AES

<**A**dvanced **E**ncryption **S**tandard>

AES Features

- Designed to be efficient in both hardware and software across a variety of platforms.
- Not a Feistel Network
  - ✓ Iterated block cipher (like DES)
  - ✓ Not a Feistel cipher (unlike DES)
- "Secure forever" – Shamir
- Rijndael proposed
  - a variable block size, 128,192, 256-bits,
  - key size of 128-, 192-, or 256-bits.
  - Variable number of rounds (10, 12, 14):
    - 10 if B = K = 128 bits
    - 12 if either B or K is 192 and the other is ≤ 192
    - 14 if either B or K is 256 bits

Note

- AES는 128 비트 평문을 128 비트 암호문으로 출력하는 알고리즘으로 non-Feistel 알고리즘에 속한다. 10, 12, 14 라운드를 사용하며, 각 라운드에 대응하는 키 크기는 128, 192, 256 비트이다.
- AES는 128, 192, 256 비트 키를 사용하고 키 크기에 따라 각각 10, 12, 14 라운드를 갖는 3가지 버전이 있다. 그러나 마스터 키의 크기가 달라도 라운드 키는 모두 128 비트이다.

AES Overview

- Definition: State→ 4X4 array of bytes
  - ✓ 128 bits = 16 bytes
- Variable number of rounds (10, 12, 14):
  - ✓ 10 if K is 128 bits
  - ✓ 12 if K is 192 bites
  - ✓ 14 if K is 256 bits
- 128-bit round key used for each round:
  - ✓ 128 bits = 16 bytes = 4 words
  - ✓ needs Nr+1 round keys for Nr rounds
    - needs 44 words for 128-bit key (10 rounds)

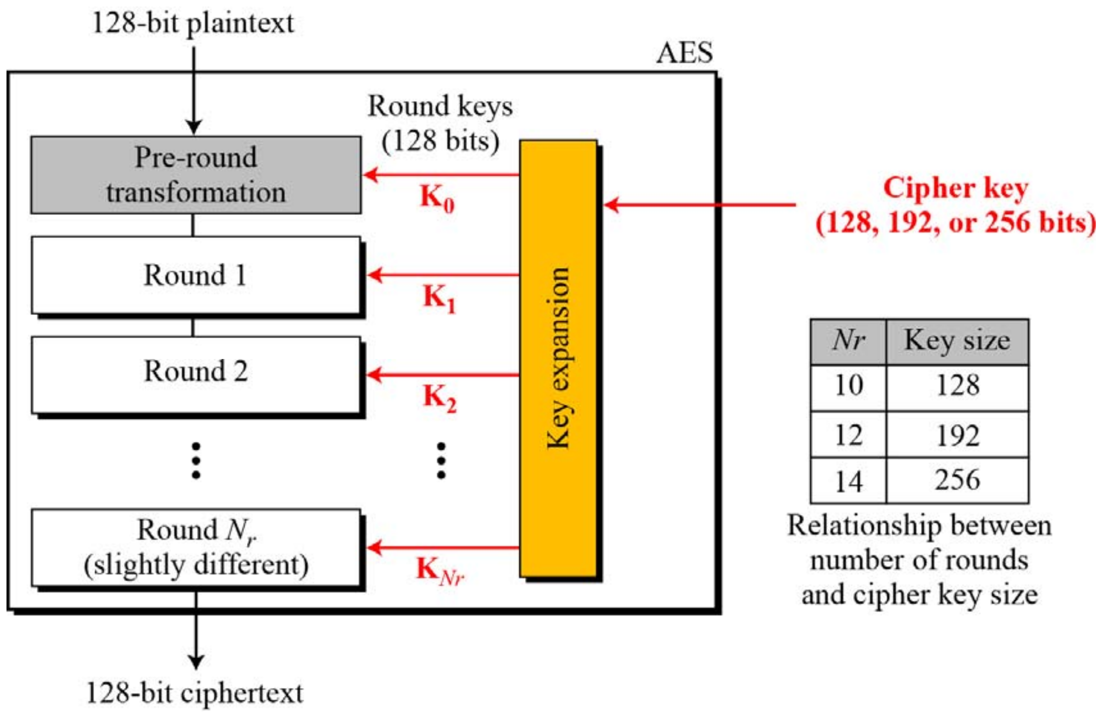


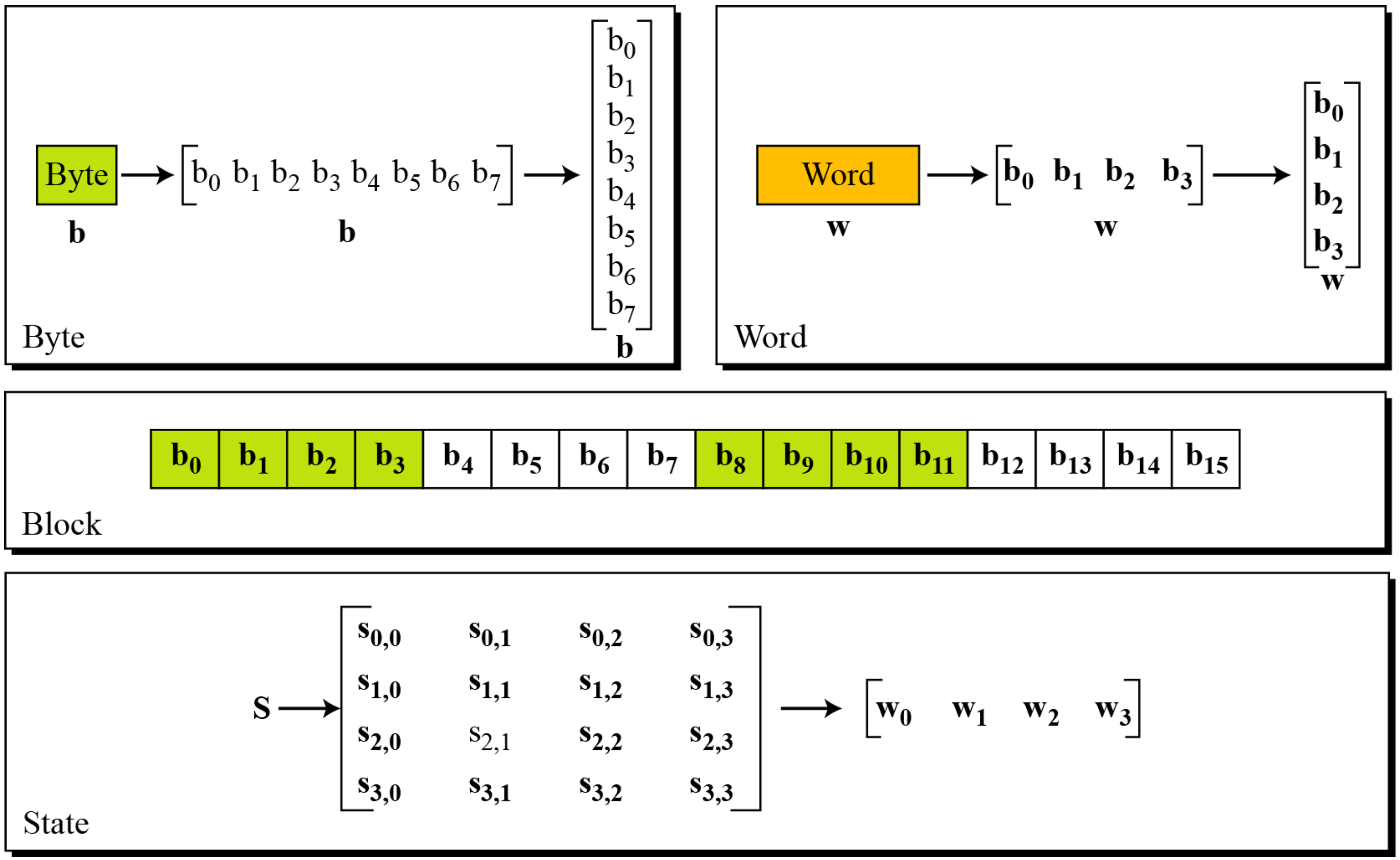
Figure 1. General design of AES encryption cipher

AES Overview

- Each round uses 4 functions (in 3 "layers")
  - ✓ 4 functions: 1 permutation and 3 substitutions
  - ✓ 3 layers: Linear, Nonlinear and Key addition
- Permutation
  - ✓ Linear mixing layer: ShiftRow (State)
- Substitutions
  - ✓ Nonlinear layer: ByteSub (State, S-box)
  - ✓ Nonlinear layer: MixColumn (State)
  - ✓ Key addition layer: AddRoundKey (State, KeyNr)

AES 암호 구조도

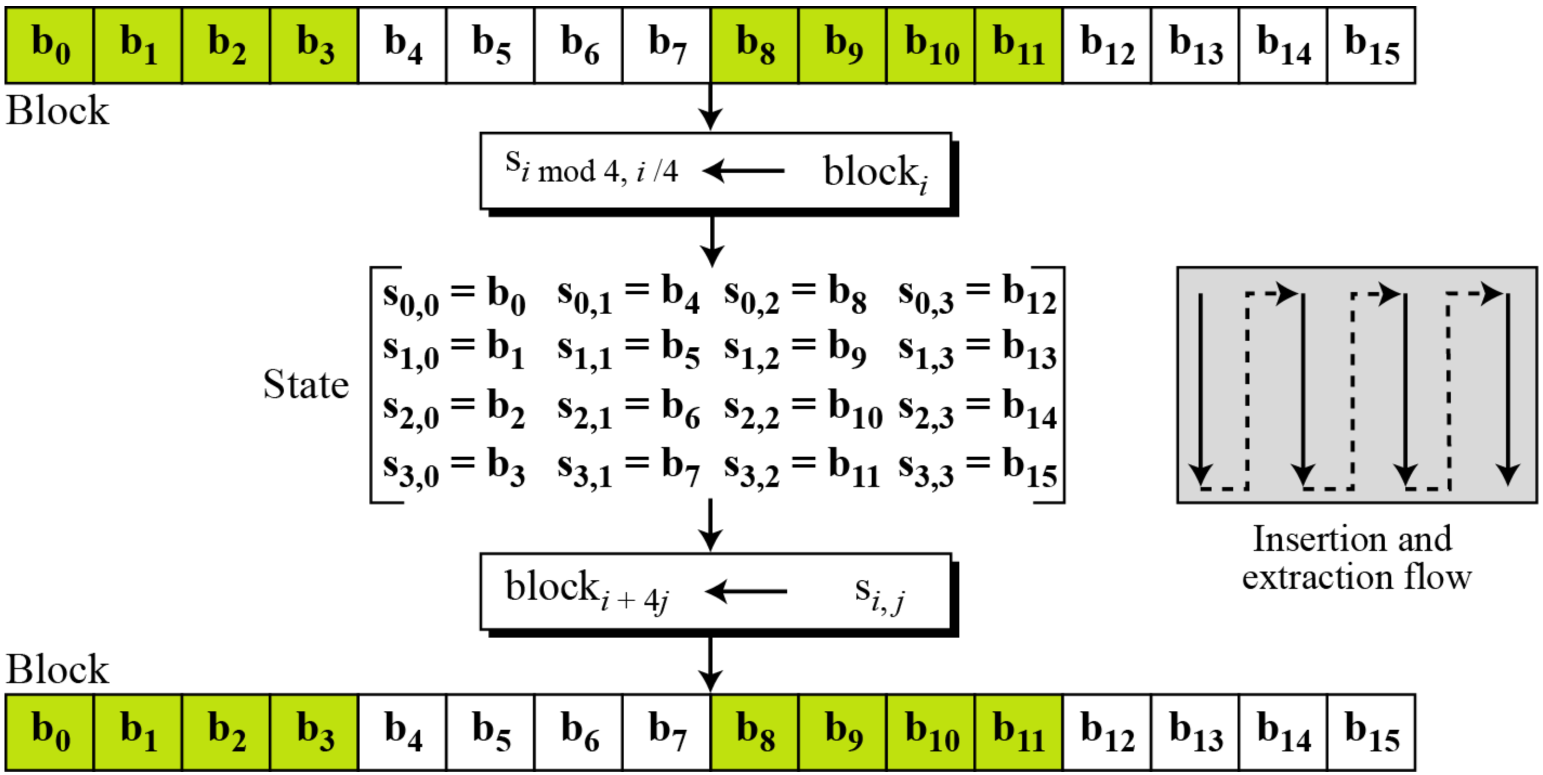
Note



AES 암호 구조도

Note

AES에서 State와 Block의 변환



AES 암호 구조도

Note

Plaintext를 State로 변환

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

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

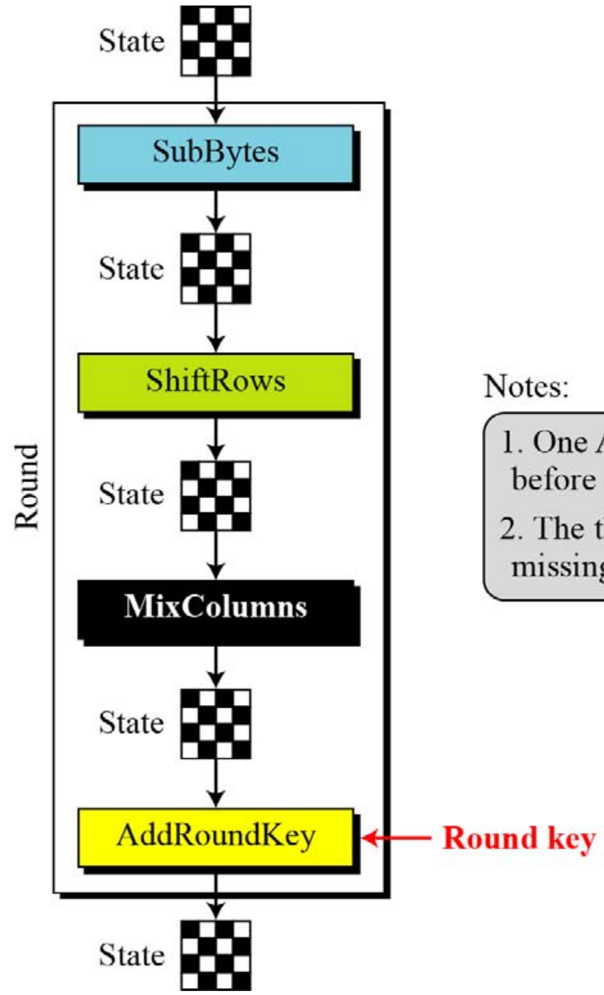
AES : High-Level Description

- State: 4 X 4 array of bytes: 128 bits = 16 bytes

```
State = X
AddRoundKey(State, Key0)                    (op1)

for r = 1 to Nr - 1
        SubBytes(State, S-box)              (op2)
        ShiftRows(State)                    (op3)
        MixColumns(State)                   (op4)
        AddRoundKey(State, KeyNr)
endfor

SubBytes(State, S-box)
ShiftRows(State)
AddRoundKey(State, KeyNr)
Y = State
```

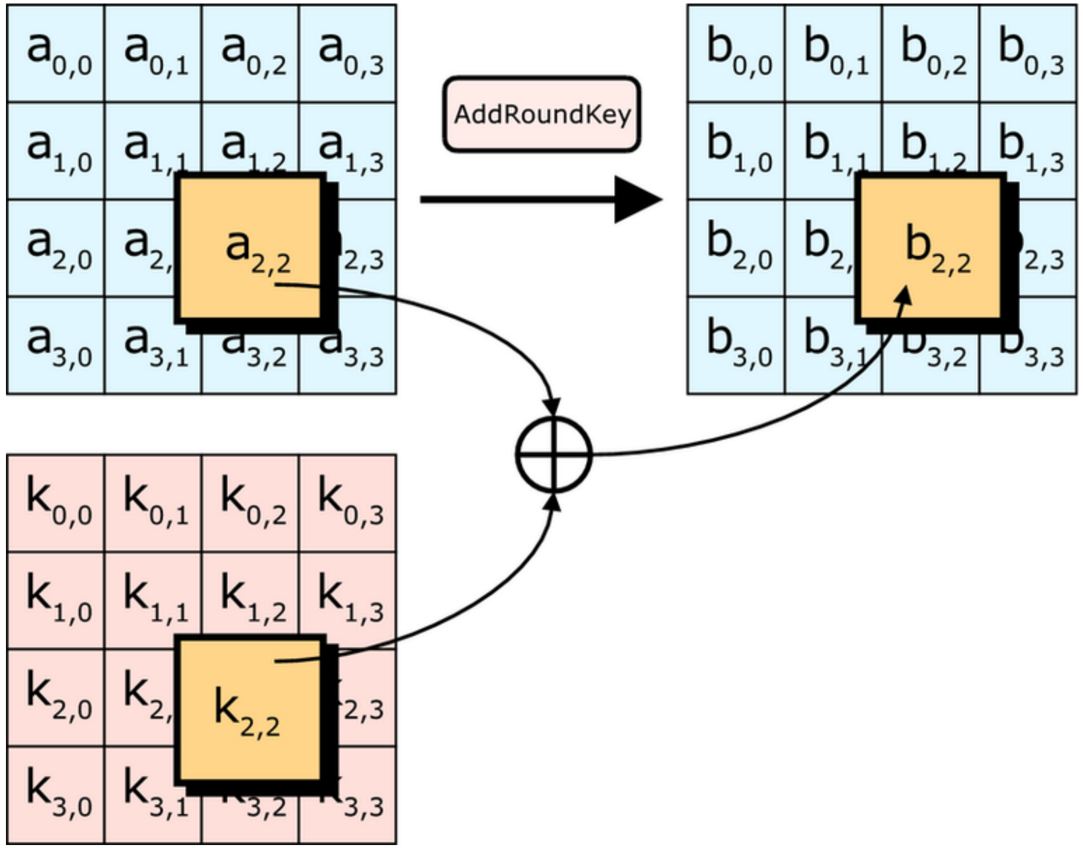


Notes:

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

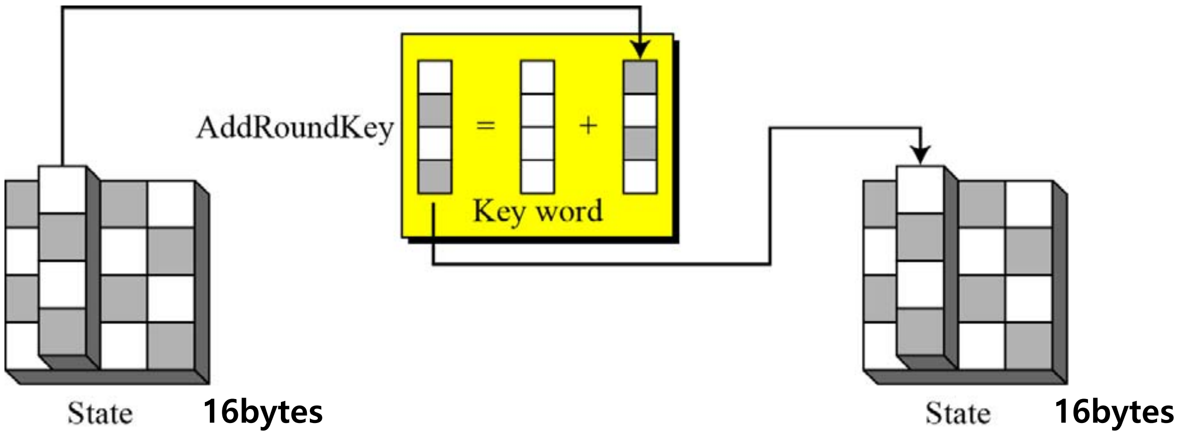
AES AddRoundKey

- XOR subkey with block: Assume 128-bits block



- RoundKey (subkey) determined by **key schedule** algorithm
- AES Key schedule : https://en.wikipedia.org/wiki/AES_key_schedule (부록참조)

AES AddRoundKey

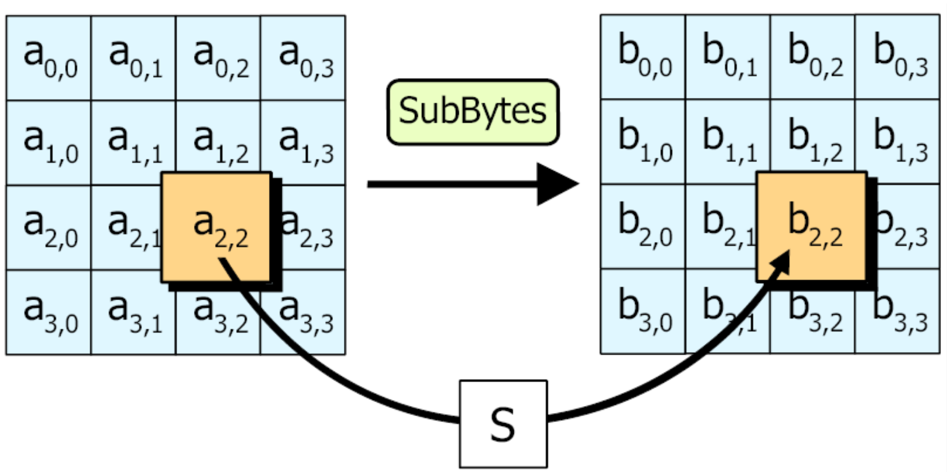


Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

```
AddRoundKey (S)
{
     for (c = 0 to 3)
          s_c ← s_c ⊕ w_{round + 4c}
}
```

AES SubBytes (or ByteSub)

- Assume 128 bit block, i.e. 4 x 4 bytes

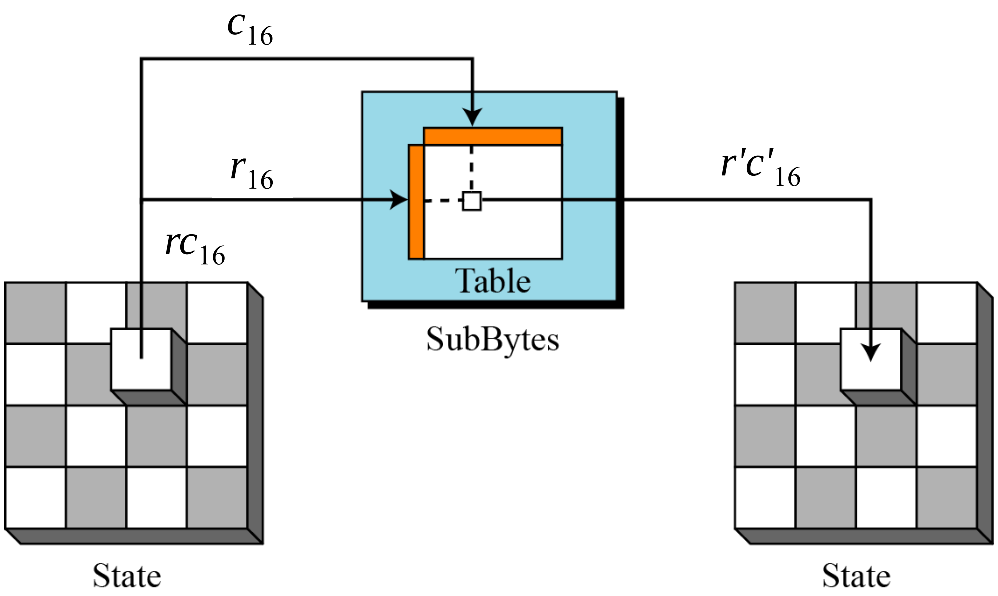


- SubByte is AES's "S-box"
- Can be viewed as nonlinear (but invertible) composition of two math operations

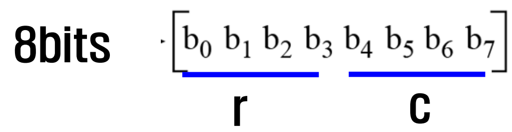
- SubByte is the Byte substitution using non-linear S-Box (independently on each byte).
- S-box is represented as a 16x16 array, rows and columns indexed by hexadecimal bits (16개 독립된 바이트 단위의 변환 수행)
- 8 bits replaced as follows:
  - ✓ 8 bits defines a hexadecimal number (r,c),
  - ✓ then (sr,sc) = binary(Sbox(r, c))

8bits $\left[\underbrace{b_0\ b_1\ b_2\ b_3}_{r}\ \underbrace{b_4\ b_5\ b_6\ b_7}_{c}\right]$

AES SubBytes (or ByteSub)



SubBytes 변환



AES "S-box"

- Example: hexa "53" is replaced with hexa "ED" (sr,sc) = binary(Sbox(r, c))

[0101 0011]

Last 4 bits of input (c)

First 4 bits of input (r)

	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	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

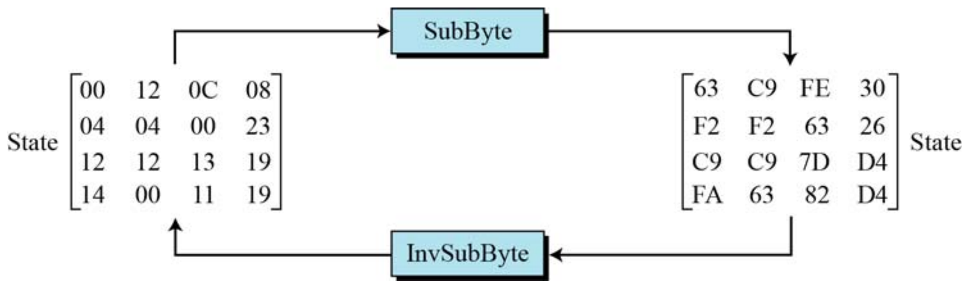
AES InvSubBytes

Note

- InvSubBytes transformation table

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

- SubBytes 변환 예



- Example: hexa "ED" is replaced with hexa "53"

4. Block Ciphers – AES

GF(2^8) 이용한 변환

Note

Finite Field Theory 참조

- GF(2^8) 를 이용한 변환 방법 (Transformation Using the GF(2^8) Field)
- AES는 그림에서 보여주는 것처럼 기약 다항식 (Irreducible Polynomial) ($x^8 + x^4 + x^3 + x + 1$) 를 가진 체 GF(2^8) 를 이용하여 대수적인 변환으로 S-박스를 정의할 수 있다.

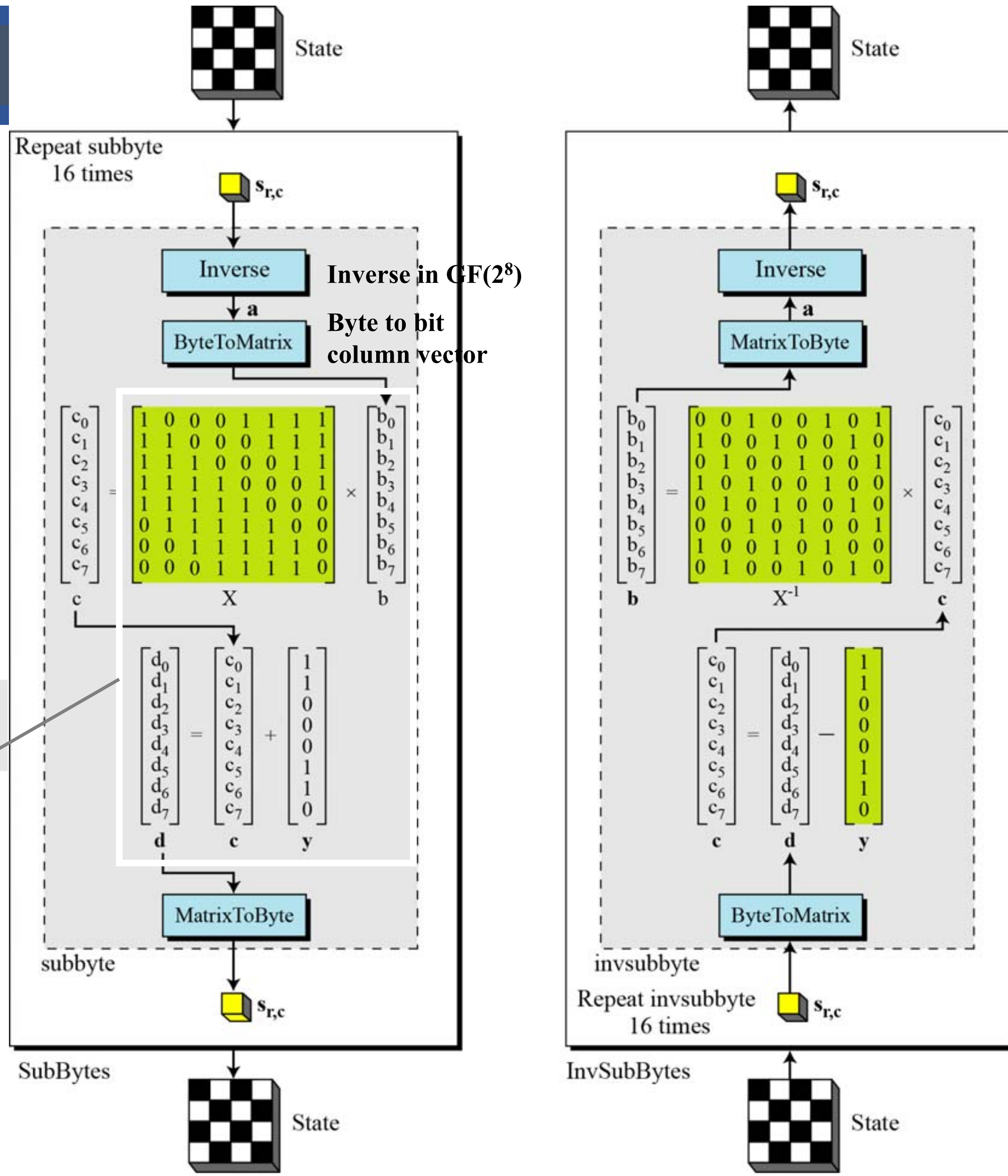
ByteToMartix (0x63)

subbyte: $\rightarrow$ $\mathbf{d} = \mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y}$

invsubbyte: $\rightarrow$ $[\mathbf{X}^{-1}(\mathbf{d} \oplus \mathbf{y})]^{-1} = [\mathbf{X}^{-1}(\mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y} \oplus \mathbf{y})]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

- SubBytes와 InvSubBytes 과정
- **SubBytes 변환과 InvSubBytes 변환은 서로 역변환 관계이다.**



기약다항식: 더는 인수분해 할 수 없는 다항식

GF(2^8) 이용한 변환

Note

Example

16진수 값 0C를 *subbyte*를 통해 로 계산하는 과정을 보이고 역으로 *invsubbyte*를 통해 다시 0C로 계산하는 과정을 설명한다.

1. ***subbyte*** :
   a. *GF(2^8)* 필드에서 *0C*의 역원(multiplicative inverse)은 *B0* 이고 이것을 비트 단위로 표현하면 *b=(10110000)*이다.
   b. 행렬 *X*와 곱셈 연산 후 값은 *c=(10011101)* 이 된다.
   c. 이어서 XOR 연산 후의 값은 *d=(11111110)* 이 되며, 이는 *FE* 의 비트 단위 표현이다.

2. ***invsubbyte*** :
   a. XOR 연산 후의 값으로 *c=(10011101)*을 얻을 수 있다.
   b. 행렬 X^{-1}를 곱한 이후의 값은 *(10110000)* 또는 *B0* 가 된다.
   c. *B0* 의 역원은 *0C* 가 된다.

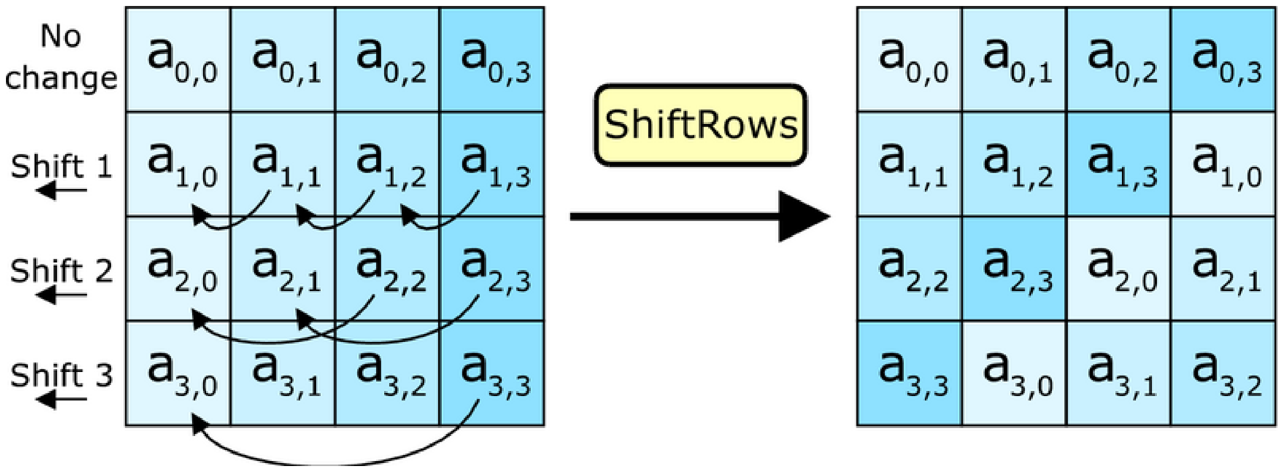
Algorithm 7.1 *Pseudocode for SubBytes transformation*

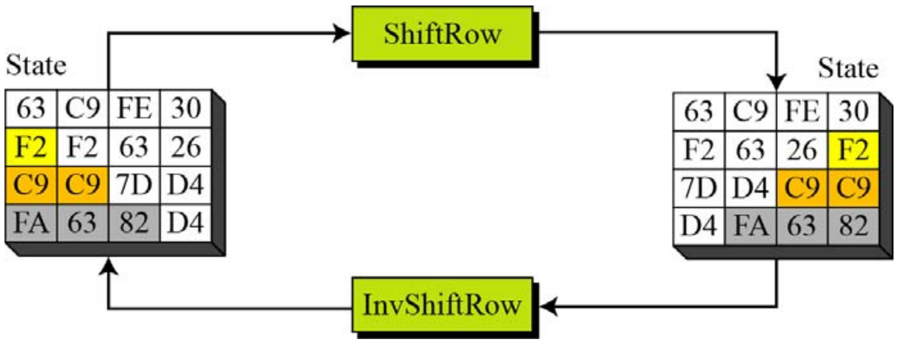
```
SubBytes (S)
{
   for (r = 0 to 3)
     for (c = 0 to 3)
          S_{r,c} = subbyte (S_{r,c})
}

subbyte (byte)
{
   a ← byte^{-1}          // Multiplicative inverse in GF(2^8) with inverse of 00 to be 00
    ByteToMatrix (a, b)
    for (i = 0 to 7)
    {
        c_i ← b_i ⊕ b_{(i+4)mod 8} ⊕ b_{(i+5)mod 8} ⊕ b_{(i+6)mod 8} ⊕ b_{(i+7)mod 8}
        d_i ← c_i ⊕ ByteToMatrix (0x63)
    }
    MatrixToByte (d, d)
    byte ← d
}
```

AES ShiftRow

- Cyclic shift rows





- ShiftRows는 암호화 과정에서 사용하고 왼쪽으로 순환이동을 수행한다
- InvShiftRows는 복호화 과정에서 사용하고 오른쪽으로 순환이동을 수행한다.
- ShiftRows 와 InvShiftRows 는 서로 역변환 관계이다.

AES ShiftRow

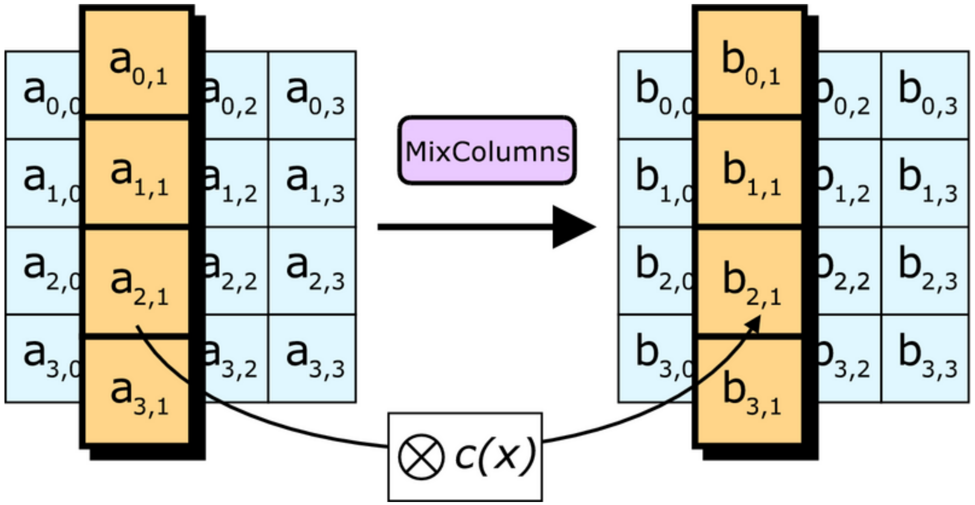
Algorithm 7.2 *Pseudocode for ShiftRows transformation*

```
ShiftRows (S)
{
    for (r = 1 to 3)
        shiftrow (s_r, r)                  // s_r is the rth row
}
```

```
shiftrow (row, n)                          // n is the number of bytes to be shifted
{
   CopyRow (row, t)                        // t is a temporary row
   for (c = 0 to 3)
        row_((c − n) mod 4) ← t_c
}
```

AES MixColumn

- Nonlinear, invertible operation applied to each column

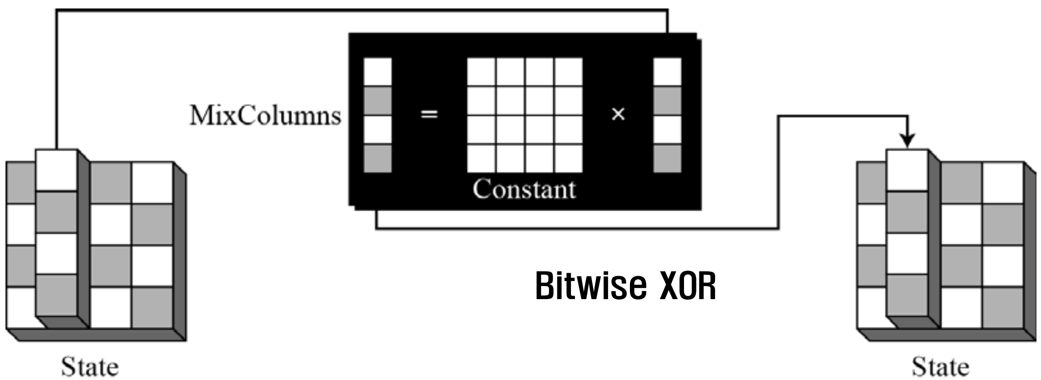


Bitwise XOR Constant matrix

- Implemented as a (big) lookup table

- MixColumn와 InvMixColumn에 사용하는 상수행렬

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



- MixColumns와 InvMixColumns는 서로 역변환 관계이다.

AES MixColumn

Algorithm 7.3 *Pseudocode for MixColumns transformation*

```
MixColumns (S)
{
    for (c = 0 to 3)
        mixcolumn (s_c)
}

mixcolumn (col)
{
   CopyColumn (col, t)              // t is a temporary column

   col_0 ← (0x02) • t_0 ⊕ (0x03 • t_1) ⊕ t_2 ⊕ t_3

   col_1 ← t_0 ⊕ (0x02) • t_1 ⊕ (0x03) • t_2 ⊕ t_3

   col_2 ← t_0 ⊕ t_1 ⊕ (0x02) • t_2 ⊕ (0x03) • t_3

   col_3 ← (0x03 • t_0) ⊕ t_1 ⊕ t_2 ⊕ (0x02) • t_3
}
```

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

AES Decryption

- To decrypt, process must be invertible
  - ✓ (InvAddRoudKey) Inverse of AddRoundKey is easy, since ⊕ is its own inverse
  - ✓ (InvMixColumn) MixColumn is invertible (inverse is also implemented as a lookup table)
  - ✓ (InvShiftRow) Inverse of ShiftRow is easy (cyclic shift the other direction)
  - ✓ (InvSubBytes) SubByte is invertible (inverse is also implemented as a lookup table)

AES Decryption Rationale

- Substitute Byte
  - ✓ To be resistant to known cryptanalytic attacks by making a low correlation between input bits and output bits.
- Shift Row
  - ✓ Note input and output are treated as State(4X4 array)
  - ✓ To move an individual byte from one column to another
- Mix Column
  - ✓ To ensure a good mixing the bytes of each column
- Add Round Key
  - ✓ To affect every bit of State
  - ✓ The complexity of the round key expansion ensure security

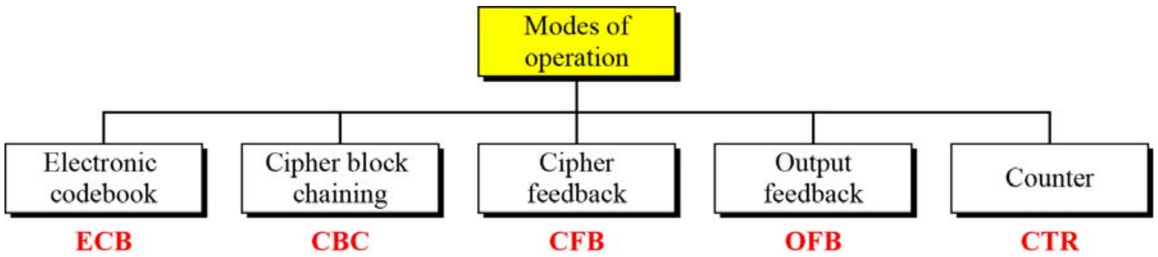
Block Cipher Mode

Symmetric cipher encryption

- Stream cipher is easy:
  - ✓ keystream is the same length as the plaintext and XOR
- How to encrypt multiple blocks?
  - ✓ A new key for each block?
    - As bad as (or worse than) a one-time pad!
  - ✓ Encrypt each block independently?
  - ✓ Make encryption depend on previous block(s), i.e., "chain" the blocks together?
  - ✓ How to handle partial blocks?

Modes of Operation

- Many encryption ways (modes of operation) for multiple block cipher – we discuss three
  - ✓ Electronic Codebook (ECB) mode
    - Obvious thing to do
    - Encrypt each block independently
    - There is a serious weakness
  - ✓ Cipher Block Chaining (CBC) mode
    - Chain the blocks together
    - More secure than ECB, virtually no extra work
  - ✓ Counter Mode (CTR) mode
    - Acts like a stream cipher
    - Popular for random access

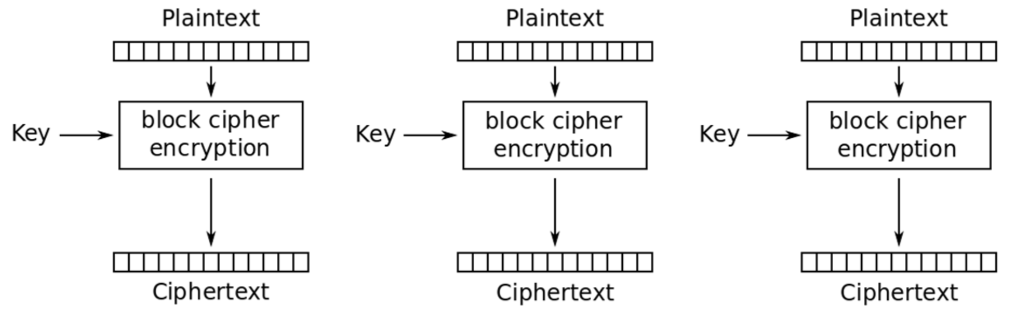


ECB(Electronic Codebook) Mode

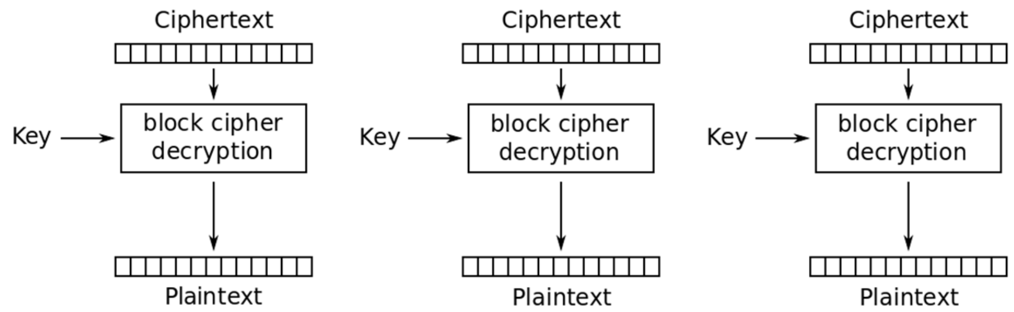
- 블록단위로 순차적 암호 : 모든 블록이 같은 암호화 키 사용 → 두 블록의 값이 같으면, 암호값도 동일
- Notation: C=E(P,K)
- Given plaintext P_0, P_1, …, P_m, …
- Obvious way to use a block cipher is

Encrypt	Decrypt
$C_0 = E(P_0, K)$,	$P_0 = D(C_0, K)$,
$C_1 = E(P_1, K)$,	$P_1 = D(C_1, K)$,
$C_2 = E(P_2, K)$,…	$P_2 = D(C_2, K)$,…

- No error propagation : 한 블록에서 에러가 발생하더라도 다음 블록에 영향을 주지 않음



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

[Wikipedia]
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

ECB Weakness

- 한 개의 블록만 복호되면, 그 외 블록도 복호 (Brute-Force Attack, Dictionary Attack)
- 암호문이 블록의 배수가 되므로, 복호후 평문을 알기 위해서 padding을 해야 함
- Lack of diffusion : Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well. (→Next Page)

ECB Cut and Paste Attack

- Suppose plaintext is

  Alice digs Bob. Trudy digs Tom.
- Assuming 64-bit blocks and 8-bit ASCII:

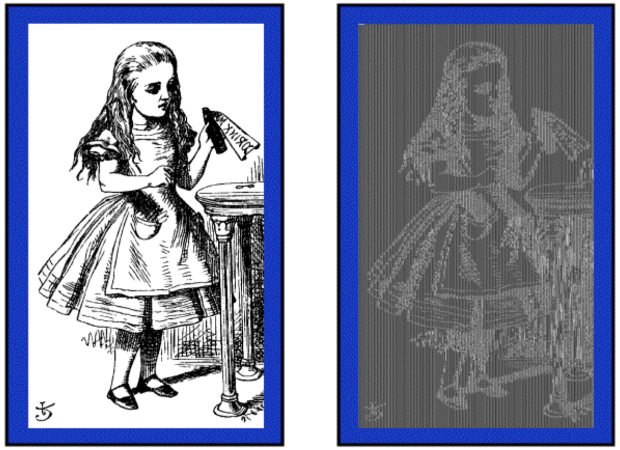
  P_0 = "Alice di", P_1 = "gs Bob. ",

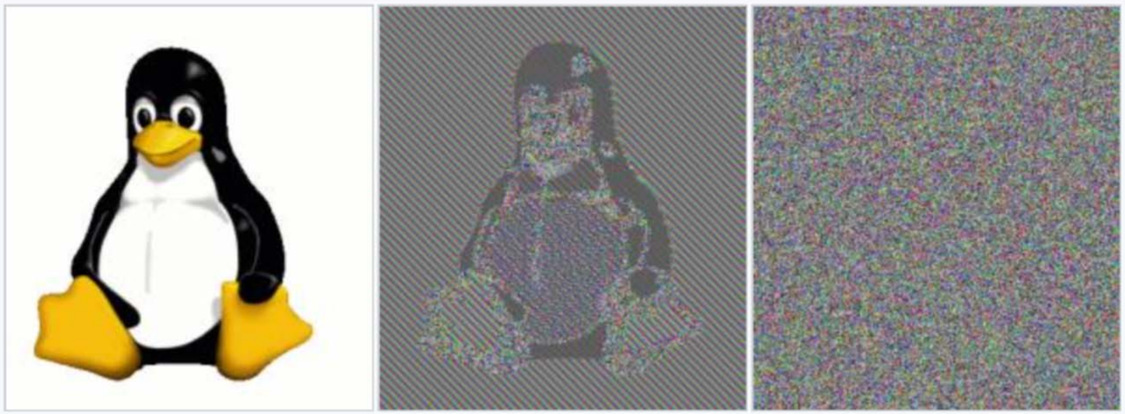
  P_2 = "Trudy di", P_3 = "gs Tom."
- Ciphertext: C_0, C_1, C_2, C_3
- Trudy cuts and pastes 복사-붙여넣기 공격: C_0, C_3, C_2, C_1
- Decrypts as

  Alice digs Tom. Trudy digs Bob.

Alice Hates ECB Mode (Lack of diffusion)

- Alice's uncompressed image, Alice ECB encrypted (TEA; Tiny Encryption Algorithm)





Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

- Why does this happen?
  - Same plaintext block ⇒ same ciphertext!
- Solution??? → Next slide

Cipher Block Chaining Mode

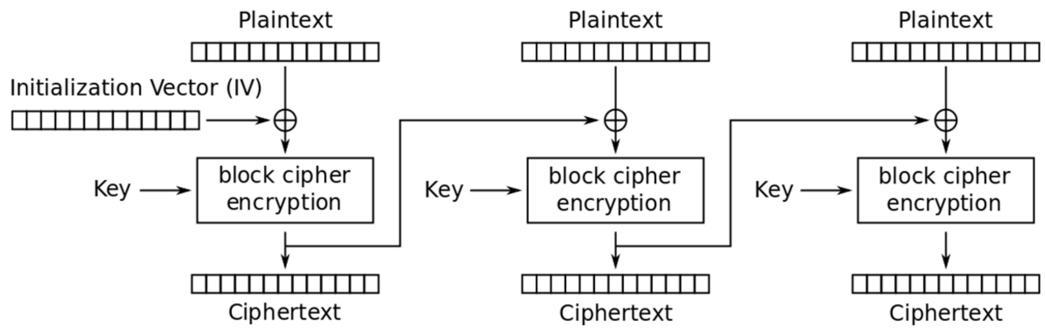
- Blocks are "chained" together
- A random initialization vector, "IV", is required to initialize CBC mode
- IV is random, but need not be secret

Encryption

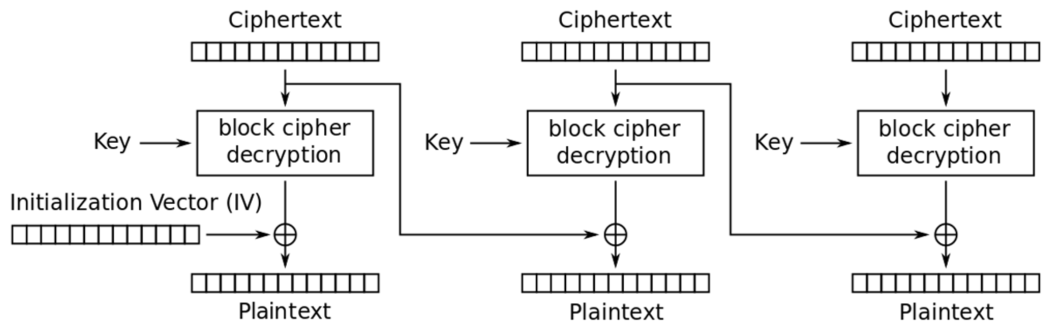
$C_0 = E(IV \oplus P_0, K)$,
$C_1 = E(C_0 \oplus P_1, K)$,
$C_2 = E(C_1 \oplus P_2, K), \cdots$

Decryption

$P_0 = IV \oplus D(C_0, K)$,
$P_1 = C_0 \oplus D(C_1, K)$,
$P_2 = C_1 \oplus D(C_2, K), \cdots$



Cipher Block Chaining (CBC) mode encryption

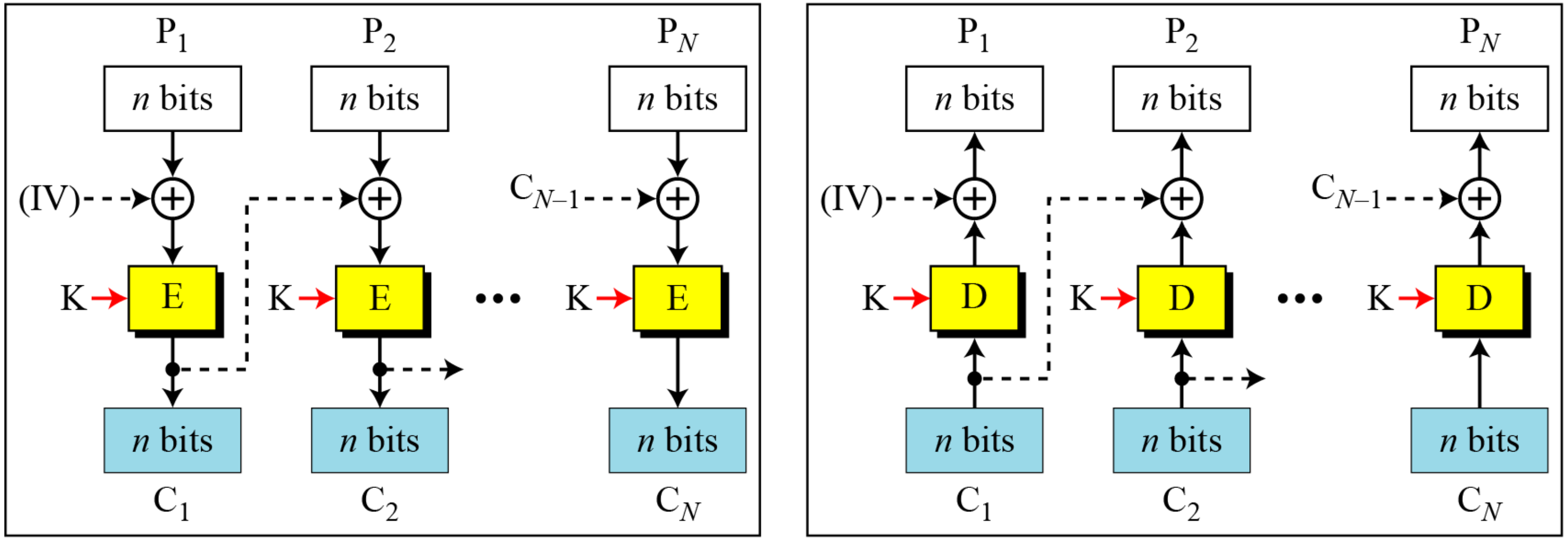


Cipher Block Chaining (CBC) mode decryption

- Main drawbacks
  - ✓ Encryption is sequential (i.e., it cannot be parallelized),
  - ✓ the message must be padded to a multiple of the cipher block size.
  - ✓ Error Propagation : 깨진 암호문의 해당블록과 다음블록의 평문까지 영향을 가짐

Cipher Block Chaining Mode

E: Encryption D : Decryption
P_i: Plaintext block i C_i : Ciphertext block i
K: Secret key IV: Initial vector (C_0)



Encryption Decryption

Encryption:
$C_0 = IV$
$C_i = E_K (P_i \oplus C_{i-1})$

Decryption:
$C_0 = IV$
$P_i = D_K (C_i) \oplus C_{i-1}$

Cipher Block Chaining Mode

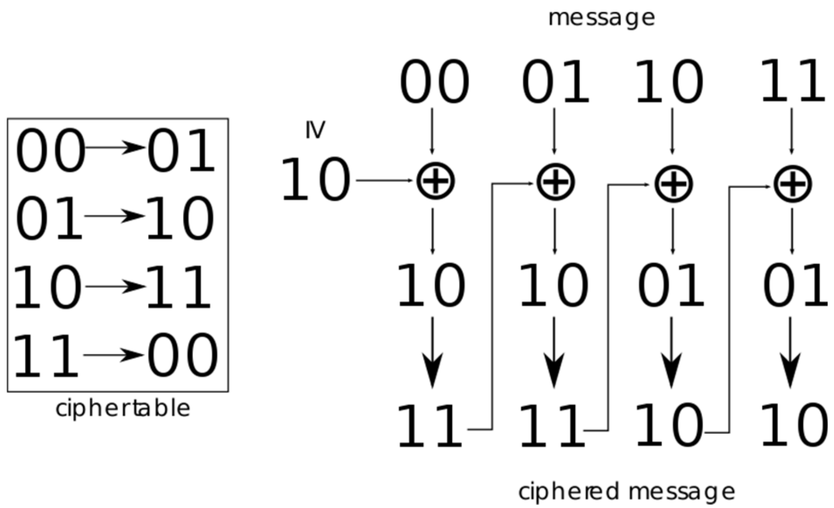
- Identical plaintext blocks yield different ciphertext blocks
- Cut and paste is still possible, but more complex (and will cause garbles)
- If C_1 is garbled to, say, G then

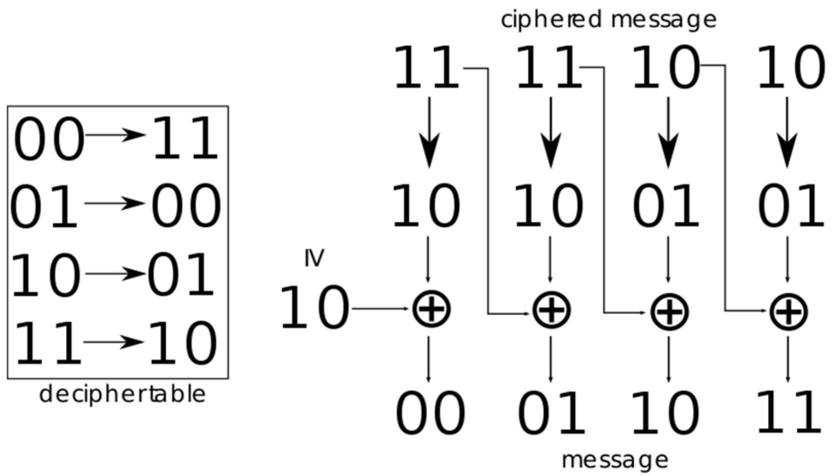
$$P_1 \neq C_0 \oplus D(G, K),\ P_2 \neq G \oplus D(C_2, K)$$

- But

$$P_3 = C_2 \oplus D(C_3, K),\ P_4 = C_3 \oplus D(C_4, K), \cdots$$

- Automatically recovers from errors!





6. Block Cipher Modes – CBC

Algorithm 8.1 *Encryption for ECB mode*

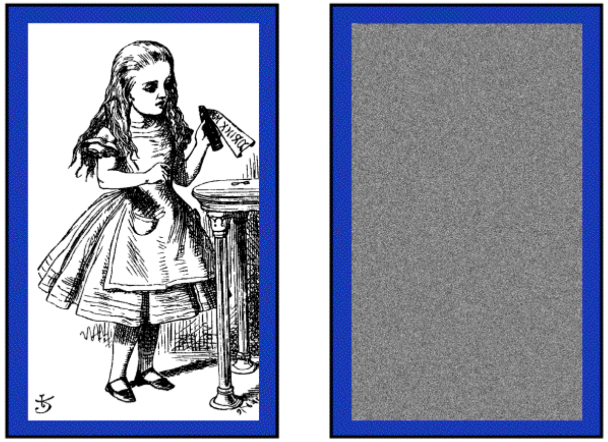
```
ECB_Encryption (K, Plaintext blocks)
{
    for (i = 1 to N)
    {
        C_i ← E_K (P_i)
    }
    return Ciphertext blocks
}
```

Algorithm 8.2 *Encryption algorithm for CBC mode*

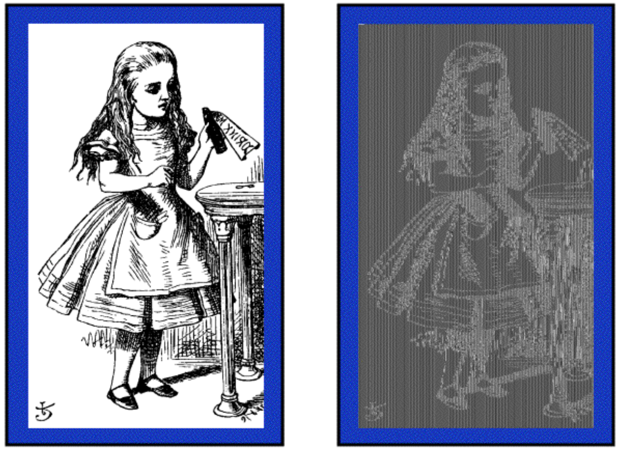
```
CBC_Encryption (IV, K, Plaintext blocks)
{
    C_0 ← IV
    for (i = 1 to N)
    {
        Temp ← P_i ⊕ C_{i−1}
        C_i ← E_K (Temp)
    }
    return Ciphertext blocks
}
```

Alice Likes CBC Mode

- Alice's uncompressed image, Alice CBC encrypted (TEA)



CBC



ECB

- Why does this happen?
  - ✓ Same plaintext yields different ciphertext!

❖ A nonce is an arbitrary number that can be used just once in a cryptographic communication.

Counter (CTR) Mode

- CTR is popular for random access
- Use block cipher like stream cipher

Encryption

$C_0 = P_0 \oplus E(IV, K)$,

$C_1 = P_1 \oplus E(IV+1, K)$,

$C_2 = P_2 \oplus E(IV+2, K)$,⋯

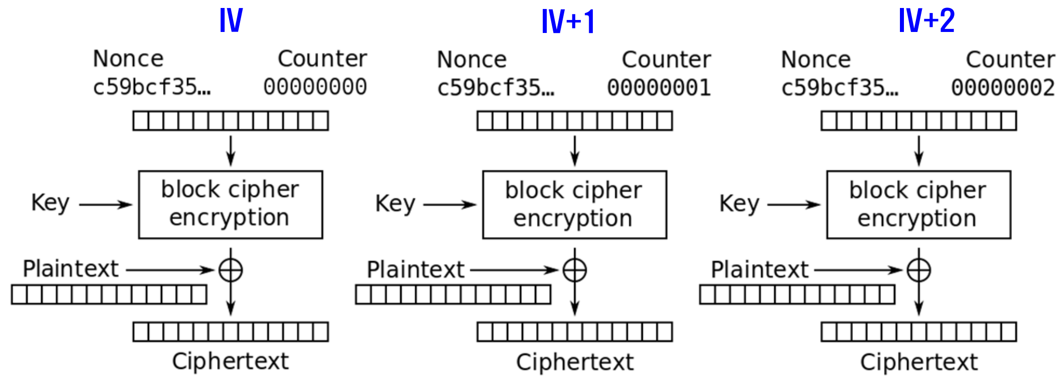
Decryption

$P_0 = C_0 \oplus E(IV, K)$,

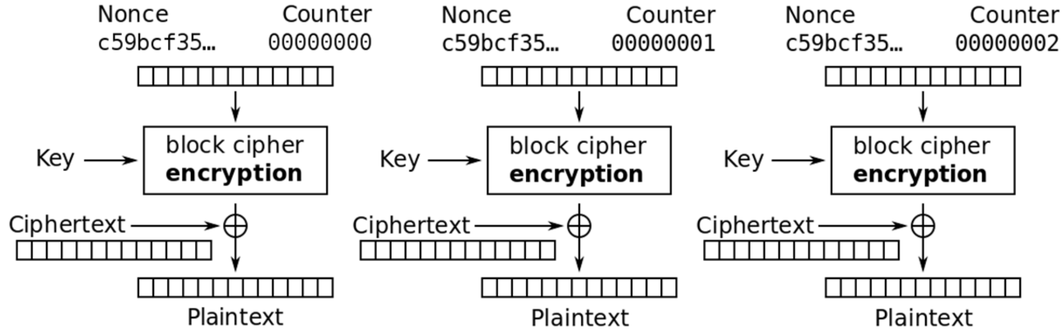
$P_1 = C_1 \oplus E(IV+1, K)$,

$P_2 = C_2 \oplus E(IV+2, K)$,⋯

- 블록 암호화할 때마 1씩 증가하는 Counter를 암호화하여 Key Stream을 생성. 암호화한 Counter 비트열과 평문과 XOR
- Random Access : 블록 순서를 임의로 암/복호 가능 (비표와 블록번호로부터 Counter 구할 수 있음)
- 블록을 임의의 순서로 처리 → 병렬 처리 가능



Counter (CTR) mode encryption

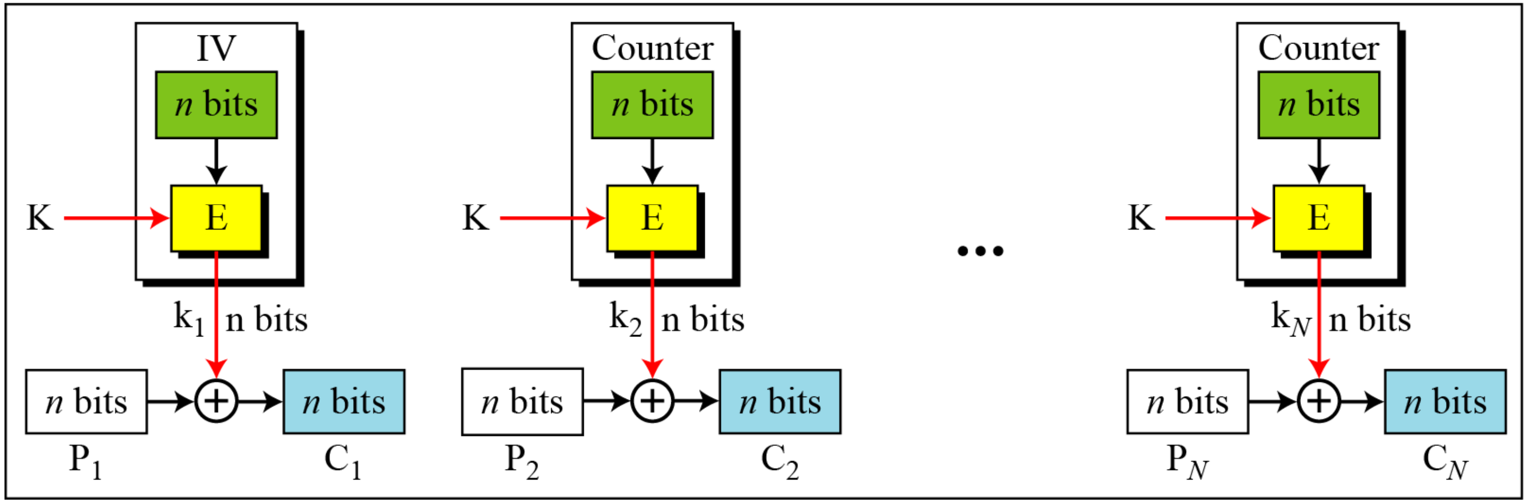


Counter (CTR) mode decryption

Counter (CTR) Mode

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

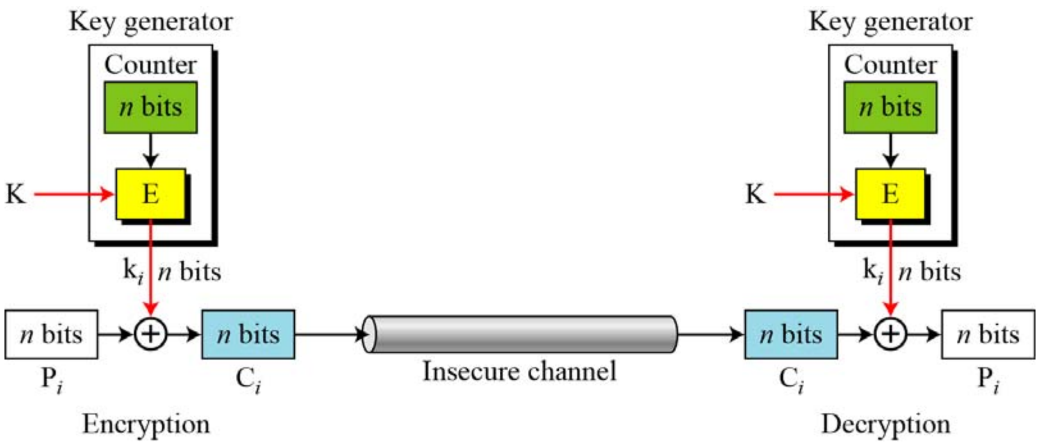


Figure 8.9 스트림 암호로서의 **Counter (CTR)** 모드

Algorithm 8.5 *Encryption algorithm for CTR*

CTR_Encryption (IV, K, Plaintext blocks)
{
  Counter $\leftarrow$ IV
  for (i = 1 to N)
  {
    Counter $\leftarrow$ (Counter + i − 1) mod 2^N
    $k_i \leftarrow E_K$ (Counter)
    $C_i \leftarrow P_i \oplus k_i$
  }
  return Ciphertext blocks
}

Integrity

Data Integrity

- **Integrity (무결성)**
  - ✓ Prevent (or at least detect) unauthorized modification of data
  - ✓ [Wikipedia] **maintenance** of, and the **assurance** of the **accuracy** and **consistency** of **data** over its entire life-cycle, and is a critical aspect to the design, implementation and usage of any system which *stores*, *processes*, or *retrieves* **data**.
- Example: Inter-bank fund transfers
  - ✓ Confidentiality(비밀성) is nice, but integrity is critical
- Encryption provides **confidentiality**
  - ✓ prevents unauthorized disclosure (무단 공개)
- Encryption alone does not assure integrity
  - ✓ recall one-time pad and attack on ECB

MAC

- **Message Authentication Code (MAC)**
  - ✓ Used for **data integrity**
  - ✓ Integrity **not** the same as confidentiality
- MAC is computed as ***CBC residue***
  - ✓ Compute CBC encryption, **but only save the final ciphertext block**

MAC Computation

- MAC computation (assuming N blocks)

$C_0 = E(IV \oplus P_0, K),$

$C_1 = E(C_0 \oplus P_1, K),$

$C_2 = E(C_1 \oplus P_2, K), \cdots,$

$\boxed{C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = \text{MAC}}$

- MAC sent along with plaintext
- Receiver does same computation and verifies that result agrees with MAC
  - ✓ Receiver must also know the key K

Why does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = MAC$
- Alice sends IV, P_0, P_1, P_2, P_3 and MAC to Bob
- Suppose Trudy changes P_1 to X
- Bob computes

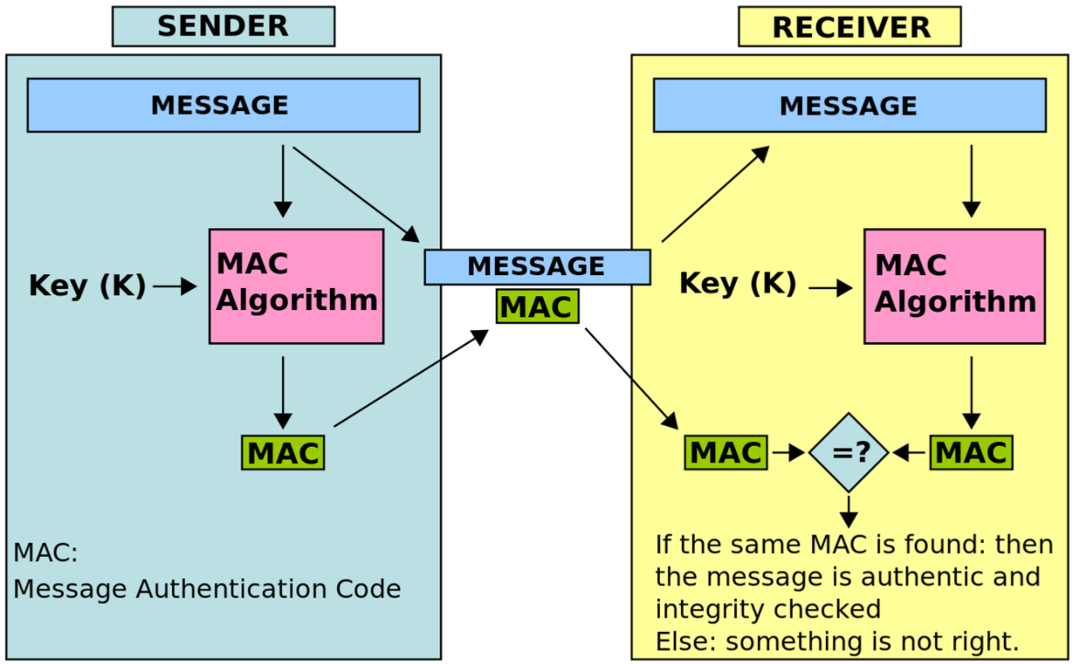
  $C_0 = E(IV \oplus P_0, K)$,

  $C_1 = E(C_0 \oplus X, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$,

  $C_3 = E(C_2 \oplus P_3, K) = MAC \neq MAC$

An example of MAC use



[wikipedia]
https://en.wikipedia.org/wiki/Message_authentication_code

Why does a MAC work?

- Error propagates into MAC (unlike CBC decryption)
  - ✓ Recall CBC decryption
    - If C_1 is garbled to, say, G then
      $P_1 \neq C_0 \oplus D(G, K)$, $P_2 \neq G \oplus D(C_2, K)$
    - But
      $P_3 = C_2 \oplus D(C_3, K)$, $P_4 = C_3 \oplus D(C_4, K), \cdots$
  - ✓ Compare the above to the following
    - $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus X, K)$,
    - $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = MAC' \neq MAC$
- Trudy can't change MAC' to MAC without key K

Confidentiality and Integrity

- Encrypt with one key, compute MAC with another
- Why not use the same key?
  - ✓ Send last encrypted block (MAC) twice?
    - Remember sender have to send Plaintext and MAC for integrity
  - ✓ Can't add any security!

- Using different keys to encrypt and compute **MAC** works, even if keys are related
  - ✓ But still twice as much work as encryption alone
- Confidentiality and integrity with one "encryption" is a research topic

나중에 Message Integrity and Message Authentication를 Chapter로 학습 필요!!!

Uses for Symmetric Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later…)
- Anything you can do with a hash function (upcoming chapter…)