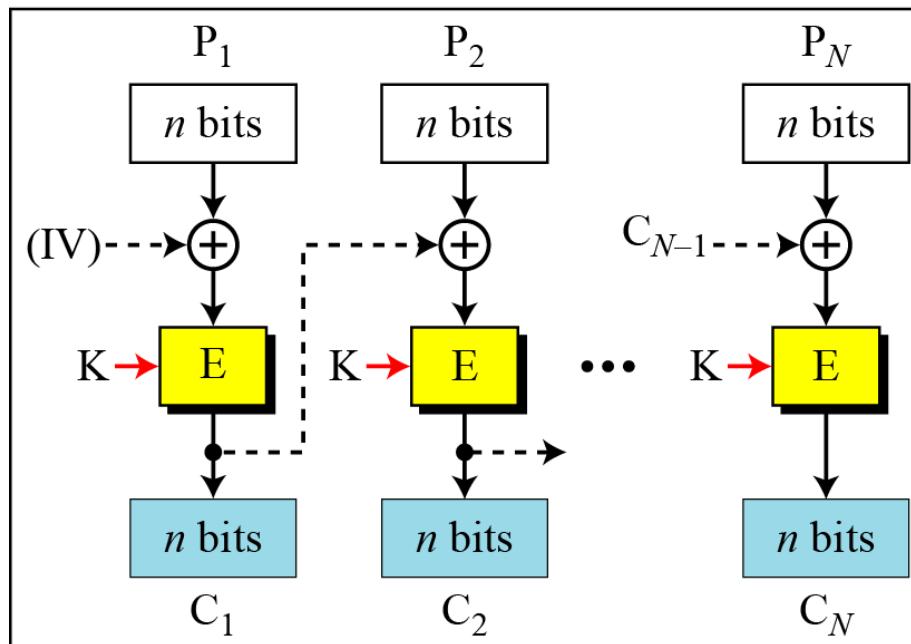
P_i : Plaintext block i

K: Secret key

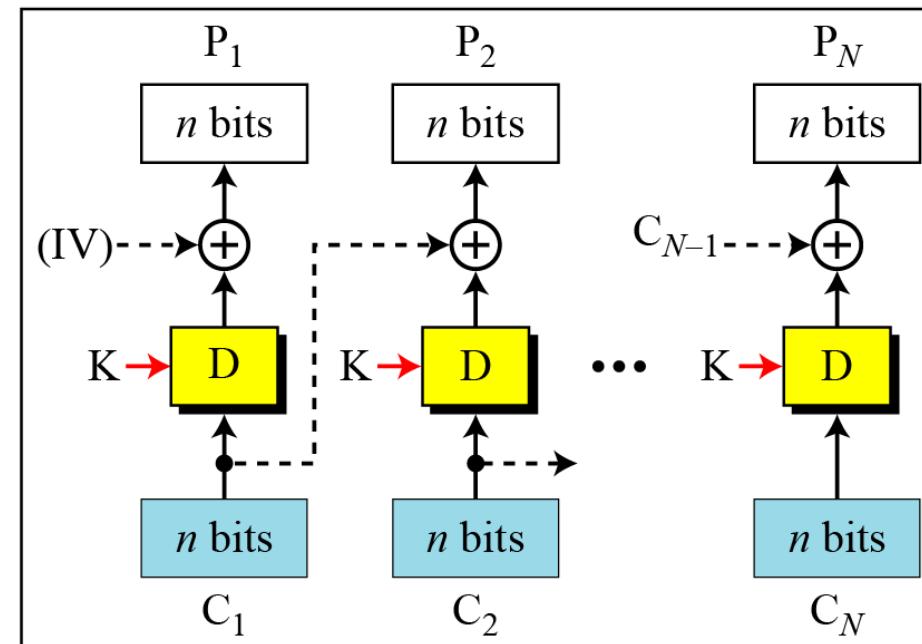
D : Decryption

C_i : Ciphertext block i

IV: Initial vector (C_0)



Encryption



Decryption

CBC Mode (Cont.)

Initialization Vector (IV)

- The initialization vector (**IV**) should be known by the sender and the receiver.

Security Issues

- Patterns at the block level are not preserved.
- However, if the first **M** blocks in two different messages are equal, they are enciphered into equal blocks unless different **IVs** are used. Hence, recommend the use of timestamp as an **IV**.

Applications

- Parallel processing is not possible.
- CBC mode is not used to encrypt and decrypt random-access files records because of the need to access the previous records.
- CBC mode is used for authentication.

Cipher Feedback (CFB) Mode

- In some situations, we *need to use DES or AES as secure ciphers*, but the plaintext or ciphertext block sizes are to be smaller.
- The relation between plaintext and ciphertext blocks is shown below:

Encryption: $C_i = P_i \oplus \text{SelectLeft}_r \{ E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1}] \}$

Decryption: $P_i = C_i \oplus \text{SelectLeft}_r \{ E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1}] \}$

E : Encryption

P_i : Plaintext block i

K: Secret key

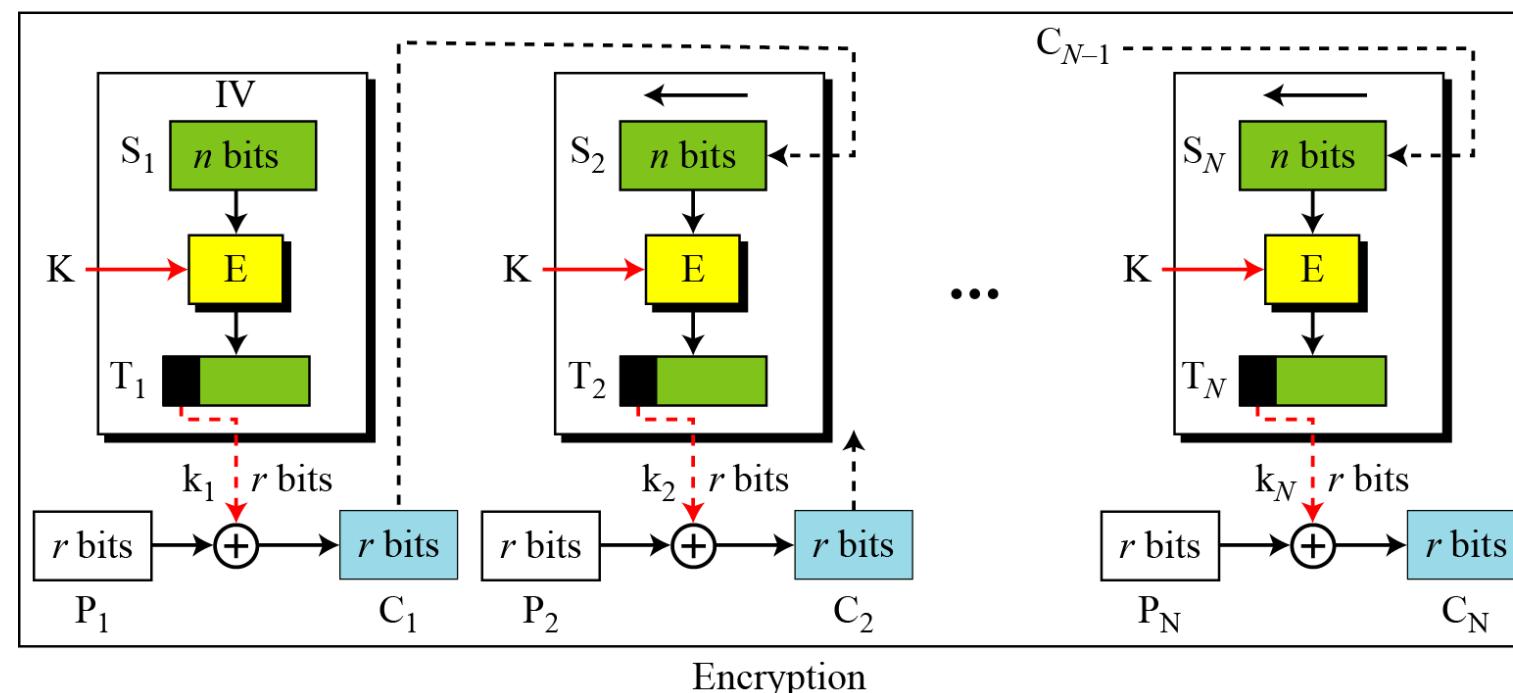
D : Decryption

C_i : Ciphertext block i

IV: Initial vector (S_1)

S_i : Shift register

T_i : Temporary register



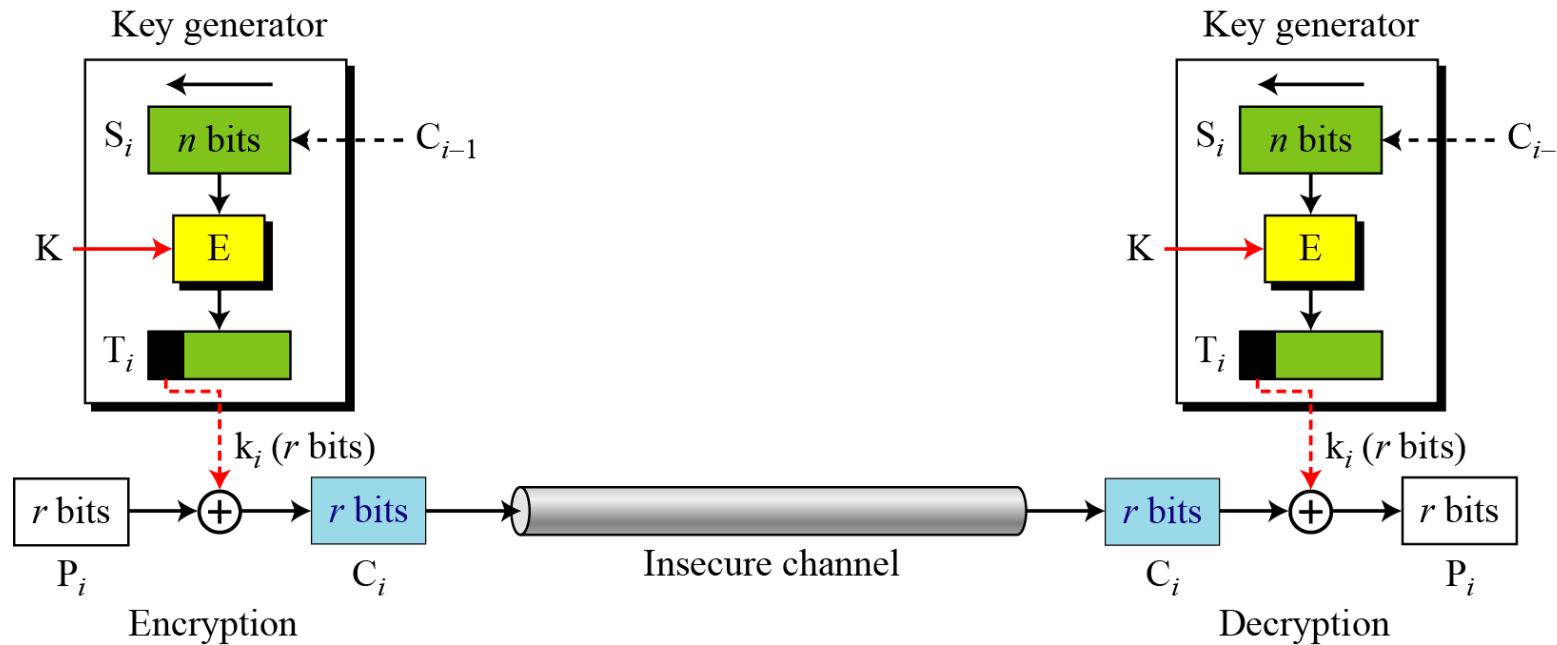
CFB Mode (Cont.)

Advantages

- This mode **does not need padding** because the size of the block r , is normally chosen to fit the data unit to be encrypted (a character for example).
- The system **does not have to wait until it has received a large block of data** (64 or 128 bits) before starting the encryption.

Disadvantages

- CFB is **less efficient** than CBC and ECB because it needs to apply the encryption function for each small block of size r .



Output Feedback (OFB) Mode

- In this mode *each bit in the ciphertext is independent of the previous bit or bits.*

E : Encryption

P_i : Plaintext block i

K: Secret key

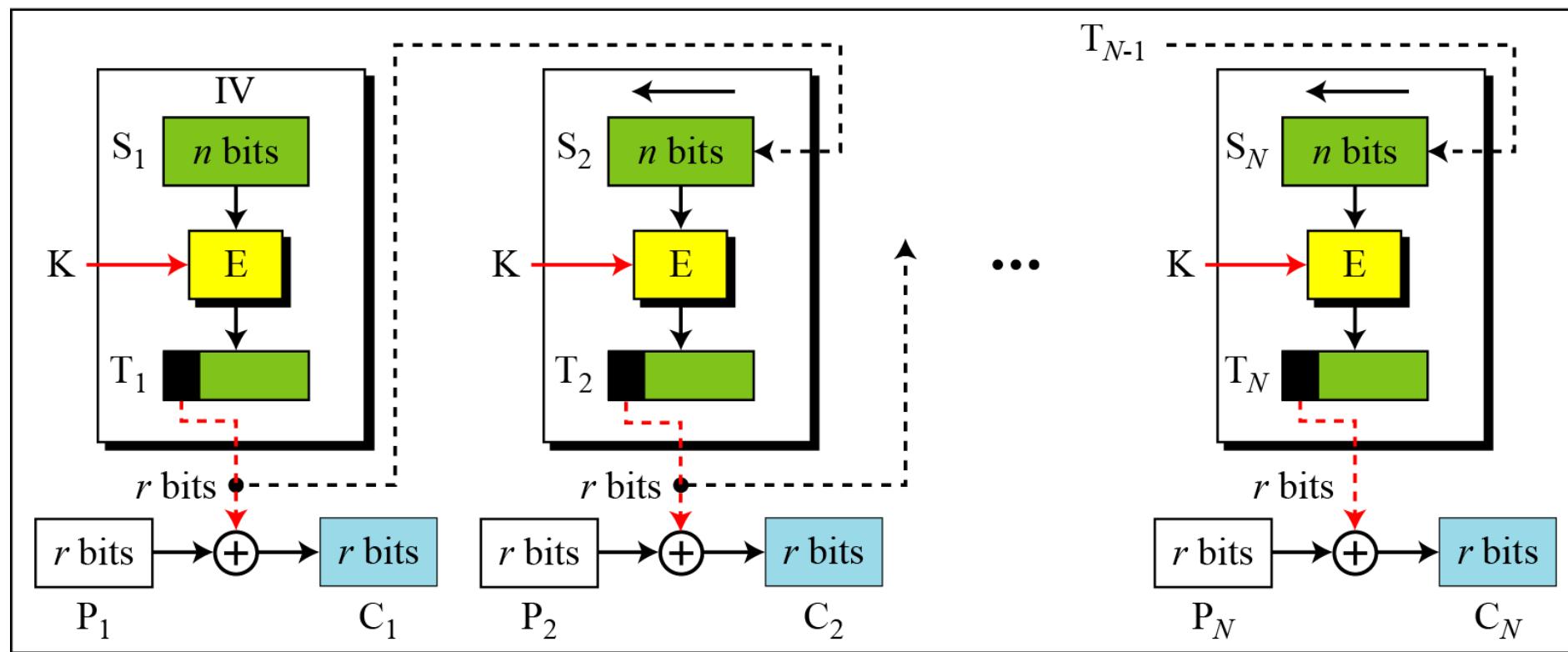
D : Decryption

C_i : Ciphertext block i

IV: Initial vector (S_1)

S_i : Shift register

T_i : Temporary register

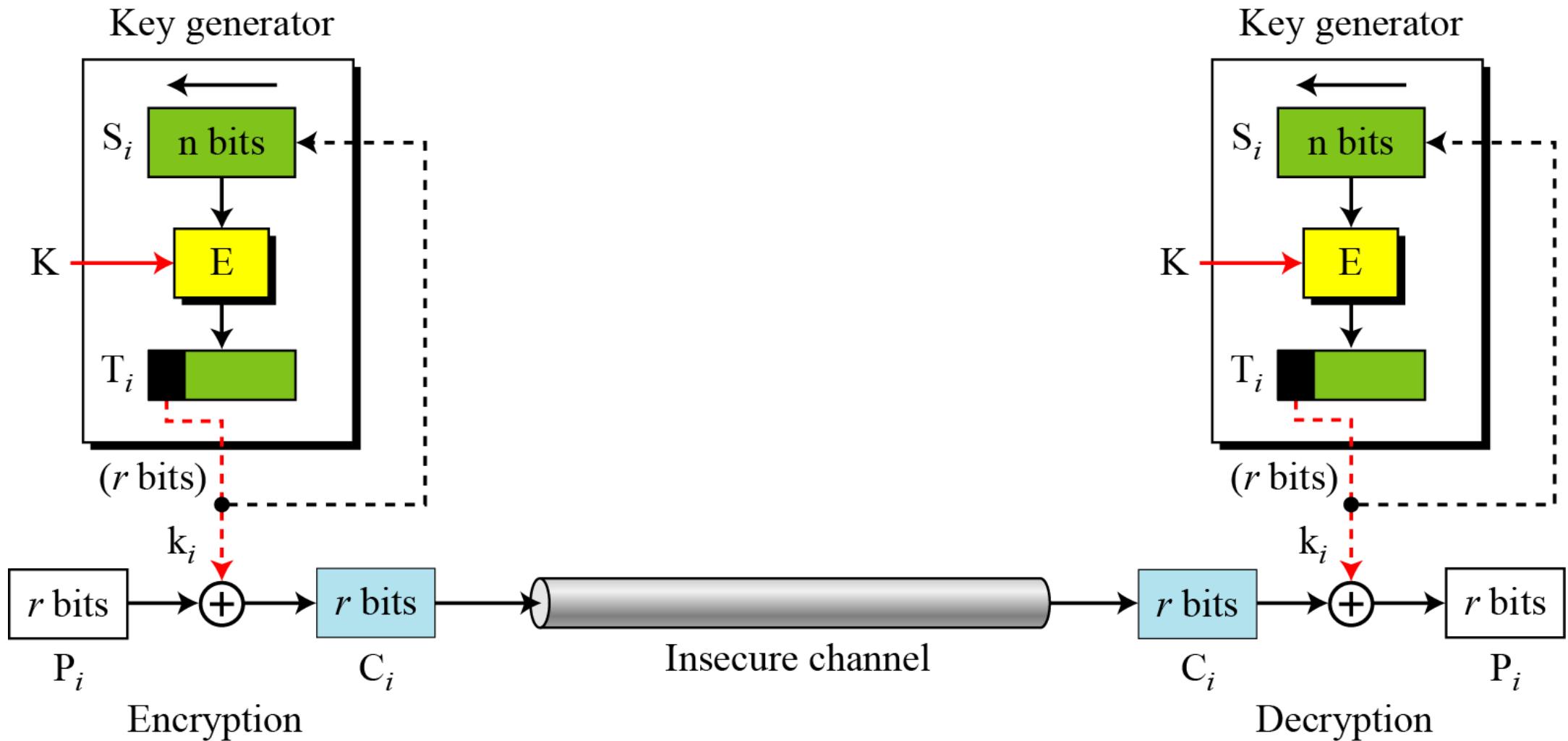


Encryption

Output Feedback (OFB) Mode (Cont.)

Security Issues

- The patterns are not preserved.



Counter (CTR) Mode

- In the counter (CTR) mode, there is no feedback. The pseudo-randomness in the key stream is achieved using a counter.

E : Encryption

P_i : Plaintext block i

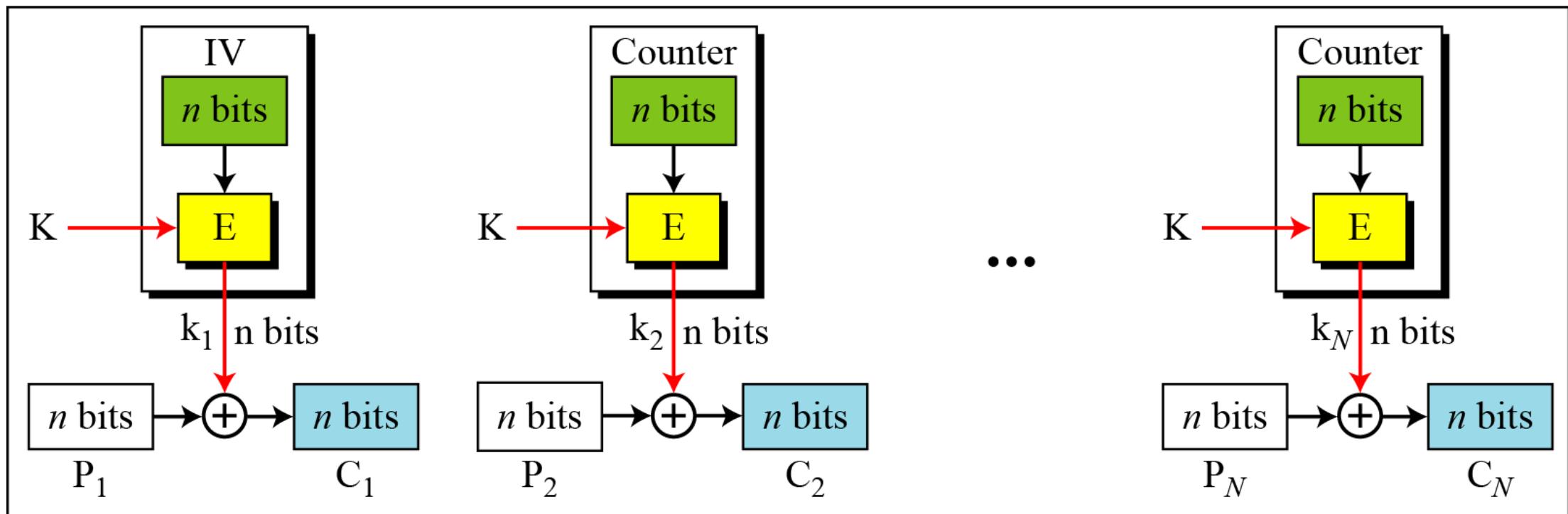
K : Secret key

IV: Initialization vector

C_i : Ciphertext block i

k_i : Encryption key i

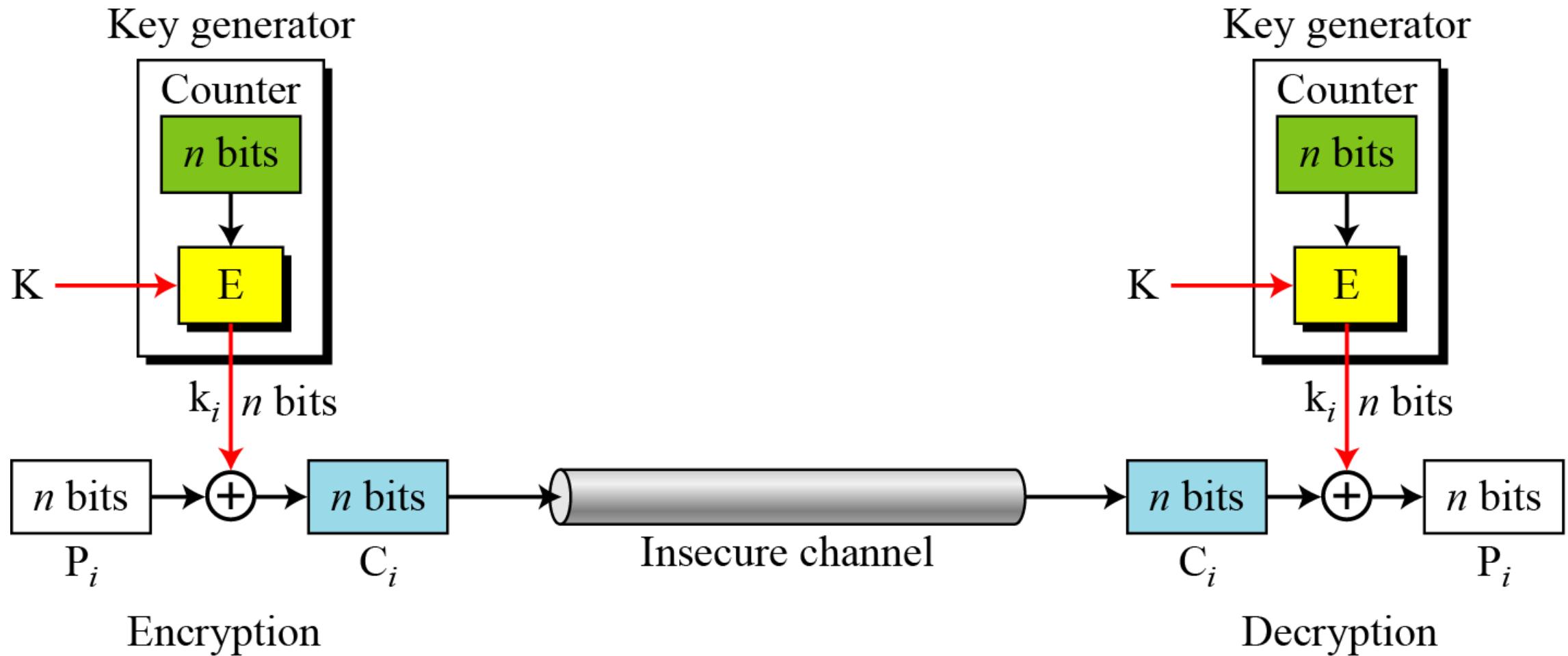
The counter is incremented for each block.



Encryption

Counter (CTR) Mode (Cont.)

- Counter (CTR) mode as a stream cipher



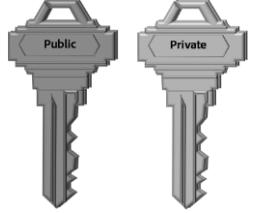
CTR Mode (Cont.)

- CTR creates ***n-bit*** blocks that are **independent from each other**; they **depend only on the value of the counter**.
- CTR, like ECB mode, **can be used to encrypt and decrypt random access files** if the value of the counter can be related to the record number in the file.
- Used in applications requiring faster encryption speed

Comparison of Different Modes

<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

RSA



Overview

- RSA scheme is the most widely accepted and implemented general-purpose approach to public-key encryption.
- RSA is a cipher in which plaintext and ciphertext are integers between 0 and $n - 1$ for some n .
- A typical size for n is 1024 bits or 309 decimal digits.
- So, n is less than 2^{1024} .



RSA Algorithm

- RSA makes use of exponential expressions.
- Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C .

$$C = M^e \bmod n$$

$$M = C^d \bmod n$$

- Sender knows value of e and only receiver knows value of d .
- However, both sender and receiver must know the value of n .



RSA Algorithm (Cont.)

- RSA is a public key encryption algorithm with the following:
 - *Public key of PU = {e, n}*
 - *Private key of PR = {d, n}*
- For this algorithm to be satisfactory for public-key encryption, following requirements must be met:
 1. It is possible to find values of **e**, **d** and **n** such that $M^{ed} \bmod n = M$, for all $M < n$.
 2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.
 3. It is infeasible to determine **d** given **e** and **n**.

RSA Algorithm (Cont.)

- RSA algorithm is based on a fact that finding factors of large composite numbers is difficult *when the integers are prime numbers.*

Following are required in RSA algorithm:

- p, q are two prime numbers *(private, chosen)*
- $n = pq$ *(public, calculated)*
- $\varphi(pq) = (p - 1)(q - 1)$.
- e , such that $\gcd(\varphi(n), e) = 1$,
where $1 < e < \varphi(n)$ *(public, chosen)*
- $d \equiv e^{-1} \pmod{\varphi(n)}$ *(private, calculated)*

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

RSA Example

Example of RSA algorithm:

- Select two prime numbers, $p = 17$ and $q = 11$
- Calculate $n = pq = 17 \times 11 = 187$
- Calculate $\varphi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$
- Select e such that e is relatively prime to $\varphi(n) = 160$ and $1 < e < \varphi(n)$.
We choose $e = 7$
- Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$
- Value of $d = 23$ (*calculated using extended Euclid's algorithm*)
- Resulting keys are $PU = \{7, 187\}$ and $PR = \{23, 187\}$

RSA Example (Cont.)

Example of RSA algorithm (Cont.):

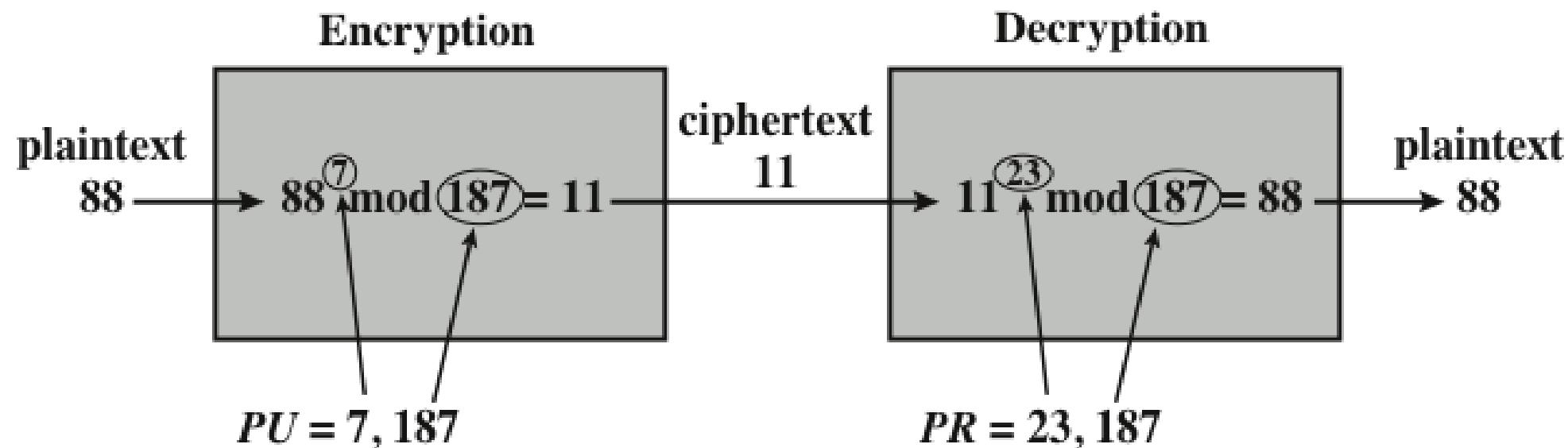
- Use the generated keys for a plaintext input of $M = 88$
- For encryption, we calculate $C = 88^7 \text{ mod } 187$
- By exploiting properties of modular arithmetic, we have:
 - $88^7 \text{ mod } 187 = [(88^4 \text{ mod } 187) \times (88^2 \text{ mod } 187) \times (88^1 \text{ mod } 187)] \text{ mod } 187$
 - $88^1 \text{ mod } 187 = 88$
 - $88^2 \text{ mod } 187 = 7744 \text{ mod } 187 = 77$
 - $88^4 \text{ mod } 187 = 59,969,536 \text{ mod } 187 = 132$
 - So, $88^7 \text{ mod } 187 = (88 \times 77 \times 132) \text{ mod } 187 = 894,432 \text{ mod } 187 = 11$
 - So, **C = 11**

RSA Example (Cont.)

Example of RSA algorithm (Cont.):

- For decryption, we calculate $M = 11^{23} \bmod 187$:
 - $11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$
 - $11^1 \bmod 187 = 11$
 - $11^2 \bmod 187 = 121$
 - $11^4 \bmod 187 = 14,641 \bmod 187 = 55$
 - $11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$
 - $11^{23} \bmod 187 = (11 * 121 * 55 * 33 * 33) \bmod 187 = 79,720,245 \bmod 187 = 88$
 - So, **M = 88**

RSA Example (Cont.)



Thank You!