

Symmetric Cryptography

2025.08

자동차융합대학



CONTENTS

01

암호이론과 보안개요

- 암호학 소개
- 고전 암호

02

Data Encryption Standard (DES)

- Confusion(혼돈)과 Diffusion(확산)
- DES 개요
- DES 상세 구조

03

AES (Advanced Encryption Standard)

- AES 소개
- AES 개요
- AES의 내부 구조

04

Mode of Operation

- Padding
- ECB / CBC / CFB / OFB / CTR

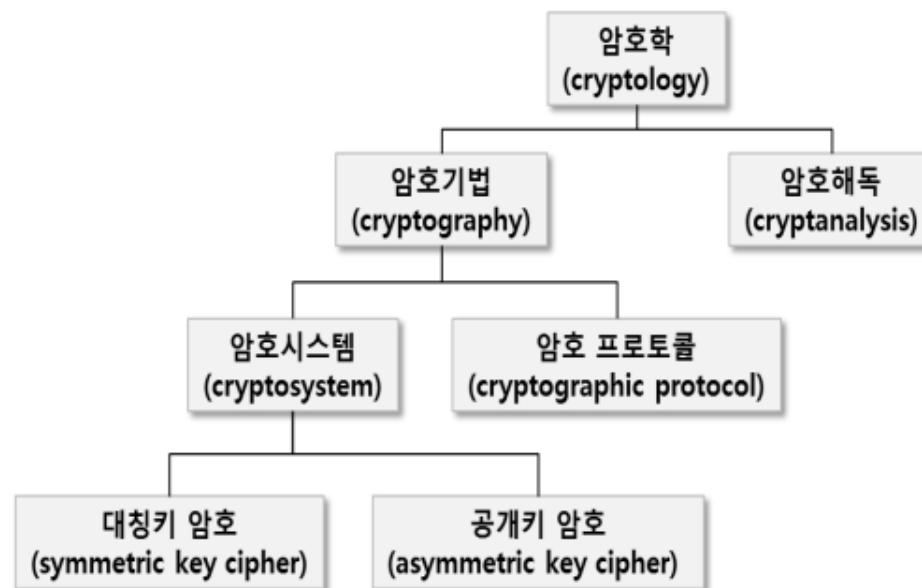
01

암호이론과
보안개요

■ 암호학(Cryptology)

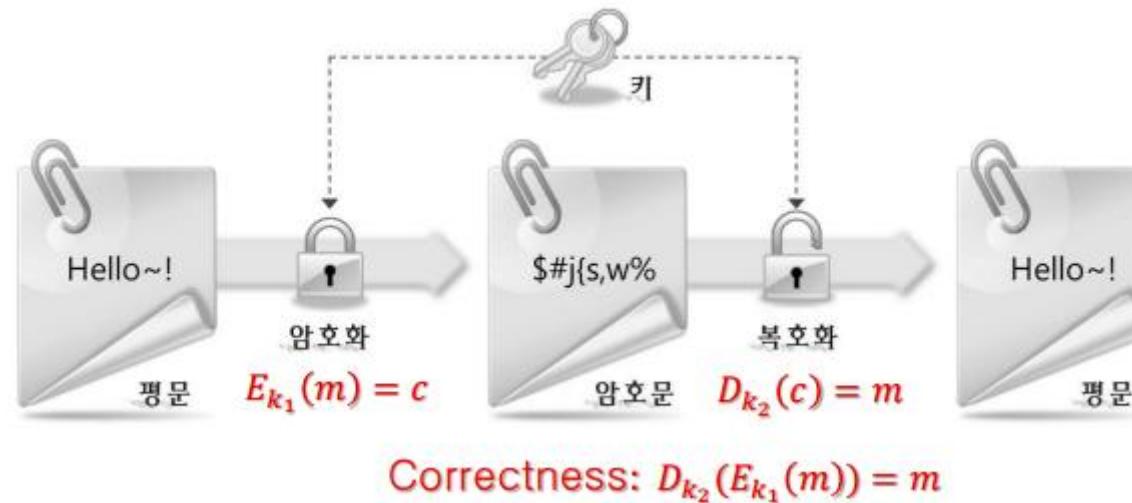
- 암호학은 보안시스템의 가장 중요한 부분이지만 그 자체로는 쓸모가 없다.
 - 웹(Web)의 취약점을 찾는 공격자는 암호를 공격하지 않고도 버퍼 오버플로우(buffer overflow) 등을 이용하여 공격
 - 즉, "A security system is only as strong as its weakest link."

■ 암호학(Cryptology)의 분류



■ 암호기법(Cryptography)

- 그리스어로 “비밀(secret)”을 의미하는 kryptos와 “쓰다(write)”를 의미하는 grapho의 합성어
- 즉, 메시지의 기밀성(confidentiality)을 제공하기 위하여 사용. 현재는 메시지를 공격자로부터 안전하게 보호하기 위하여 메시지를 변화하는 과학이나 기술을 의미
- Key: uniformly distributed random string
- Symmetric if $k_1 = k_2$, Otherwise, asymmetric



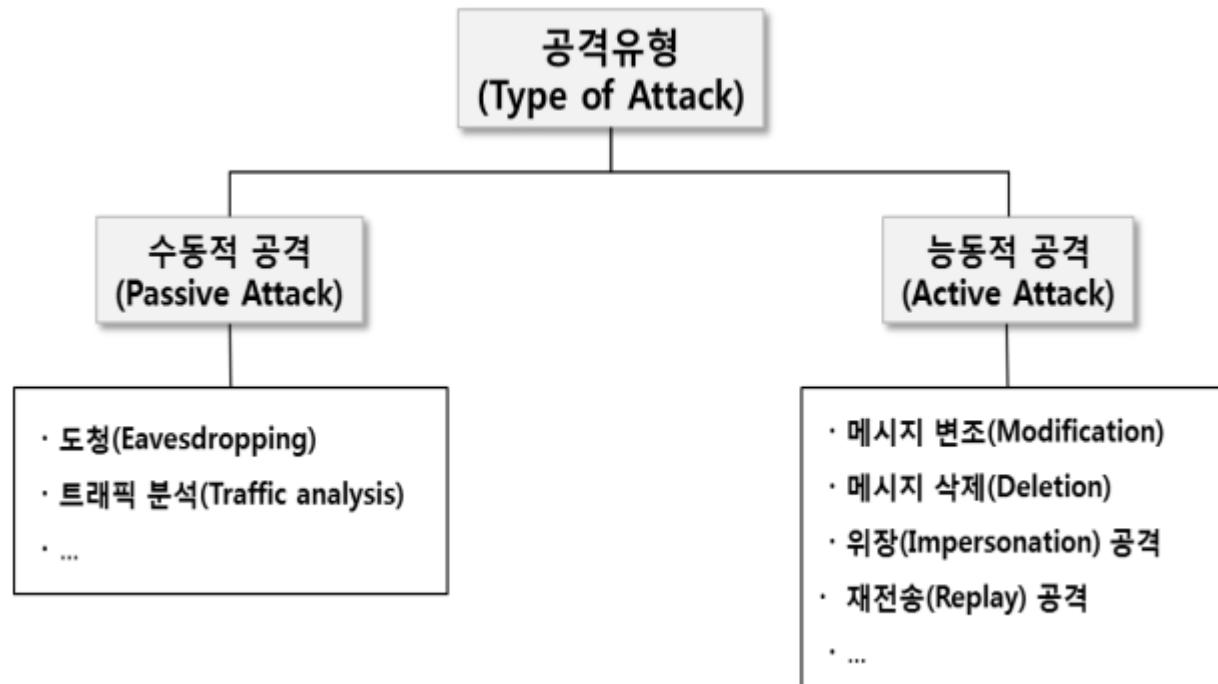
■ Symmetric vs. Public Key algorithms

- Secure communication using symmetric key k .
 - Alice → Bob : $E_k(m) = c$
 - Bob decrypts c by using k
- How can Alice and Bob share k ?
 - Public key algorithms solves this problem!
 - Encrypt k using Bob's public key k_{pub} , i.e., $E_{k_{pub}}(k)$.
 - Bob then decrypts $E_{k_{pub}}(k)$ using his private key k_{priv} to obtain k .

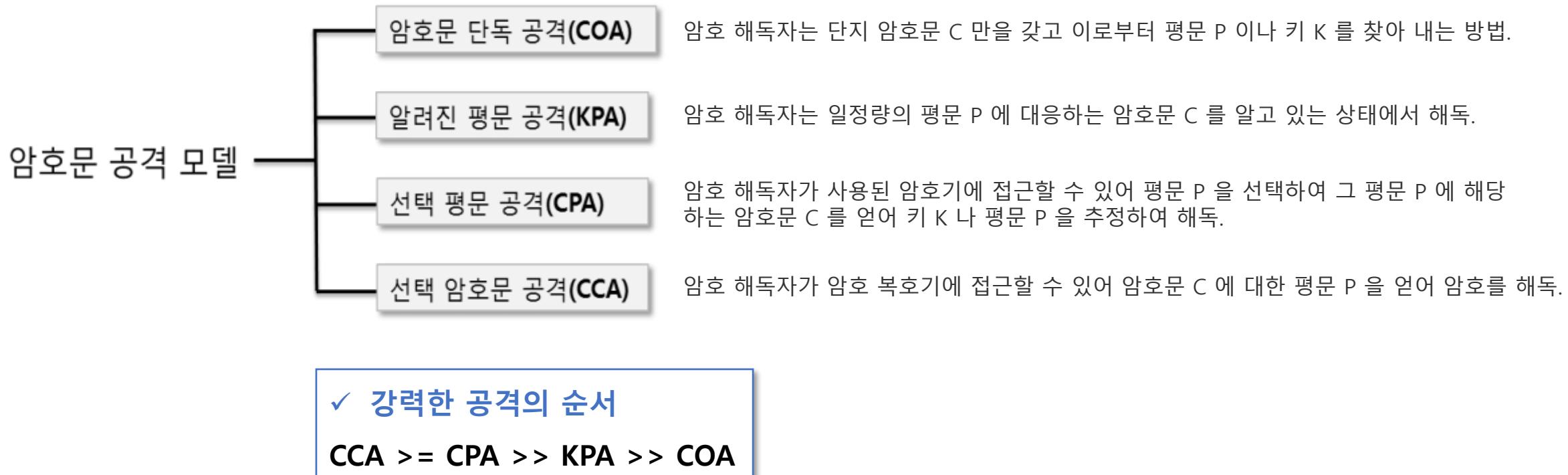
■ Kerckhoffs의 원리

- 암호 알고리즘은 알고리즘의 모든 내용이 공개되어도 키가 노출되지 않으면 안전해야 한다.
- 1. 짧은 길이의 키를 안전하게 보관하는 것은 키 보다 수천배의 사이즈인 암호 알고리즘 전체를 안전하게 보관하는 것 보다 용이. 또한 암호시스템은 역공학 등으로 노출될 수 있지만 키는 보통 난수이어서 역공학에 안전
- 2. 키가 노출되었을 때 키를 변경하는 것이 새로운 암호시스템을 설계하는 것보다 훨씬 용이
- 3. 암호시스템은 보통 다수의 사용자를 위하여 운영되며, 모든 사용자는 동일한 암호 알고리즘을 사용. 이 경우 암호 통신을 하는 당사자들마다 상이한 암호시스템을 사용하는 것 보다는 동일한 암호시스템을 사용하면서 키만 다르게 설정하는 것이 실용적. → 표준화
- 4. 내부자나 역공학에 의하여 암호시스템이 공개되면 새로운 암호 알고리즘을 설계
- 5. 알고리즘이 공개되면, 더 많은 버그를 찾아내어 알고리즘을 개선시킬 수 있다.
 - 비공개 알고리즘을 상대적으로 신뢰하기 어려운 이유.
 - 비밀이 적으면 적을수록 시스템은 안전함.

■ 공격 유형 (Type of Attack)



■ 공격 모델 (Attack Model)



- **COA:** Ciphertext Only Attack
- **KPA:** Known Plaintext Attack
- **CPA:** Chosen Plaintext Attack
- **CCA:** Chosen Ciphertext Attack

암호시스템의 안전성 증명을 위해 다음과 같은 4가지 공격모델이 존재한다. 어떤 모델에서 안전성을 증명하는 것이 가장 안전한 암호시스템임을 보이는 것인지 선택하고, 그 이유를 자유롭게 서술하시오.

암호문 단독 공격(COA), 알려진 평문 공격(KPA),
선택 평문 공격(CPA), 선택 암호문 공격(CCA)

정답 : 공격자에게 제공되는 정보 또는 능력이 많아질수록 공격자가 암호시스템을 공격할 수 있는 가능성이 높아진다. 다시 말하면, 많은 능력을 지닌 공격자에게 안전하도록 암호시스템을 설계한다면 능력이 적은 공격자에게는 암호시스템이 안전하다는 것을 보장할 수 있다. 따라서 공격자에게 가장 많은 능력이 주어진 **선택 암호문 공격모델(CCA)**에서 증명하는 것이 가장 좋다고 할 수 있다. (단, 환경에 따라 선택 평문 공격과 선택 암호문 공격에서의 공격자의 능력이 비슷한 경우도 있다.)

■ 암호의 안전성 개념

- Secure Encryption?
- Given C,
 - No adversary can find k?
 - No adversary can find P?
 - No adversary can find any character?
 - No adversary can find any meaningful information?
 - No adversary can compute any function of P from C.

■ 2가지 원칙

- 치환(Substitution)과 전치(Transposition)

■ 암호 단위

- 고전 암호: 문자
- 현대 암호: 비트

■ 공격 유형

- Brute Force Attack: 전수 키 탐색 공격 (Exhaustive Key Search Attack) → 현대 암호는 키의 길이가 길기 때문에 전사적 공격은 사실상 불가능
- Frequency Analysis: 평문의 통계학적 특성이 암호문에 나타나는 성질을 이용하여 공격하는 방법
- Cryptanalytic Attack
 - The nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs

■ 치환 암호

- 단일 문자 치환 암호(Monoalphabetic Substitution Cipher)
 - 평문의 한 문자와 암호문의 한 문자는 언제나 일대일 관계
- 다중 문자 치환 암호(Polyalphabetic Substitution Cipher)
- Plaintext and Ciphertext in Z_{26}

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Plaintext → | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| Ciphertext → | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Value → | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

■ 단일 문자 치환 암호: 덧셈 암호(Additive Cipher)

- 시저 암호(Caesar Cipher)
 - 평문의 한 문자가 오른쪽 세 자리 뒤에 위치한 문자로 치환
- 암호화: $c \equiv m + 3 \pmod{26}, m \in Z_{26}$
- 복호화: $m \equiv c - 3 \pmod{26}, c \in Z_{26}$

| | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|
| m | A | B | C | D | E | F | G | H | I | J | K | L | M |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| $c \equiv m + 3 \pmod{26}$ | d | e | f | g | h | i | j | k | l | m | n | o | p |
| | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| m | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| $c \equiv m + 3 \pmod{26}$ | q | r | s | t | u | v | w | x | y | z | a | b | c |
| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 0 | 1 | 2 |

■ 덧셈 암호(Additive Cipher)

- 암호화: $c \equiv m + k \pmod{26}, m \in Z_{26}$
- 복호화: $m \equiv c - k \pmod{26}, c \in Z_{26}$

■ 치환 암호의 경우

- 가능한 키의 개수는 총 $26 \times 25 \times \dots \times 1 = 26!$
 - 전사적 공격을 이용하여 공격자가 키를 찾는 것은 불가능

■ 빈도수 공격

- 통계적 특성 이용
- 988,968개의 영어 단어 중, "E"가 사용되는 횟수는 12.7%

| Letter | Probability | Letter | Probability | Letter | Probability |
|--------|-------------|--------|-------------|--------|-------------|
| A | 0.082 | B | 0.015 | C | 0.028 |
| D | 0.043 | E | 0.127 | F | 0.022 |
| G | 0.020 | H | 0.061 | I | 0.070 |
| J | 0.002 | K | 0.008 | L | 0.040 |
| M | 0.024 | N | 0.067 | O | 0.075 |
| P | 0.019 | Q | 0.001 | R | 0.060 |
| S | 0.063 | T | 0.091 | U | 0.028 |
| V | 0.010 | W | 0.023 | X | 0.001 |
| Y | 0.020 | Z | 0.001 | | |

■ 다중 문자 치환 암호

- 비제네르 암호 (Vigenere Cipher)
- 길이가 d인 키워드를 암호화 키로 사용
- $K = k_1k_2k_3 \dots k_d, f_i(m) = (m + k_i) \bmod n$

▶ 예 : K = "CIPHER" (d = 6)

▶ "THISISASECRETMESSAGE"

| 평문 | T | H | I | S | I | S | A | S | E | C | R | E | T | M | E | S | S | A | G | E |
|-----|----|----|----|----|----|----|---|----|----|---|----|----|----|----|----|----|----|----|---|----|
| | 19 | 7 | 8 | 18 | 8 | 18 | 0 | 18 | 4 | 2 | 17 | 4 | 19 | 12 | 4 | 18 | 18 | 0 | 6 | 4 |
| 키 | C | I | P | H | E | R | C | I | P | H | E | R | C | I | P | H | E | R | C | I |
| | 2 | 8 | 15 | 7 | 4 | 17 | 2 | 8 | 15 | 7 | 4 | 17 | 2 | 8 | 15 | 7 | 4 | 17 | 2 | 8 |
| 암호문 | V | P | X | Z | M | J | C | A | T | J | V | V | V | U | T | Z | W | R | I | M |
| | 21 | 15 | 23 | 25 | 12 | 9 | 2 | 0 | 19 | 9 | 21 | 21 | 21 | 20 | 19 | 25 | 22 | 17 | 8 | 12 |

주기 d = 6

[문제] 비제네르 암호

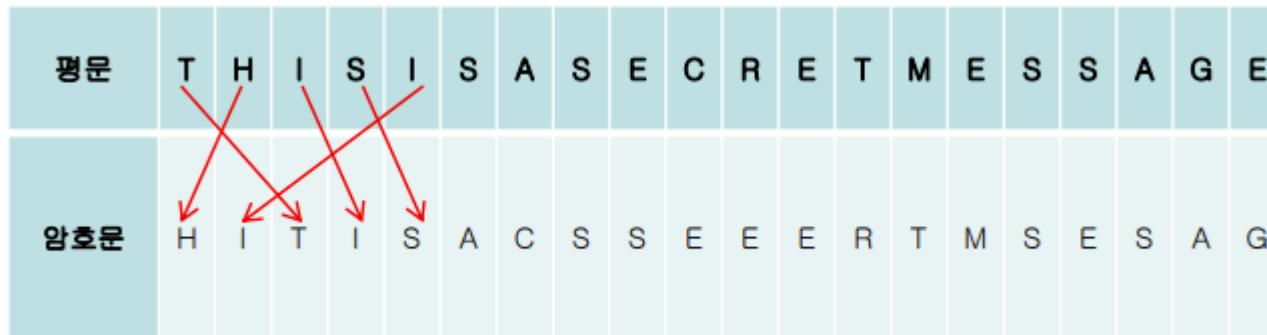
■ 비제네르 암호를 이용하여, 다음과 같은 암호문을 복호화 하시오.

- Ciphertext: BUJCIKTAMCEQXJYFZRR
- Key: THEORY

| 암호문 | B | U | J | C | I | K | T | A | M | C | E | Q | X | J | Y | F | Z | R | R |
|-----|----|----|---|----|----|----|----|----|----|----|----|----|----|---|----|----|----|----|----|
| 키 | 1 | 20 | 9 | 2 | 8 | 10 | 19 | 0 | 12 | 2 | 4 | 16 | 23 | 9 | 24 | 5 | 25 | 17 | 17 |
| 평문 | T | H | E | O | R | Y | T | H | E | O | R | Y | T | H | E | O | R | Y | T |
| | 19 | 7 | 4 | 14 | 17 | 24 | 19 | 7 | 4 | 14 | 17 | 24 | 19 | 7 | 4 | 14 | 17 | 24 | 19 |
| | 8 | 13 | 5 | 14 | 17 | 12 | 0 | 19 | 8 | 14 | 13 | 18 | 4 | 2 | 20 | 17 | 8 | 19 | 24 |
| | I | N | F | O | R | M | A | T | I | O | N | S | E | C | U | R | I | T | Y |

■ 평문 메시지의 문자들을 재배열

- 전치 암호화 함수를 π 라 하고 문자열의 길이를 l 이라 하면,
 - $\pi = (\pi(1), \dots, \pi(l))$
- $\pi(i)$: 평문에서 i 번째 위치에 있는 문자의 암호문에서의 위치
- 예제: $\pi = (\pi(1), \dots, \pi(5)) = (3, 1, 4, 5, 2)$



02

DES
(Data Encryption Standard)

■ 블록 암호 설계를 위하여 적용되는 기본적인 원리

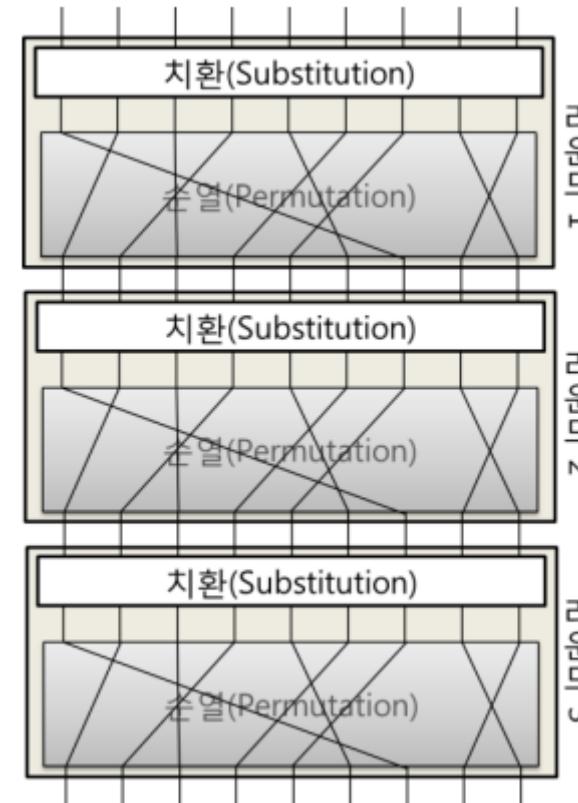
- 정보이론 학자인 Shannon 제안
- 혼돈(Confusion)
 - 키와 암호문과의 관계를 감추는 성질
 - 현대 블록암호는 혼돈을 위해 치환 (Substitution)
- 확산(Diffusion)
 - 평문과 암호문과의 관계를 감추는 성질
 - 평문 한 비트의 변화가 암호문의 모든 비트에 확산
 - 주로 평문과 암호문의 통계적 성질을 감추기 위해 사용
 - DES에서는 여러 번의 순열(Permutation)을 사용하여 확산 성질을 만족하고, AES에서는 좀 더 발전된 형태인 MixColumn을 사용



Shannon

■ 곱 암호 (Product Cipher)

- 샤논은 혼돈과 확산을 함께 사용하면 안전한 암호를 설계할 수 있다고 제안
 - 라운드(Round)라 불리는 부분이 여러 번 반복되는 구조
 - 한 라운드에는 치환과 순열이 사용됨

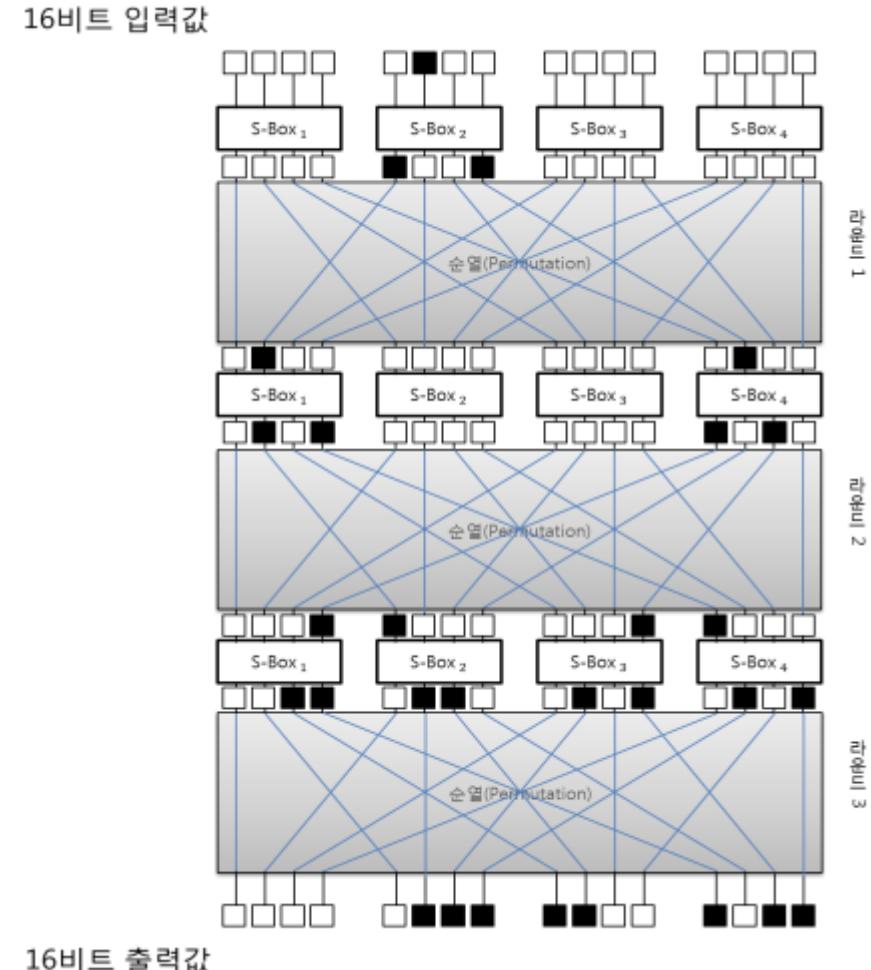


혼동(Confusion)과 확산(Diffusion)

■ DES: 곱 암호의 형태

- Design principle: S-Box
 - 조건 1: 한비트가 상이한 두 입력 값이 S-Box에 입력되었을 때 각 입력 값에 대응되는 두 출력 값은 2비트 이상이 상이해야 한다.
 - 조건 2: 하나의 S-Box에서 출력된 비트들은 다음 라운드에서 모든 S-Box들의 입력 값으로 확산되어 들어 갈 수 있도록 순열을 설계하여야 한다.

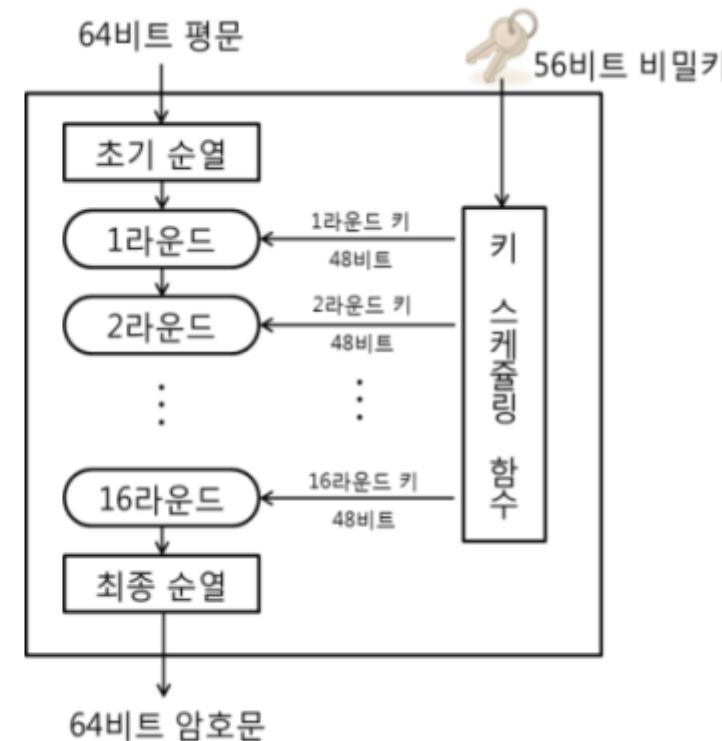
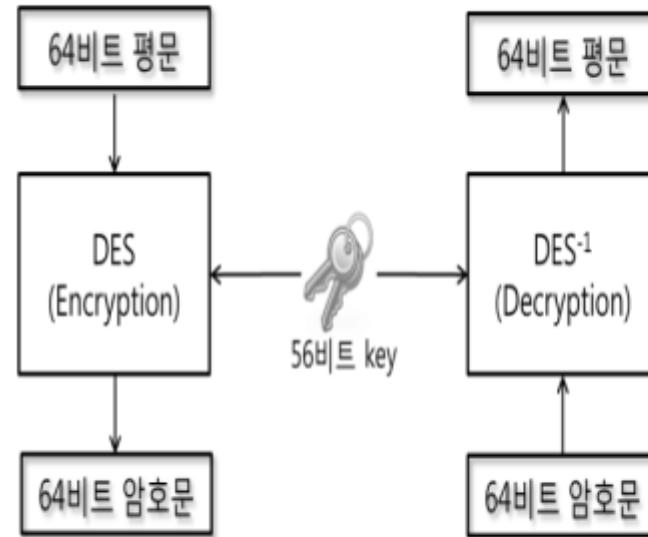
Note: 블록 사이즈가 2^n 인 곱 암호가 충분한 혼돈과 확산 성질을 만족하기 위해서는 $(n - 1)$ 라운드 이상으로 구성



■ DES (Data Encryption Standard)

- 1973년 미국의 연방 표준국(National Bureau of Standards) DES 공모
- IBM은 자사의 루시퍼(Lucifer)를 제출
- 미 연방 표준국은 1977년 루시퍼를 수정하여 DES로 선정
 - FIPS PUB 46
 - 국가안보국(National Security Agency, NSA)는 루시퍼에서 사용된 64비트 키를 56비트로, S-Box의 형태도 변형
- Most widely used block cipher in the world
- 64-bit 데이터 블록, 56-bit 키
 - 최대 2^{56} 번의 계산 필요 → 현재 2^{113} 정도의 계산이 안전
 - 따라서, 3-DES(Triple DES)와 AES(Advanced Encryption Standard) 사용

■ 64비트의 평문을 56비트의 키로 암호화하여 65비트의 암호문을 생성



■ 곱 암호의 두 가지 형태

- Feistel 암호
 - 가역(Invertible) 요소와 비가역(Non-Invertible) 요소 모두를 사용
 - 암호화와 복호화 과정이 동일
- Non-Feistel 암호
 - 가역 요소만 사용

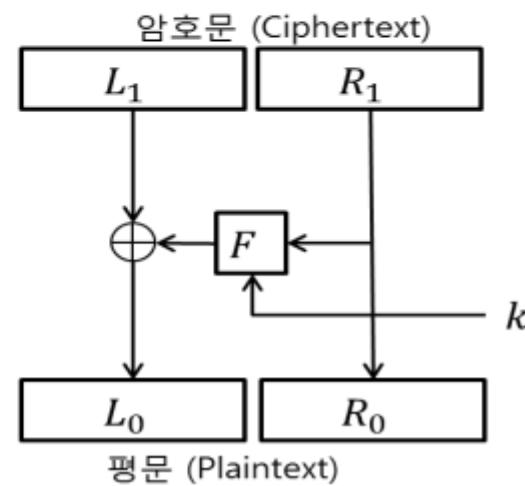
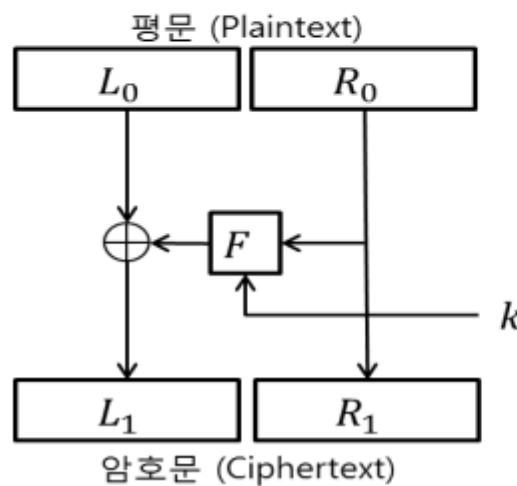
Note : 전단사 함수 $f:X \rightarrow Y$ 에 대하여 Y 에서 X 로의 역관계가 존재하면 이를 역함수(Inverse Function)라고 하며 $f^{-1}:Y \rightarrow X$ 로 나타낸다.
가역함수는 바로 역함수가 존재하는 전단사함수를 의미

■ 1라운드 Feistel 구조

암호화: $L_1 = L_0 \oplus F(k, R_0); R_1 = R_0$

복호화:

- * $L_1 \oplus F(k, R_1) = L_1 \oplus F(k, R_0) = L_0 \oplus F(k, R_0) \oplus F(k, R_0) = L_0$
- * $R_0 = R_1$



■ 2라운드 Feistel 구조

$$\text{암호화: } L_1 = R_0$$

$$R_1 = L_0 \oplus F(k_1, R_0)$$

$$L_2 = L_1 \oplus F(k_2, R_1)$$

$$R_2 = R_1$$

$$\text{복호화: } L'_0 = L_2, \quad R'_0 = R_2$$

$$\times \quad L'_1 = R'_0 = R_2$$

$$\times \quad R'_1 = L'_0 \oplus F(k_2, R'_0)$$

$$= L_1 \oplus F(k_2, R_1) \oplus F(k_2, R_2)$$

$$= L_1 \oplus F(k_2, R_2) \oplus F(k_2, R_2)$$

$$= L_1$$

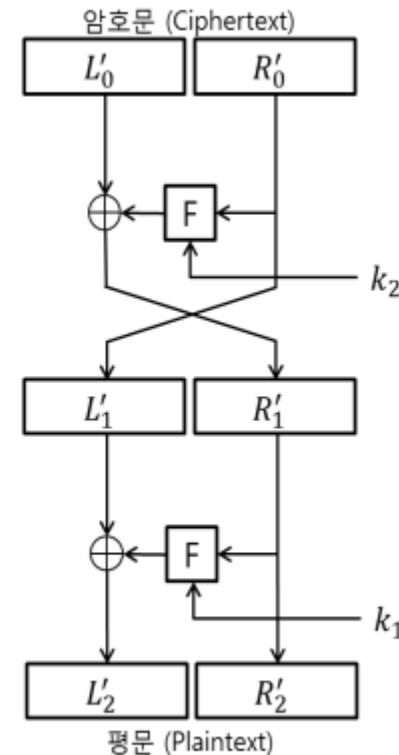
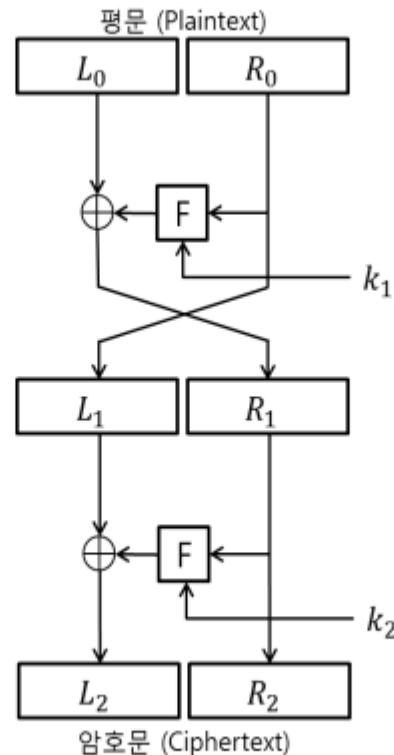
$$\times \quad R'_2 = R'_1 = L_1 = R_0$$

$$\times \quad L'_2 = L'_1 \oplus F(k_1, R'_1)$$

$$= R_2 \oplus F(k_1, L_1)$$

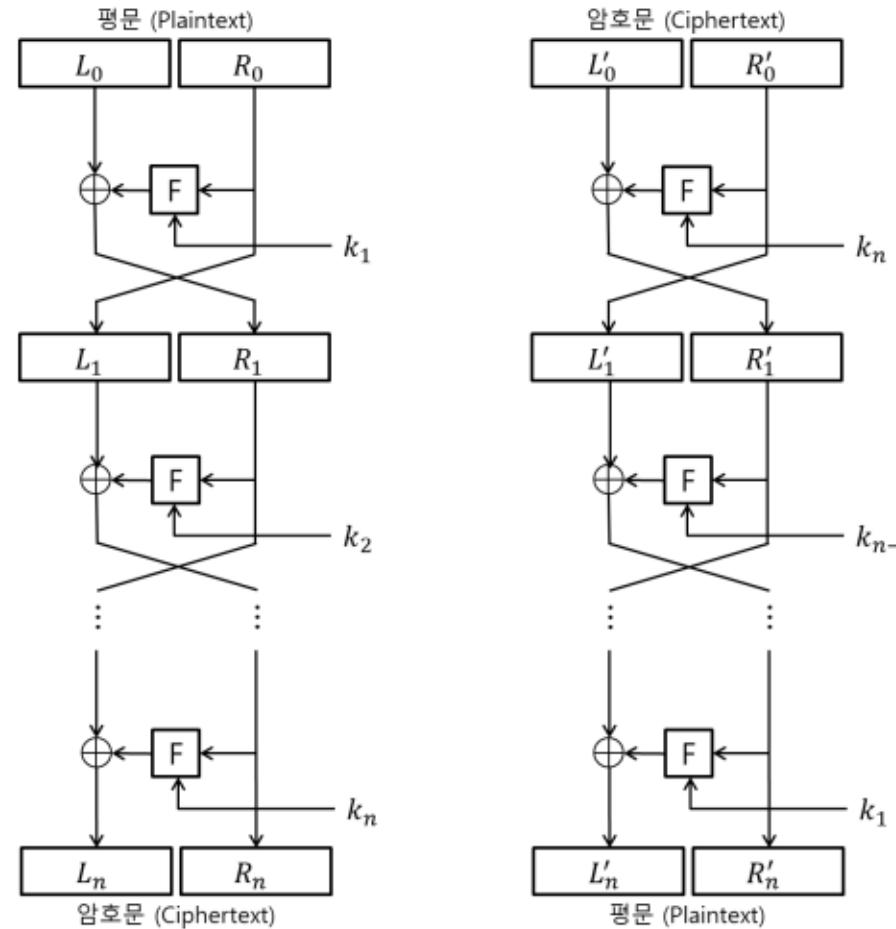
$$= L_0 \oplus F(k_1, R_0) \oplus F(k_1, R_0)$$

$$= L_0$$

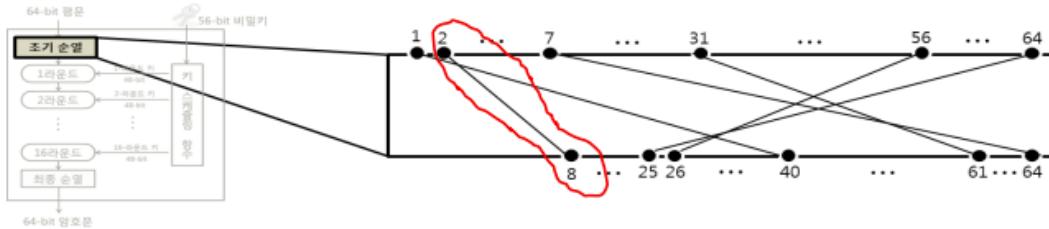


■ 다중라운드 Feistel 구조

- Feistel 구조 2라운드는 SPN(Substitution Permutation Network)구조 1라운드와 같은 안전성을 갖기 때문에, 많은 라운드 수가 필요하다.

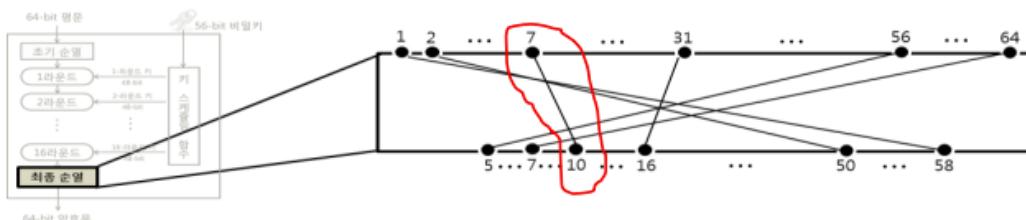


■ 초기 순열 IP / 최종 순열 FP ($= IP^{-1}$)



초기 순열 (Initial Permutation)

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

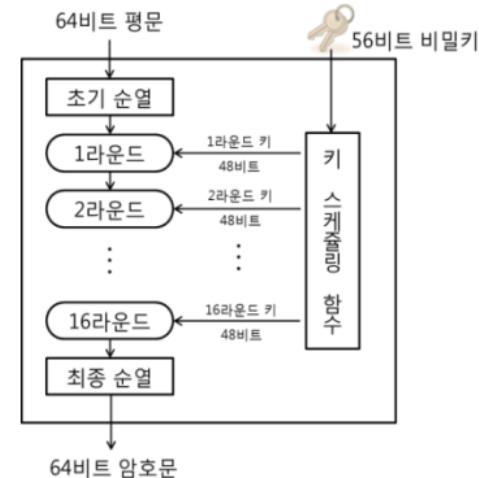


최종 순열 (Final permutation)

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



```
def permutation(block, table):
    outlen = len(table)
    permuted = bytearray(outlen)
    for n in range(outlen):
        m = table[n] - 1
        permuted[n] = block[m]
    return permuted
```

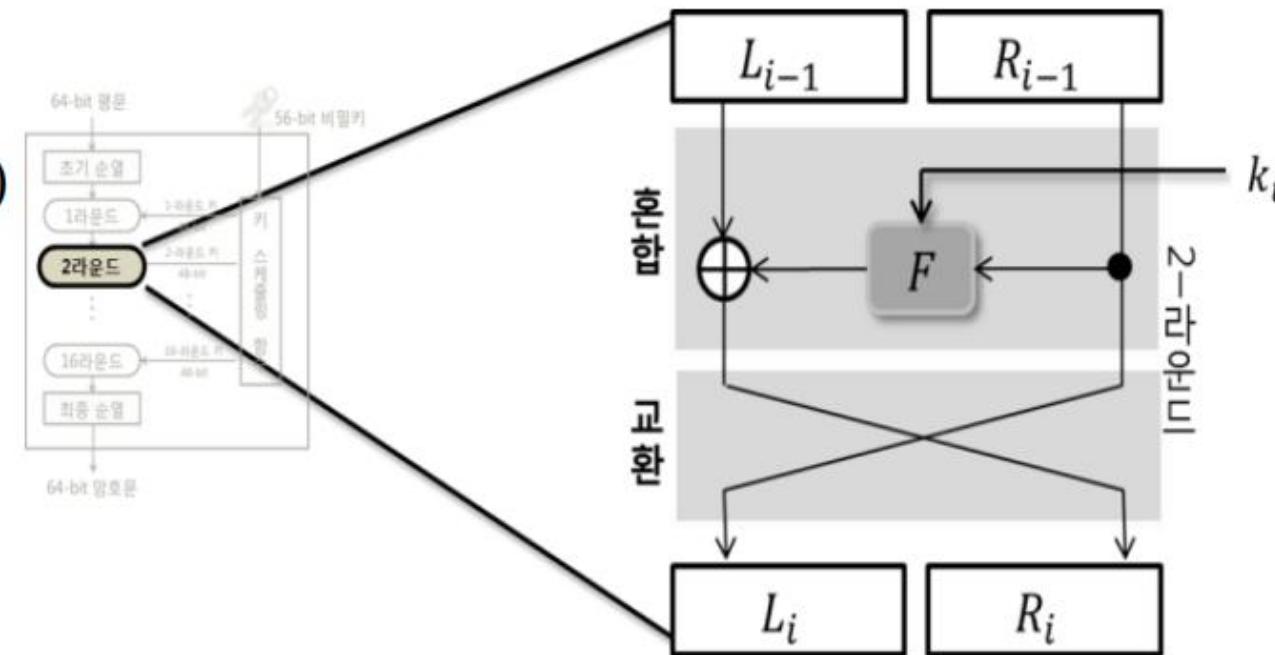


■ 라운드 함수(F 함수)

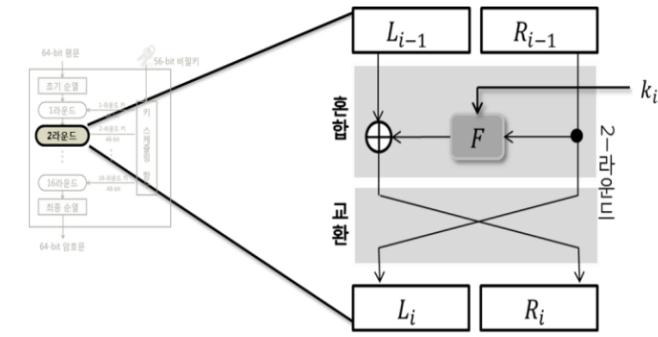
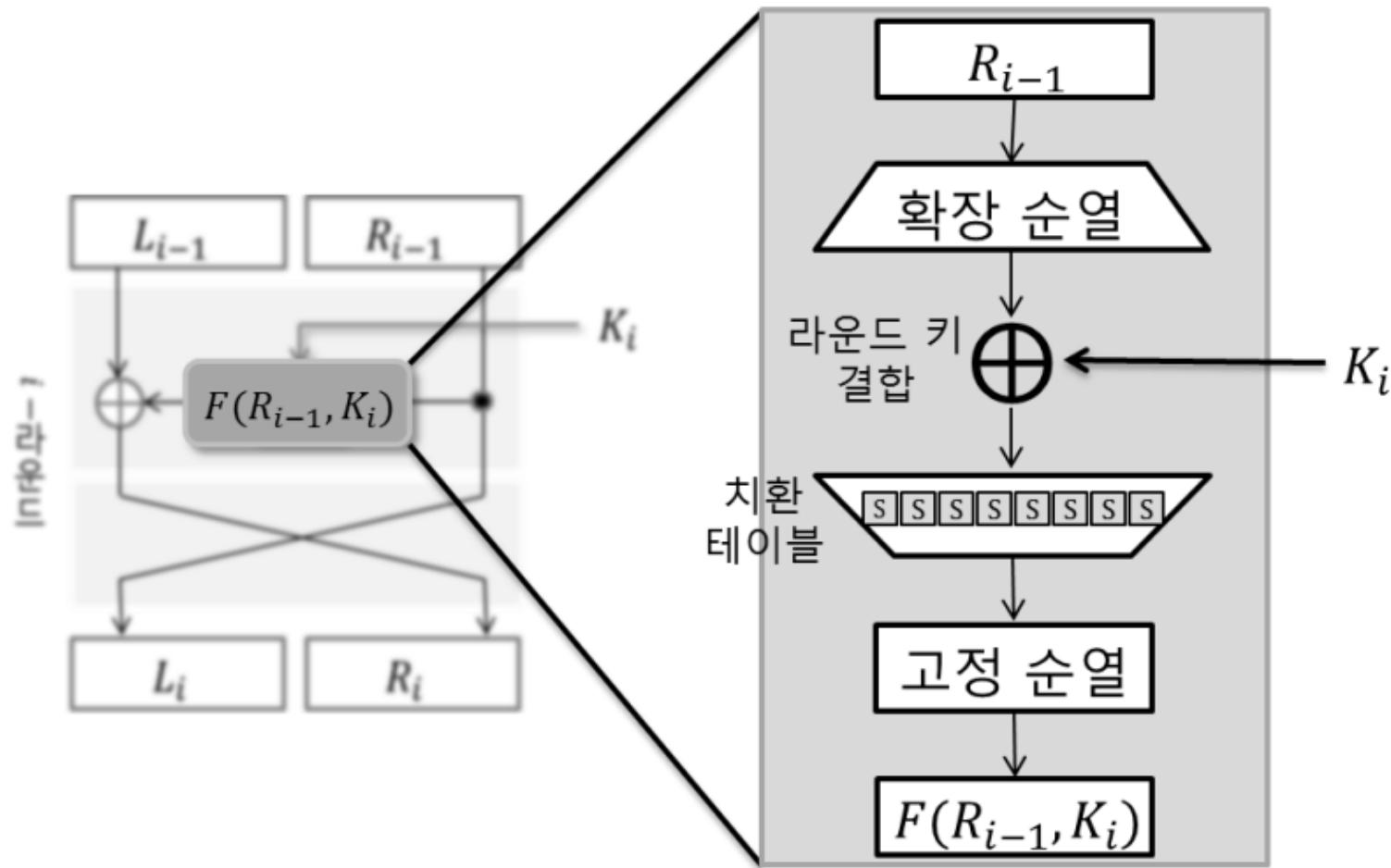
- 혼합(Mixer)
 - 오른쪽 32비트 부분과 그 라운드에 해당하는 키를 이용해 F함수 값을 계산한 후, 왼쪽 32비트 부분과 그 값을 XOR 연산
- 교환(Swapper)
 - XOR 연산을 통해 계산된 오른쪽으로 처음 입력된 값의 오른쪽 32비트를 왼쪽으로 위치를 바꿔준다.

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$$

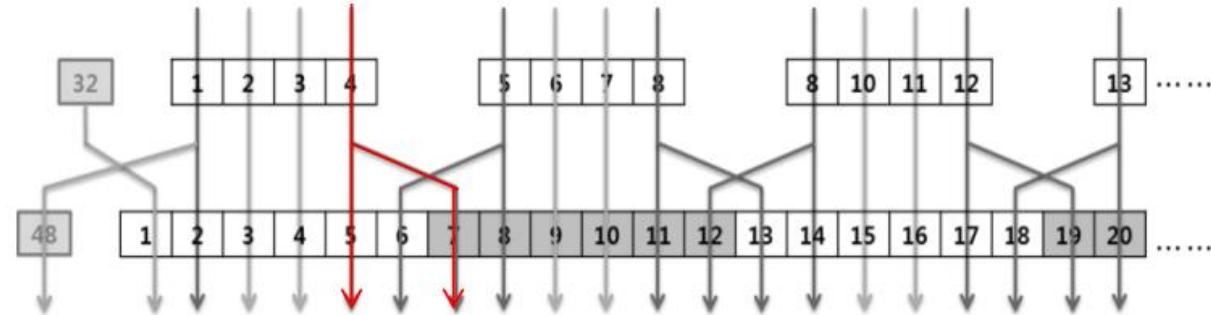


■ 라운드 함수(F 함수)



DES 상세구조 (cont'd)

■ 확장 순열

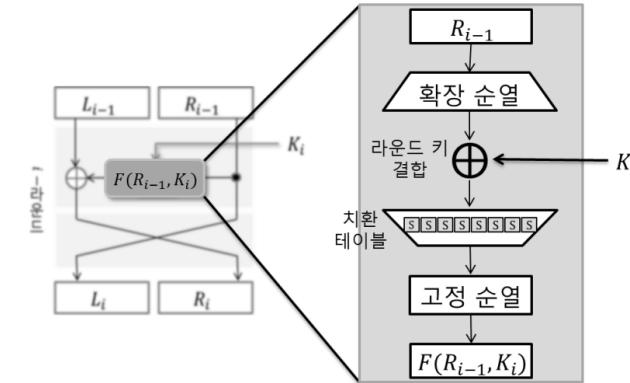


32비트 입력값

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

48비트 출력값

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

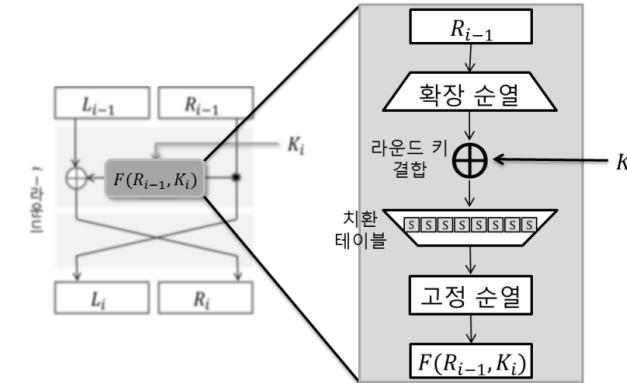
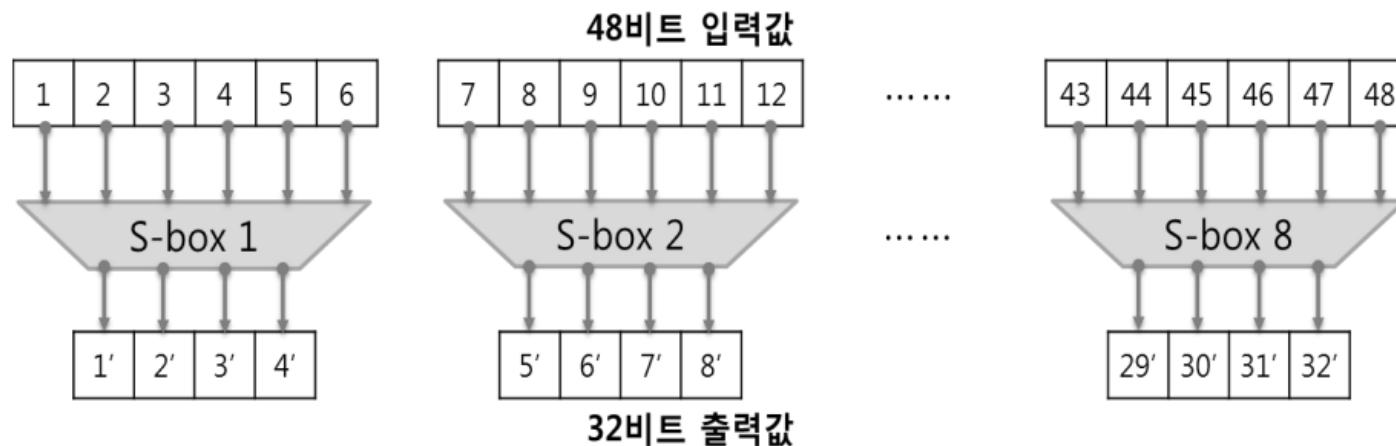


■ 라운드 키 결합 (XOR)

- 확장 된 48비트 XOR 라운드 키

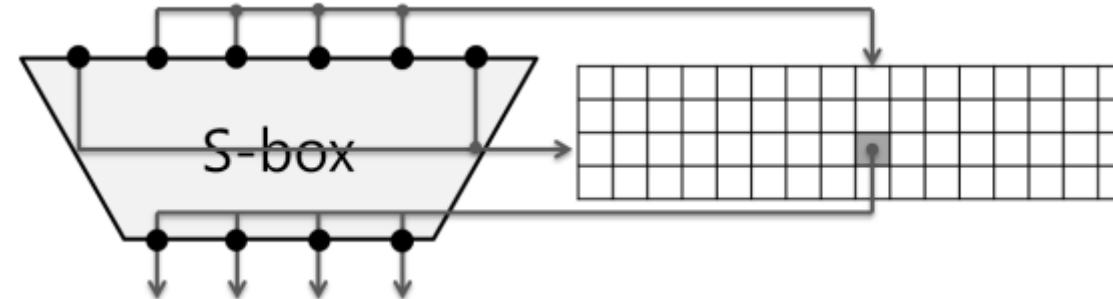
■ 변환 테이블 (S-Box)

- 48비트의 입력 값을 6비트씩 8개의 부분으로 나눈다.
- 각 6비트를 순서대로 8개의 X-Box에 입력한다.
- 각 S-Box는 6비트를 입력 받아 4비트로 줄여 출력한다.



■ 변환 테이블 (S-Box)

- 규칙 1
 - 입력된 6비트 중 1번째와 6번째 비트를 붙여서 십진수로 나타내면 0부터 3까지의 수 중 하나가 되는데, 이 값으로 행을 결정한다.
- 규칙 2
 - 나머지 2번째부터 5번째까지 4비트를 붙여서 십진수로 나타내면 0부터 15까지의 수 중 하나가 되는데, 이 값으로 열을 결정한다.



■ 변환 테이블 (S-Box)

100111

- ▶ Row = 11 = 3
- ▶ Column = 0011 = 3
- ▶ → 02

비선형성 :

$S(A) \oplus S(B) \neq S(A \oplus B)$.



```
def substitution(block, table):
    row = (block[0] << 1) + block[5]
    column = (block[1] << 3) + (block[2] << 2) + (block[3] << 1) + block[4]
    val = table[row][column]
    binary = bin(val)[2:4].zfill(4)
    return bytearray([int(b) for b in binary])
```

S-box 1

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 14 | 04 | 13 | 01 | 02 | 15 | 11 | 08 | 03 | 10 | 06 | 12 | 05 | 09 | 00 | 07 |
| 1 | 00 | 15 | 07 | 04 | 14 | 02 | 13 | 10 | 03 | 06 | 12 | 11 | 09 | 05 | 03 | 08 |
| 2 | 04 | 01 | 14 | 08 | 13 | 06 | 02 | 11 | 15 | 12 | 09 | 07 | 03 | 10 | 05 | 00 |
| 3 | 15 | 12 | 08 | 02 | 04 | 09 | 01 | 07 | 05 | 11 | 03 | 14 | 10 | 00 | 06 | 13 |

S-box 2

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 15 | 01 | 08 | 14 | 06 | 11 | 03 | 04 | 09 | 07 | 02 | 13 | 12 | 00 | 05 | 10 |
| 1 | 03 | 13 | 04 | 07 | 15 | 02 | 08 | 14 | 12 | 00 | 01 | 10 | 06 | 09 | 11 | 05 |
| 2 | 00 | 14 | 07 | 11 | 10 | 04 | 13 | 01 | 05 | 08 | 12 | 06 | 09 | 03 | 02 | 15 |
| 3 | 13 | 08 | 10 | 01 | 03 | 15 | 04 | 02 | 11 | 06 | 07 | 12 | 00 | 05 | 14 | 09 |

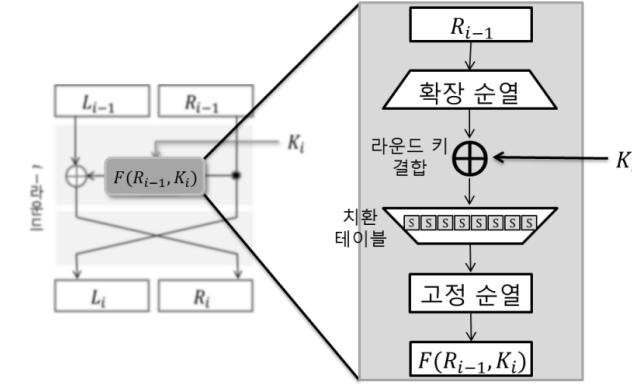
... . . .

S-box 8

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 13 | 02 | 08 | 04 | 06 | 15 | 11 | 01 | 10 | 09 | 03 | 14 | 05 | 00 | 12 | 07 |
| 1 | 01 | 15 | 13 | 08 | 10 | 03 | 07 | 04 | 12 | 05 | 06 | 11 | 10 | 14 | 09 | 02 |
| 2 | 07 | 11 | 04 | 01 | 09 | 12 | 14 | 02 | 00 | 06 | 10 | 13 | 15 | 03 | 05 | 08 |
| 3 | 02 | 01 | 14 | 07 | 04 | 10 | 8 | 13 | 15 | 12 | 09 | 00 | 03 | 05 | 06 | 11 |

■ 고정 순열

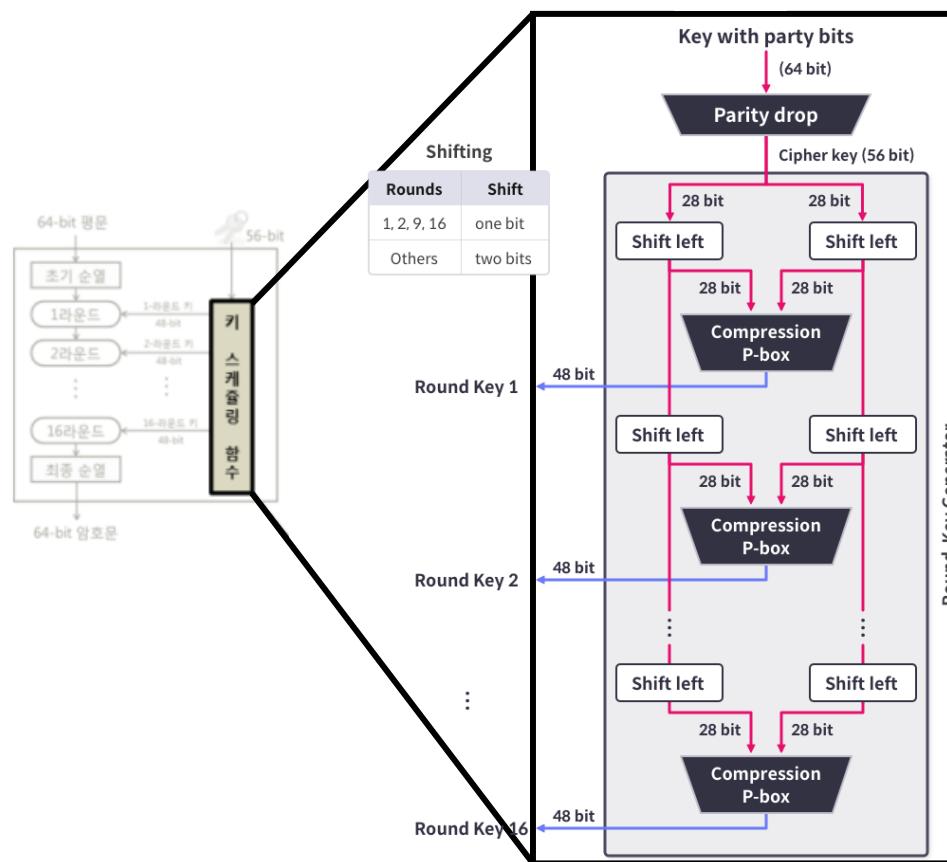
| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 16 | 07 | 20 | 21 | 29 | 12 | 28 | 17 |
| 01 | 15 | 23 | 26 | 05 | 18 | 31 | 10 |
| 02 | 08 | 24 | 14 | 32 | 27 | 03 | 09 |
| 19 | 13 | 30 | 06 | 22 | 11 | 04 | 25 |



고정 순열 표(Straight Permutation)

■ 키 스케줄링

- 함수는 64비트인 비밀키를 입력으로 받아 16개의 48비트 라운드 키를 출력



```

def key_scheduling(self):
    shift_1 = [0, 1, 8, 15]
    self.__round_keys = []

    # Drop parity bit (56bit)
    parity_erased = permutation(self.__key, self.__pbt)
    left = parity_erased[:28]
    right = parity_erased[28:]

    # Shift
    for i in range(16):
        if i in shift_1:
            left = left[1:] + left[0:1]
            right = right[1:] + right[0:1]
        else:
            left = left[2:] + left[:2]
            right = right[2:] + right[:2]

    # Compression (48bit)
    self.__round_keys.append(permutation(left + right, self.__cpt))

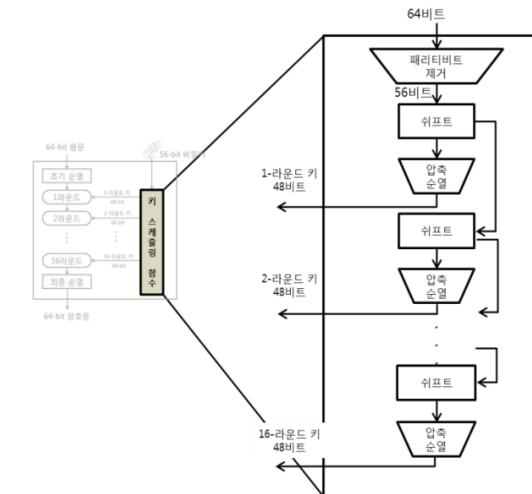
```

■ 패리티 비트 제거

- DES 암호의 64비트 비밀키 중 8개의 패리티 비트(8의 배수 비트)는 키의 안전성에 기여하지 않기 때문에, 일반적으로 DES 암호는 56비트 비밀키를 사용한다.

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 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 |

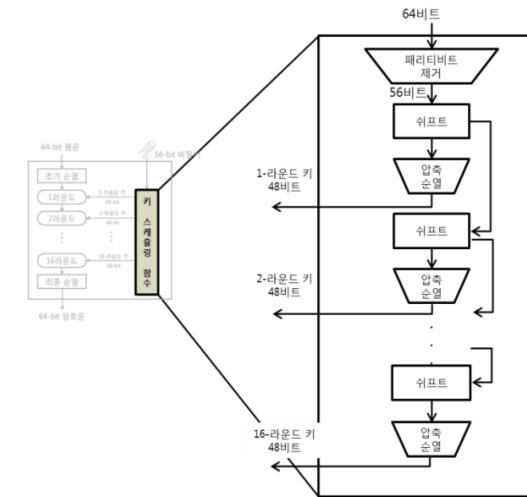
패리티 비트 제거 표



■ 쉬프트 (Shift)

$X = x_l x_{l-1} \dots x_0$,
Cyclic Shift(X, k) = X' , $i' \equiv i + k \pmod{l}$
 $X' = x_l x_{(l-1)} \dots x_0$

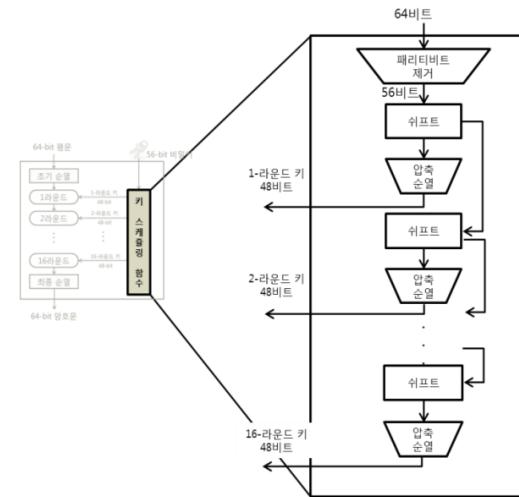
| 라운드 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| 횟수 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |



■ 압축 순열 (Compression P-Box)

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 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 |

압축 순열 표(Compression P-Box)



■ DES 복호화

- 암호화 과정과 동일한 알고리즘에 라운드 키를 역순으로 입력 받아 동작

```

def encrypt(self):
    self.key_scheduling()

    self.initial_permutation(self.__plaintxt)

    for round in range(16):
        self.round_function(round)

    self.__ciphertext = self.final_permutation()

    return bitarray2bytes(self.__ciphertext)

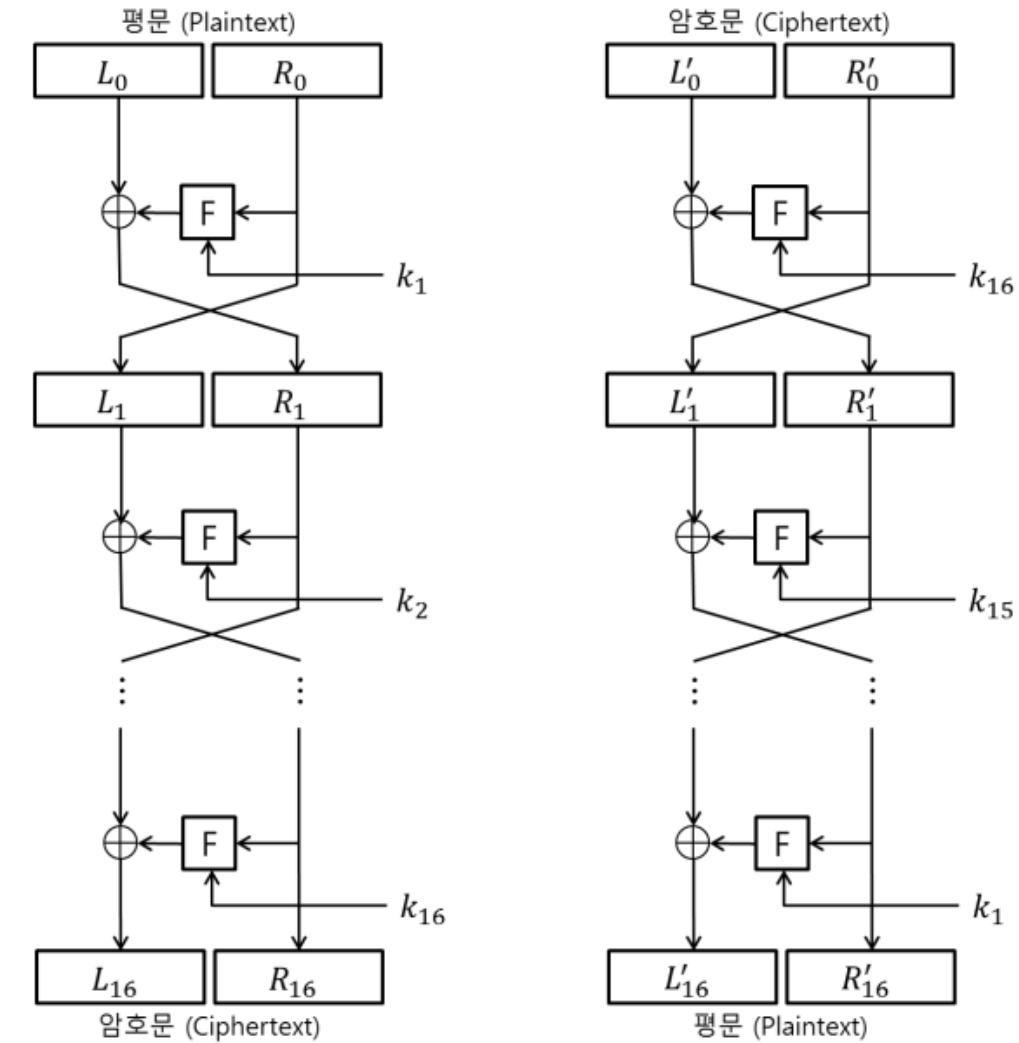
def decrypt(self):
    self.key_scheduling()

    self.initial_permutation(self.__ciphertext)

    for round in reversed(range(16)):
        self.round_function(round)

    return bitarray2bytes(self.final_permutation())

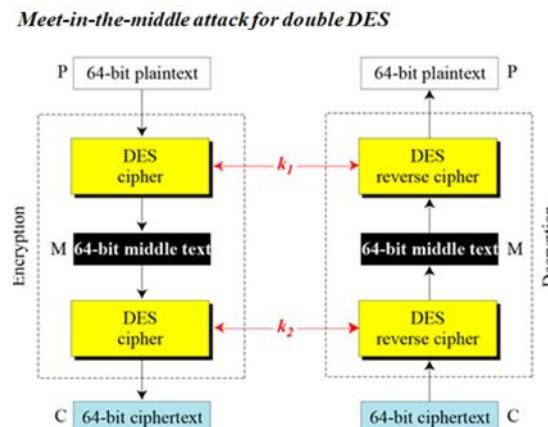
```



[추가] Meet-in-the-Middle Attack

■ Double DES

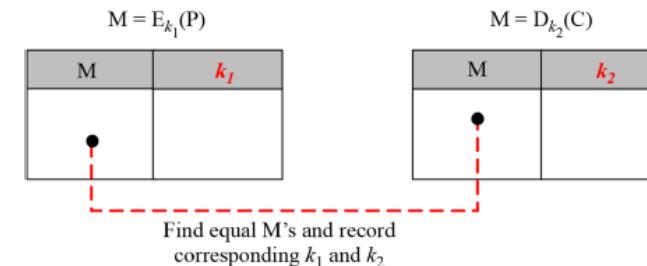
- 암호화를 위해 두 개의 DES 암호 알고리즘을 사용하고 복호화를 위해서 DES 복호화 알고리즘을 두 개 사용한다. DES 알고리즘을 2번 사용하기 때문에 Key 크기도 56 Key의 두 배인 112 Key가 된다.
- BUT,



Alice는 평문 P를 암호문 C로 암호화하기 위해 k_1 과 k_2 를 사용하고, Bob은 평문 P를 복호화하기 위해 암호문 C와 k_2 와 k_1 을 사용한다.

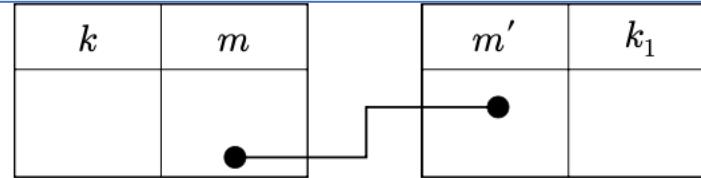
$$M = E_{k1}(P) : \text{암호화 중간텍스트}$$
$$M = D_{k2}(C) : \text{복호화 중간텍스트}$$

- ✖ using a known-plaintext attack called *meet-in-the-middle attack* proves that double DES improves this vulnerability slightly (to 2^{57} tests), but not tremendously (to 2^{112}).



✖ $2 \times 2^{56} + O(n \log n) = 2^{63}$

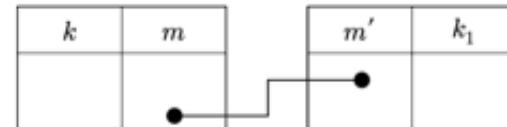
[문제#1] 암호 알고리즘의 안전성



즉, $DES_k(p)$ 의 56비트 키 k 에 대한 전수조 . . . 수행하고, $c \oplus k_1$ 의 64비트 키 k_1 에

답) 변형된 암호의 양 변에 k_1 에 대한 XOR 연산을 수행하면 $DES_k(p) = c \oplus k_1$ 과 같은 식을 만족한다. 만일 평문 p 에 대한 암호문 c 를 알고 있는 공격자가 있다면 좌변에 대한 키 전수조사와 우변에 대한 키 전수조사를 통해 중간 일치 공격을 수행할 수 있다.

$$DES_k(p) = m \quad m' = c \oplus k_1$$



즉, $DES_k(p)$ 의 56비트 키 k 에 대한 전수조사를 2^{56} 번 수행하고, $c \oplus k_1$ 의 64비트 키 k_1 에 대한 전수조사를 2^{64} 번 수행한 뒤, 두 값이 같아지는 경우($DES_k(p) = m \dots m' = c \oplus k_1$)의 k 와 k_1 을 찾아낼 수 있다. 결론적으로 56비트와 64비트 길이를 가지는 두 개의 키를 사용하여 120비트의 안전성을 제공할 것으로 생각되지만, 중간 일치 공격을 통해 알려진 평문 공격을 수행하는 공격자는 $2^{56} + 2^{64} + \alpha$ 번의 공격만 수행하면 되기 때문에 단일 DES보다는 안전성이 높아질 수 있지만 기대한 만큼의 안전성을 제공하지는 못한다.

[문제#2] 암호 알고리즘의 안전성

1990년대 후반으로 들어와서 컴퓨팅 파워와 병렬처리 기술이 발달함에 따라 DES 알고리즘에 대한 전사적 공격(Brute Force Attack)이 가능하게 되었다. 이를 보완하기 위해 3DES 와 같은 기법들이 제안되었다. Rivest 역시 같은 문제를 해결하기 위해 아래의 DESX 알고리즘을 제안하였다.

$$Enc_{DESX}(k=(k_1, k_2), m) = Enc_{DES}(k_1, m \oplus k_2) \oplus k_2$$

위 기법을 아래와 같이 변형 할 경우, 발생 가능한 공격을 서술하시오.

$$Enc_{DESX'}(k=(k_1, k_2), m) = Enc_{DES}(k_1, m) \oplus k_2$$

(k_1 : 56비트, k_2 : 64비트)

1) 공격자가 평문, 암호문 쌍 (P, C)를 있다고 하면,

$$Enc(k_1, m) \oplus k_2 = c$$

$$Enc(k_1, m) = c \oplus k_2 \text{ 이므로,}$$

위의 조건을 만족하는 모든 k_1 과 k_2 의 table 생성 후 위의 조건을 만족하는 $k = (k_1, k_2)$ 를 찾아내는 “meet in the middle attack”이 가능하다.

2) 공격자가 평문, 암호문쌍 (m_1, c_1), (m_2, c_2)를 있다고 하면,

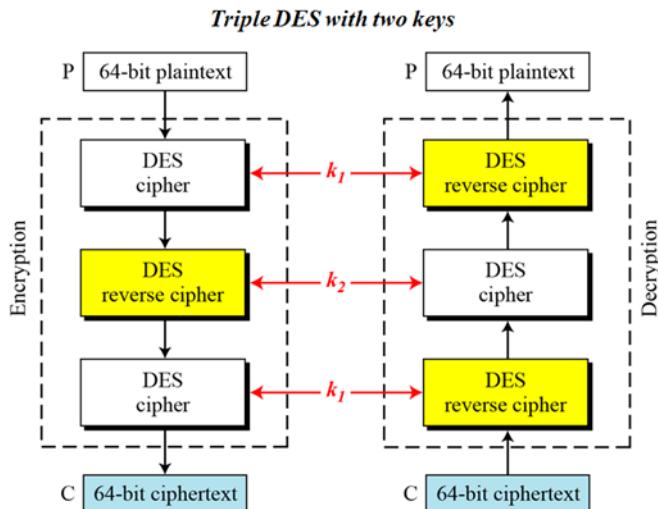
$$\textcircled{1} \quad Enc(m_1, k_1) \oplus k_2 = c_1$$

$$\textcircled{2} \quad Enc(m_2, k_1) \oplus k_2 = c_2 \text{ 이므로, } \textcircled{1} \oplus \textcircled{2} \text{ 하면,}$$

$Enc(m_1, k_1) \oplus Enc(m_2, k_1) = c_1 \oplus c_2$ 으로 k_1 에 대한 수식으로 변경할 수 있고, 이는 “ k_1 에 대한 전수조사”를 실행하여 공격 가능하다.

[문제#3] 암호 알고리즘의 안전성

Triple DES에서 EDE 구조(Encrypt-Decrypt-Encrypt)를 사용하는 이유를 서술하시오.



03

AES
(Advanced Encryption Standard)

■ 배경

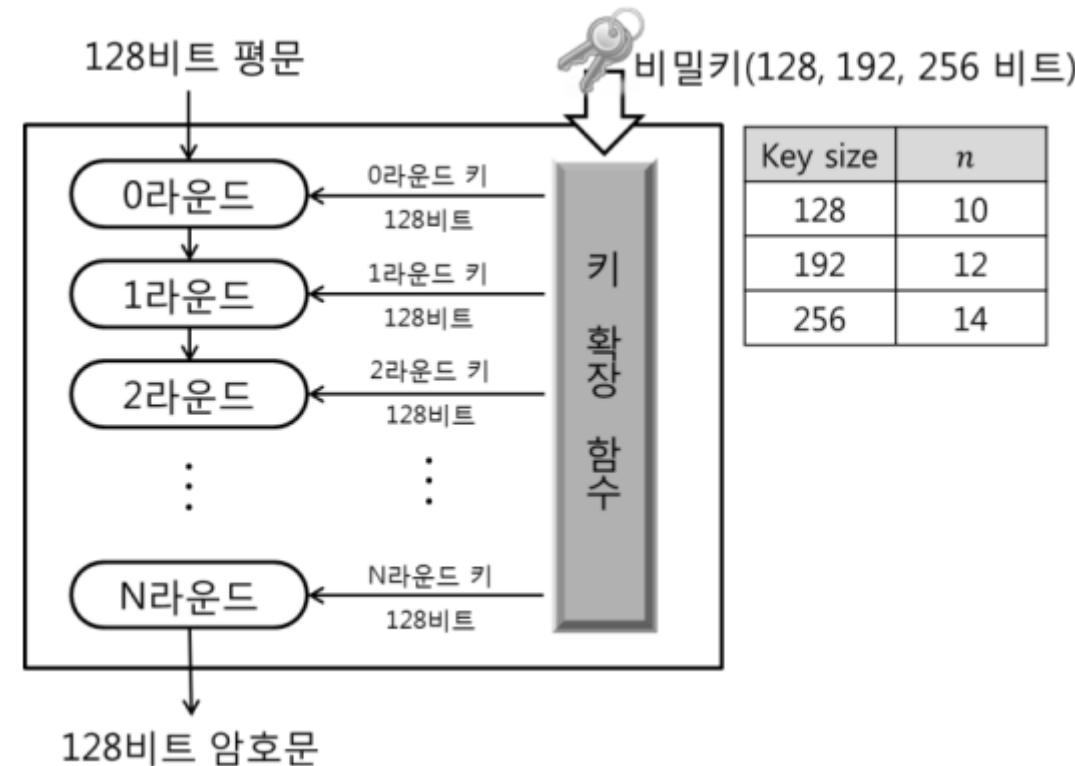
- DES의 2^{56} 개의 키에 대한 전사적 공격이 가능
 - 1999년 distributed.net과 Electronic Frontier Foundation이 협력한 공격에서 DES의 비밀키를 22시간 15분만에 찾아냄
- TDES가 있지만 다음 이유로 NIST에서는 AES 공모
 - TDES는 DES를 세 번 사용하기 때문에 속도가 느림
 - DES의 블록 크기인 64비트는 여러 가지 응용분야에 적합하지 않다. 예로 블록 암호를 이용하여 설계한 해쉬 함수의 경우 64비트의 블록 크기는 해쉬 함수의 안전성에 문제
 - 양자컴퓨터가 현실화 될 경우 적어도 256비트 크기의 키가 바람직

■ AES 공모 시 요구사항

- 블록의 크기는 128비트 이상
- 대칭키 암호이며 세 종류의 키(128비트, 192비트, 256비트)를 사용할 수 있어야 함
- 소프트웨어와 하드웨어로 구현될 경우 모두 효율적
- 모든 키를 다 찾는 전수 키 조사 이외에 현재 알려진 다른 암호 분석 공격에 강해야 함

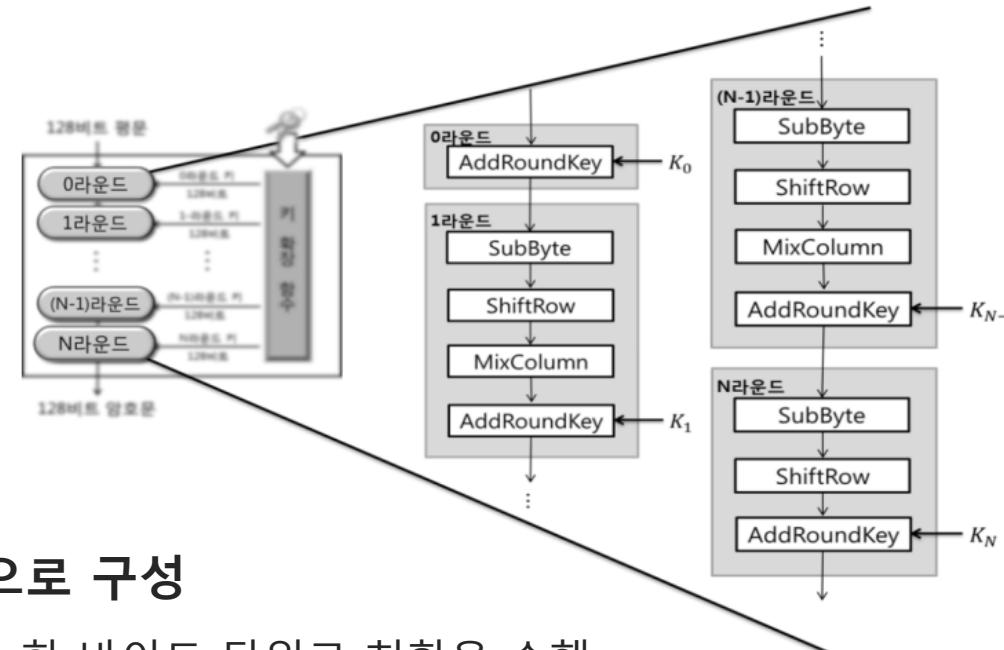
■ AES 구조

- 블록의 크기: 128비트
- 128, 192, 256비트의 비밀키에 대해 라운드의 수는 각각 10, 12, 14라운드가 실행



■ 한 블록인 16바이트(=128비트)는 원소가 한 바이트인 4*4행렬로 변환됨

- 이 행렬을 상태(state)라 부름

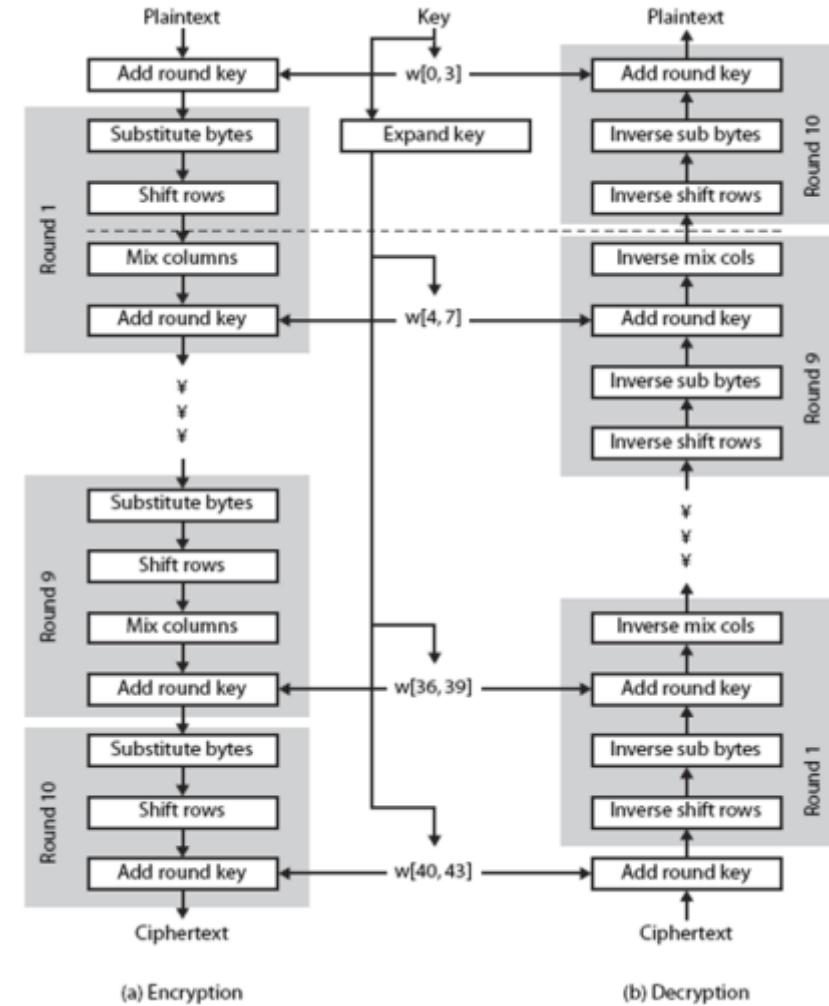


■ 한 라운드는 네 가지 계층(Layer)으로 구성

- SubBytes: DES의 S-Box에 해당하며 한 바이트 단위로 치환을 수행
 - 상태(state)의 한 바이트를 대응되는 S-Box의 한 바이트로 치환한다. 이 계층은 혼돈의 원리를 구현한다.
- ShiftRows: 상태의 한 행안에서 바이트 단위로 자리바꿈이 수행
- MixColumns: 상태가 한 열안에서 혼합이 수행. ShiftRows와 함께 분산의 원리를 구현
- AddRoundKey: 비밀키(128/192/256비트)에서 생성된 128비트의 라운드 키와 상태가 XOR됨

■ Encryption and Decryption

- Iterative이기 때문에 모든 component가 inversable해야 함
- Round Key는 DES와 동일하게 역순임



■ 블록이 상태(State)의 형태로 표현

- 상태는 원소가 한 바이트인 4*4 행렬
- AES의 한 블록이 "EASYCRYPTOGRAPHY"인 경우

16 바이트

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| E | A | S | Y | C | R | Y | P | T | O | G | R | A | P | H | Y |
| 04 | 00 | 12 | 18 | 02 | 11 | 18 | 0F | 13 | 0E | 06 | 11 | 00 | 0F | 07 | 18 |

(텍스트를 16진수로 표현)

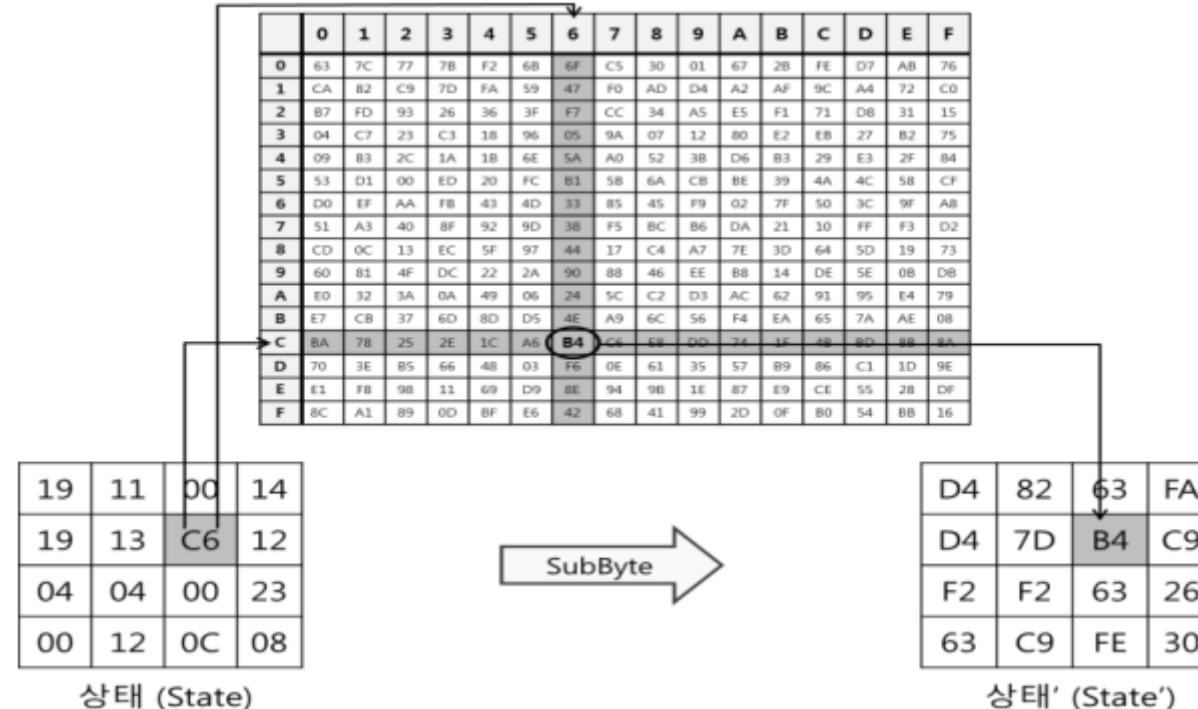
상태(State) 4x4



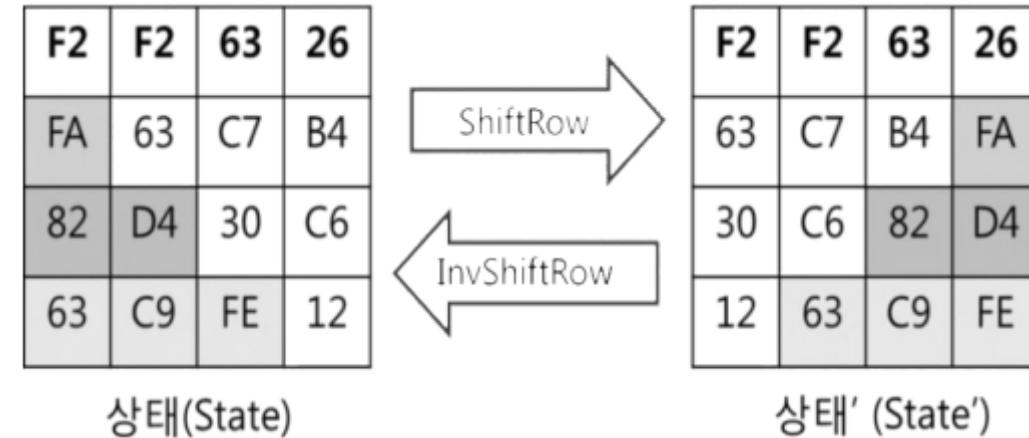
| | | | |
|----|----|----|----|
| 04 | 02 | 13 | 00 |
| 00 | 11 | 0E | 0F |
| 12 | 18 | 06 | 07 |
| 18 | 0F | 11 | 18 |

■ Substitute Bytes(SubBytes) 계층

- 한 원소가 16진수로 (xy)인 경우 상위 4비트 값인 x가 S-Box의 행을 결정하고 하위 4비트 값인 y가 열을 결정



■ ShiftRows 계층



■ MixColumns 계층

- SubBytes 계층과 ShiftRows 계층은 바이트 단위로 처리
- 충분한 분산 효과를 발생시키기 위하여, MixColumns 계층에서는 상태의 각 열을 비트 단위로 섞어 줌

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

$$s'_{0,0} = (02 \cdot s_{0,0}) \oplus (03 \cdot s_{1,0}) \oplus (01 \cdot s_{2,0}) \oplus (03 \cdot s_{3,0})$$

$$s'_{1,0} = (01 \cdot s_{0,0}) \oplus (02 \cdot s_{1,0}) \oplus (03 \cdot s_{2,0}) \oplus (01 \cdot s_{3,0})$$

.

.

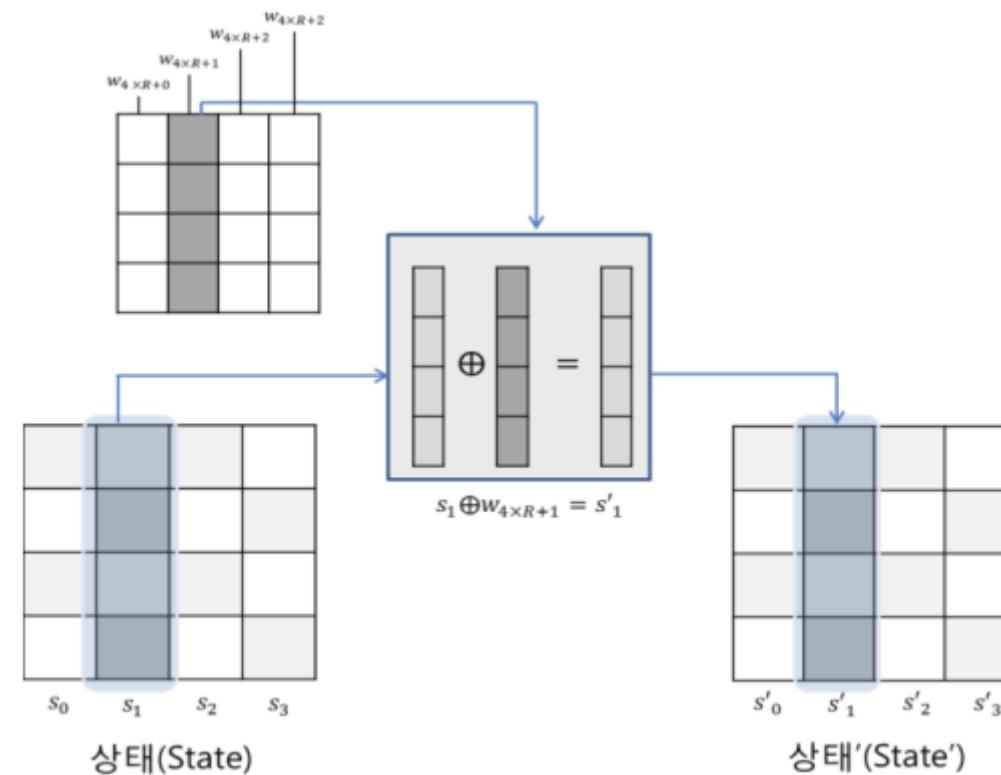
.

$$s'_{3,3} = (03 \cdot s_{0,3}) \oplus (01 \cdot s_{1,3}) \oplus (01 \cdot s_{2,3}) \oplus (02 \cdot s_{3,3})$$

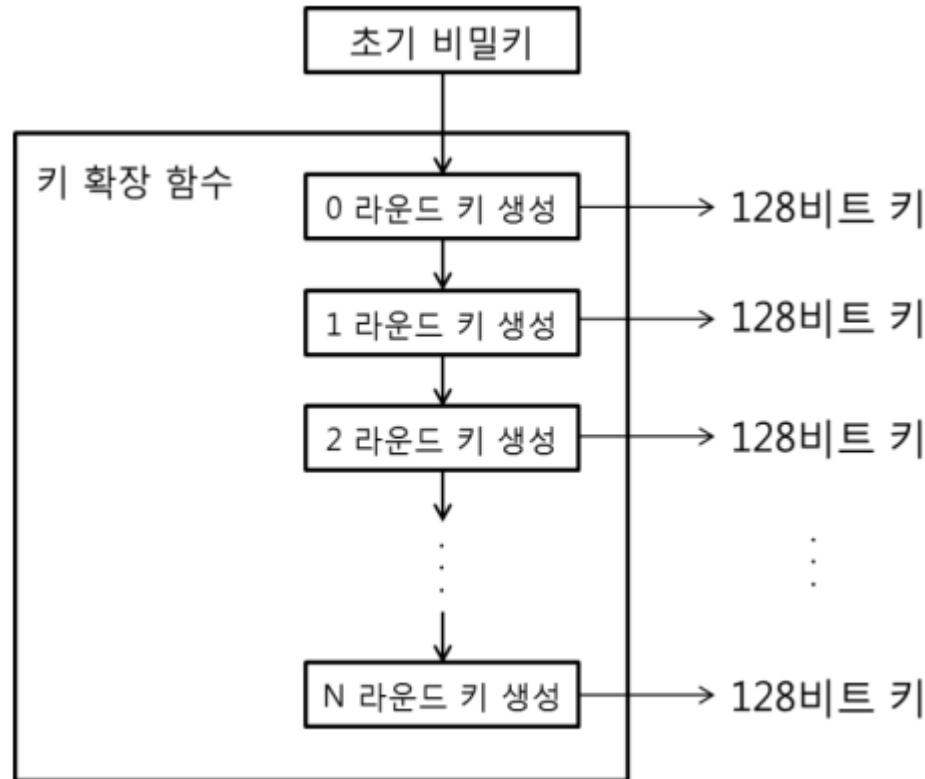
■ Inverse MixColumns 계층

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

■ AddRoundKey



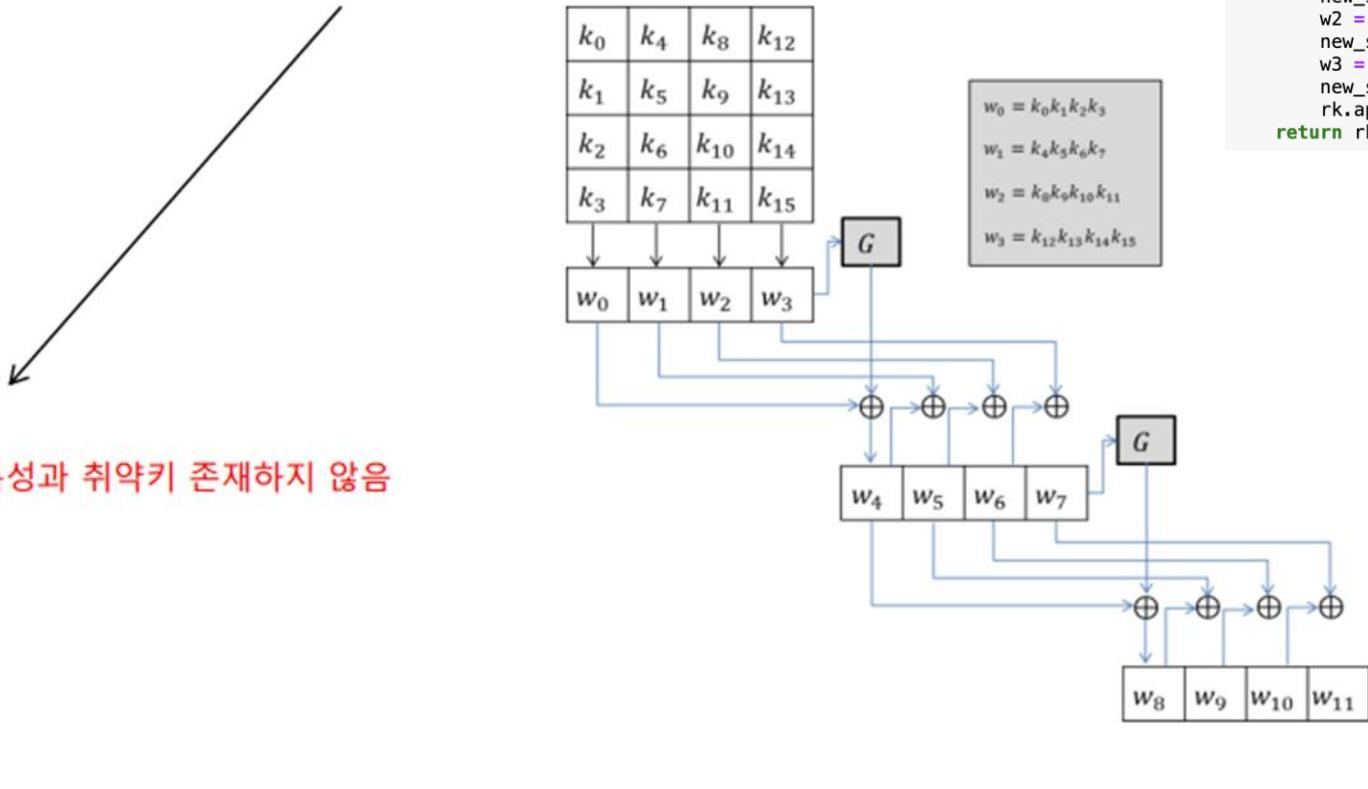
■ 키 확장 (Key Expansion)



| 라운드 | 워 드 | | | |
|------|----------|------------|------------|------------|
| 0라운드 | w_0 | w_1 | w_2 | w_3 |
| 1라운드 | w_4 | w_5 | w_6 | w_7 |
| 2라운드 | w_8 | w_9 | w_{10} | w_{11} |
| ... | ... | | | |
| N라운드 | w_{4N} | w_{4N+1} | w_{4N+2} | w_{4N+3} |

■ Key Expansion

$$G(w_{4i-1}) = \text{SubWord}(\text{RotWord}(w_{4i-1})) \oplus RCons,$$



```

def t_gen(col, r):
    new_col = [col[1], col[2], col[3], col[0]]
    return xor_col([SBT[new_col[i]] for i in range(4)], [RCONS[r-1], 0, 0, 0])

def key_expansion(key_state):
    rk = [copy.deepcopy(key_state)]
    for r in range(1, 11):
        new_state = []
        w0 = xor_col(rk[r-1][0], t_gen(rk[r-1][3], r))
        new_state.append(w0)
        w1 = xor_col(rk[r-1][1], w0)
        new_state.append(w1)
        w2 = xor_col(rk[r-1][2], w1)
        new_state.append(w2)
        w3 = xor_col(rk[r-1][3], w2)
        new_state.append(w3)
        rk.append(new_state)
    return rk

```

04

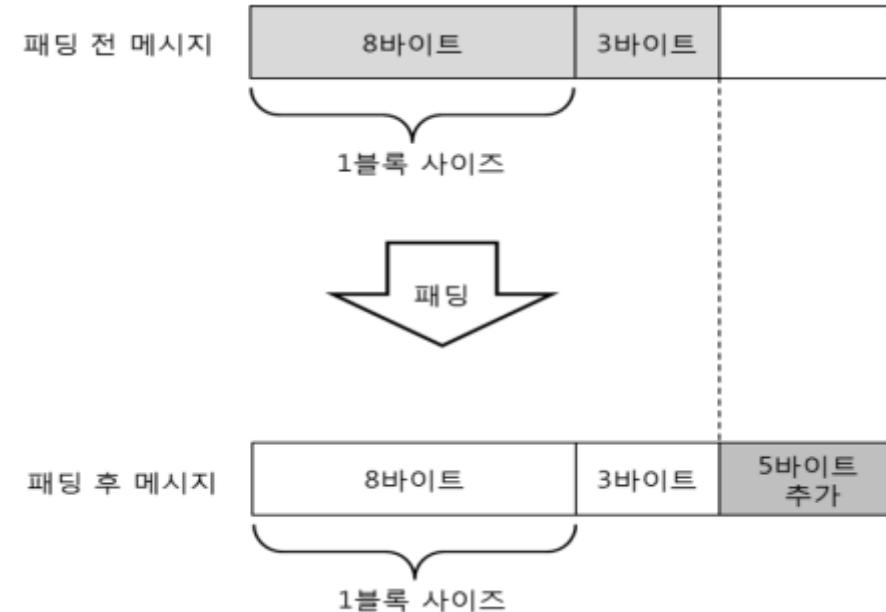
Mode of Operation

■ 운영모드란?

- DES나 AES와 같은 블록 암호를 사용하여 다양한 크기의 데이터를 암호화 하는 방식
- 실제로 사용되는 평문은 다양한 크기를 가지며 보통 블록크기보다 훨씬 큰 데이터
- ECB, CBC, CFB, OFB, CTR, ...

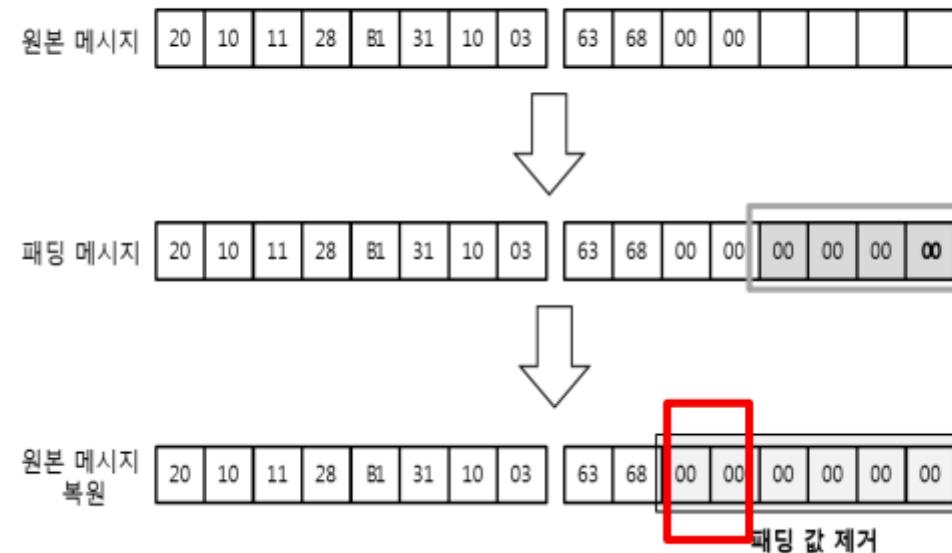
■ 블록 Cipher의 경우, 평문의 길이가 정확하게 해당 블록 암호의 블록 크기의 배수가 되어야 함

- 패딩은 평문의 전체가 블록 크기의 배수가 되도록 마지막 부분의 빈 공간을 채워 하나의 완전한 블록으로 만드는 작업



■ 제로 패딩 (Zero padding, Null Padding)

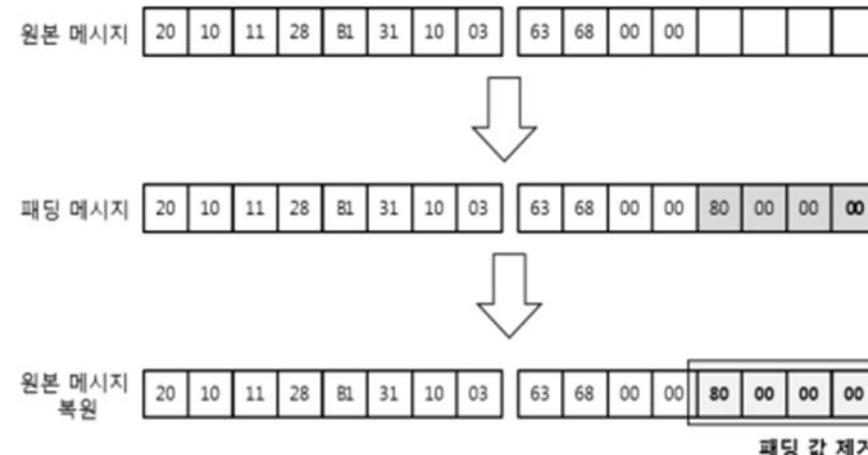
... | 31 AB 34 FE 52 5E 97 12 | 3A FE 5A **00 00 00 00 00** |₍₁₆₎



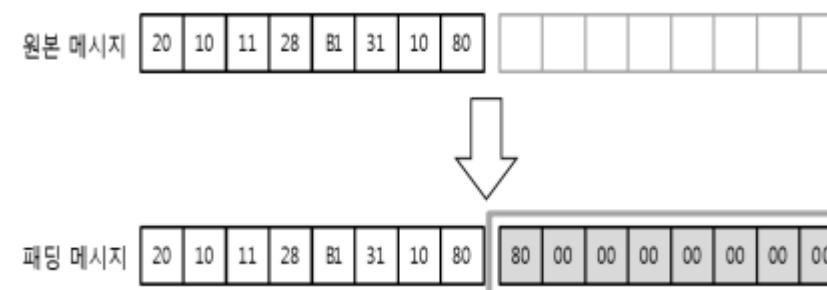
■ 비트 패딩 (Bit Padding)

... | 1001 1101 0011 1110 | 1101 1001 **1000 0000** | ₍₂₎

최상위 비트



- 메시지 길이가 블록 크기의 배수일 때 비트 패딩



■ 바이트 패딩 (Byte Padding)

최하위 바이트에 패딩
바이트 표시

FE 52 5E 97 12 | 3A FE 5A **00 00 00 00 05** |₍₁₆₎

원본 메시지 [20 10 11 28 B1 31 10 03] [63 68 00 00] [] [] []



패딩 메시지 [20 10 11 28 B1 31 10 03] [63 68 00 00] [00 00 00 00 04]



원본 메시지
복원 [20 10 11 28 B1 31 10 03] [63 68 00 00] [00 00 00 04]

패딩 값 제거

- 메시지 길이가 블록 크기의 배수일 때 바이트 패딩

원본 메시지 [20 10 11 28 B1 00 00 03] [] [] [] [] [] []



패딩 메시지 [20 10 11 28 B1 00 00 03] [00 00 00 00 00 00 00 08]

■ PKCS7 패딩

- 패딩 바이트 값을 패딩 바이트 크기로 사용

원본 메시지

| | | | |
|----|----|----|----|
| 23 | AF | 4E | 30 |
| AB | 3E | 7F | 97 |
| 64 | 64 | 90 | 5E |
| 6F | 26 | 8A | 6F |

패딩 메시지

| | | | |
|----|----|----|----|
| 23 | AF | 4E | 30 |
| AB | 3E | 7F | 97 |
| 64 | 64 | 90 | 5E |
| 6F | 26 | 8A | 6F |
| 06 | 06 | 06 | 06 |



원본 메시지

| | | | |
|----|----|----|----|
| 23 | AF | 4E | 30 |
| AB | 3E | 7F | 97 |
| 64 | 64 | 90 | 5E |
| 6F | 26 | 8A | 6F |
| 3E | AC | 68 | 20 |

패딩 메시지

| | | | |
|----|----|----|----|
| 23 | AF | 4E | 30 |
| AB | 3E | 7F | 97 |
| 64 | 64 | 90 | 5E |
| 6F | 26 | 8A | 6F |
| 3E | AC | 68 | 20 |
| 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 |

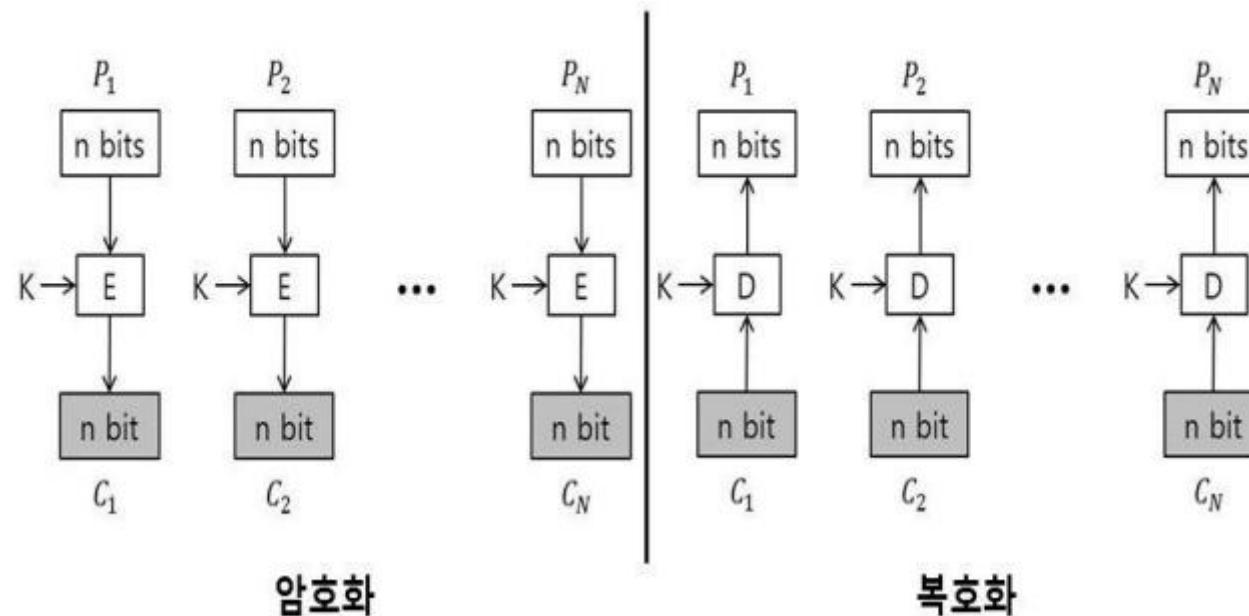


Electronic Code Block (ECB) 모드

- 한 블록의 평문은 한 블록의 암호문으로 암호화된다.

※ 암호화 : $C_i = E_K(P_i)$

복호화 : $P_i = D_K(C_i)$



■ 장점

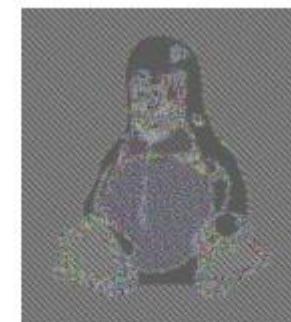
- 병렬 처리 가능 & 오류 확산 없음

■ 단점

- 같은 평문에 대해 같은 암호문
- 즉, 블록 단위의 패턴 유지



Original



Encrypted using ECB
mode



Encrypted using other
modes

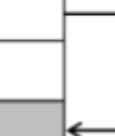
■ 단점

- 같은 평문에 대해 같은 암호문
- 즉, 블록 재사용 (Block Replay) 공격 가능

| 이름 | 암호화된 점수 (원본 점수) |
|-------|-----------------|
| Alice | 0F14D3F2 (90) |
| Bob | 3DE9001F (80) |
| Eve | 549F2D4F (50) |



| 이름 | 암호화된 점수 (원본 점수) |
|-------|----------------------|
| Alice | 0F14D3F2 (90) |
| Bob | 3DE9001F (80) |
| Eve | 0F14D3F2 (90) |

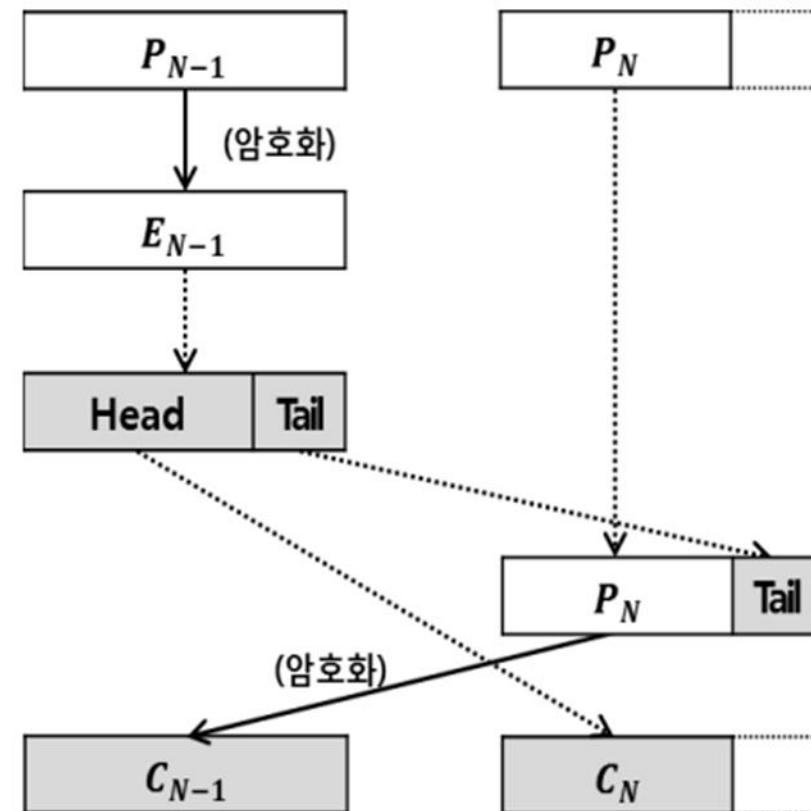


■ 암호문 스틸링 기법 (Ciphertext Stealing)

- * $X = E_K(P_{N-1})$
 \rightarrow
- $C_N = head_m(X)$
- * $Y = P_N | tail_{n-m}(X)$
 \rightarrow
- $C_{N-1} = E_K(Y)$

$head_m$: 왼쪽 최상위 m 비트
를 선택하는 함수

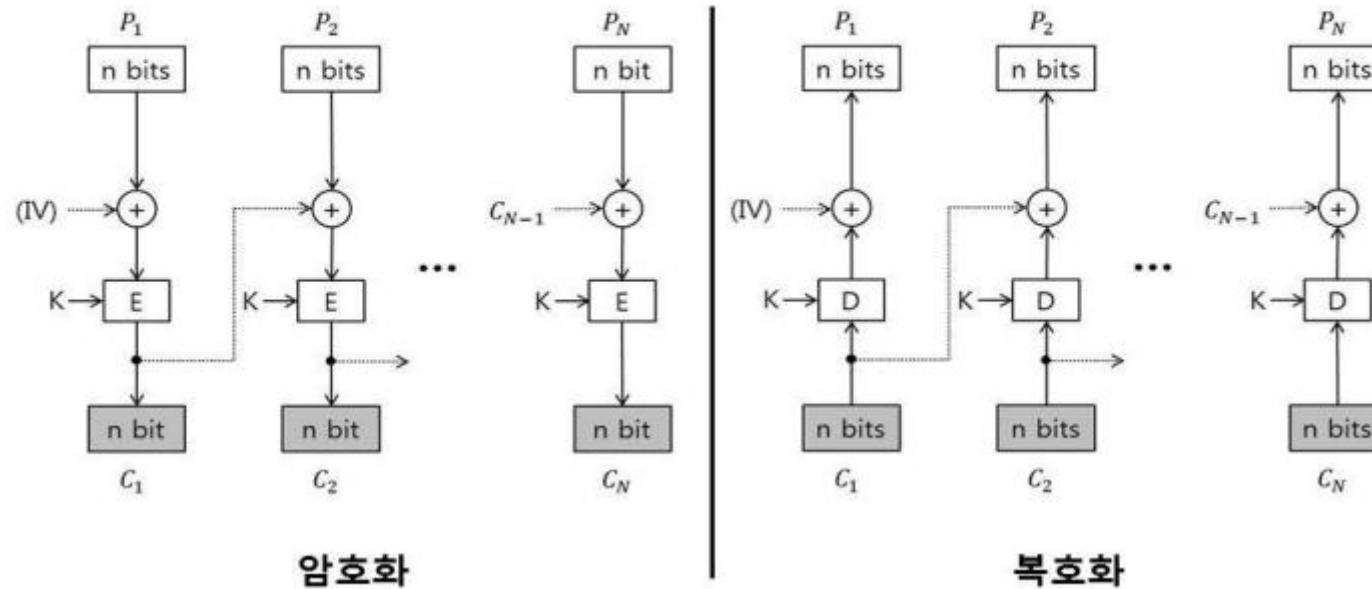
$tail_{n-m}$:오른쪽 최상위
 $(n - m)$ 비트를 선택하는 함
수



Cipher Block Chaining (CBC) 모드

- 한 평문 블록이 암호화 되기 이전에 바로 앞 평문 블록의 암호문과 XOR

$$\begin{aligned} \text{암호화} : C_0 &= IV, \quad C_i = E_K(P_i \oplus C_{i-1}), \quad i = 1, 2, 3, \dots, N \\ \text{복호화} : C_0 &= IV, \quad P_i = D_K(C_i) \oplus C_{i-1}, \quad i = 1, 2, 3, \dots, N \end{aligned}$$



■ 초기 벡터 (IV, Initialization Vector)

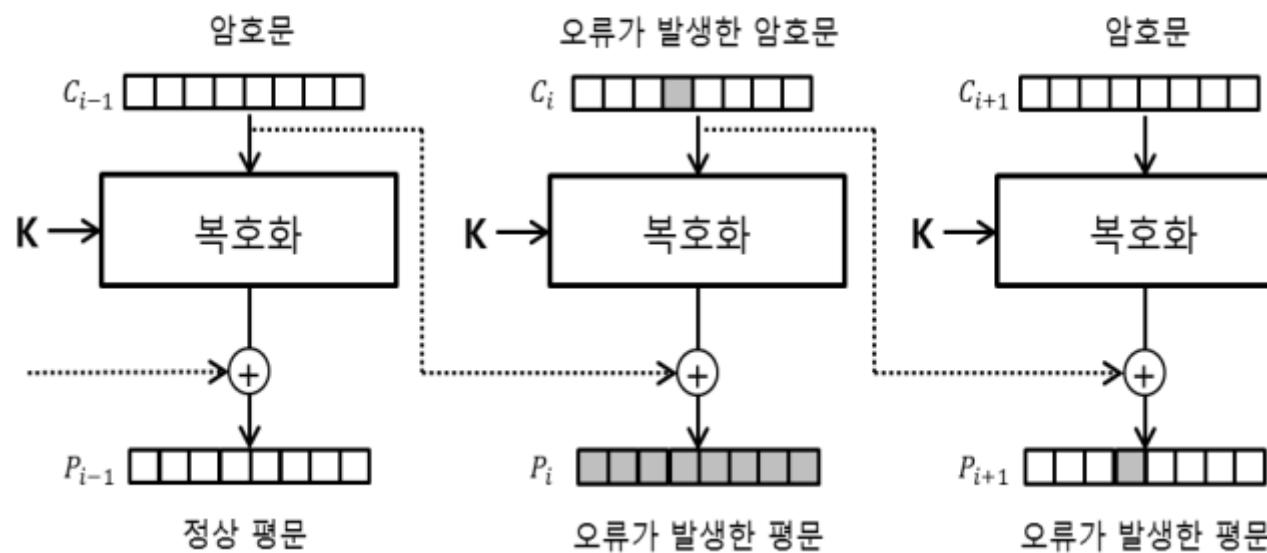
- 평문을 암호화할 때마다 초기 벡터(IV)를 바꿈으로 임의화(Randomization)
 - 확률적 암호 알고리즘 (Probabilistic Encryption Algorithm)
- 동일한 평문이 암호화될 때마다 통계적으로 독립된 서로 다른 암호문이 생성되는 성질
- 현대 암호에서는 반드시 만족되어야 하는 성질
- ECB 모드의 경우 동일한 평문에 대하여 동일한 암호문이 생성됨
 - 결정적 암호 알고리즘 (deterministic encryption algorithm)
- Nonce 사용
 - 난수를 만들어 송신자가 수신자에게 그대로 보내는 방법
 - 서로 동기화된 카운터(counter)를 사용
- 안전성을 강화하기 위하여 Nonce를 ECB모드로 암호화하여 생성된 암호문을 IV로 사용할 수도 있다.
 - ECB모드에 사용되는 키는 송신자와 수신자가 사전에 공유
- **실제 환경에서 IV의 비밀성이 아니라 무결성이 중요**
 - 만약 공격자가 전송되는 IV의 한 비트를 변경시킨다면 수신자는 제대로 된 평문을 얻을 수 없기 때문

■ 연결성 (chaining)

- 한 평문 안에 동일한 두 개의 블록에 대응되는 암호문 블록이 상이
- ECB 모드에서 보이는 평문의 블록 패턴들이 CBC의 암호문에서는 더 이상 보이지 않게 됨
- 블록단위의 재사용 공격 불가능

■ 오류 확산 (Error Propagation)

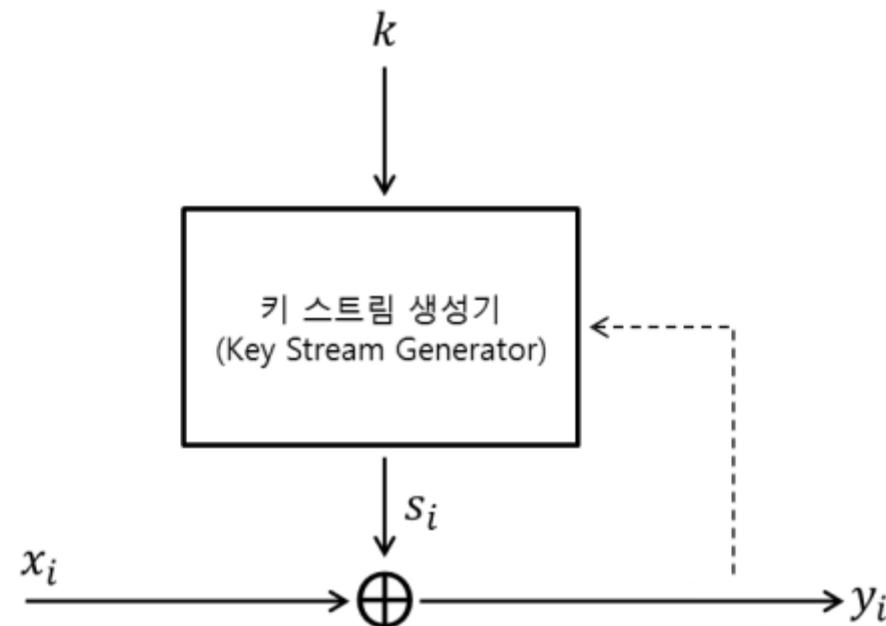
stage i : $D_k(c_i) \oplus c_{i-1} = P_i$
 stage $(i+1)$: $D_k(c_{i+1}) \oplus c_i = P_{i+1}$
 after stage $(i+1)$, CBC is **self-recovering**.



■ 스트림 암호

암호화 : $c_i = E_{s_i}(p_i) = p_i \oplus s_i$

복호화 : $p_i = D_{s_i}(c_i) = c_i \oplus s_i.$

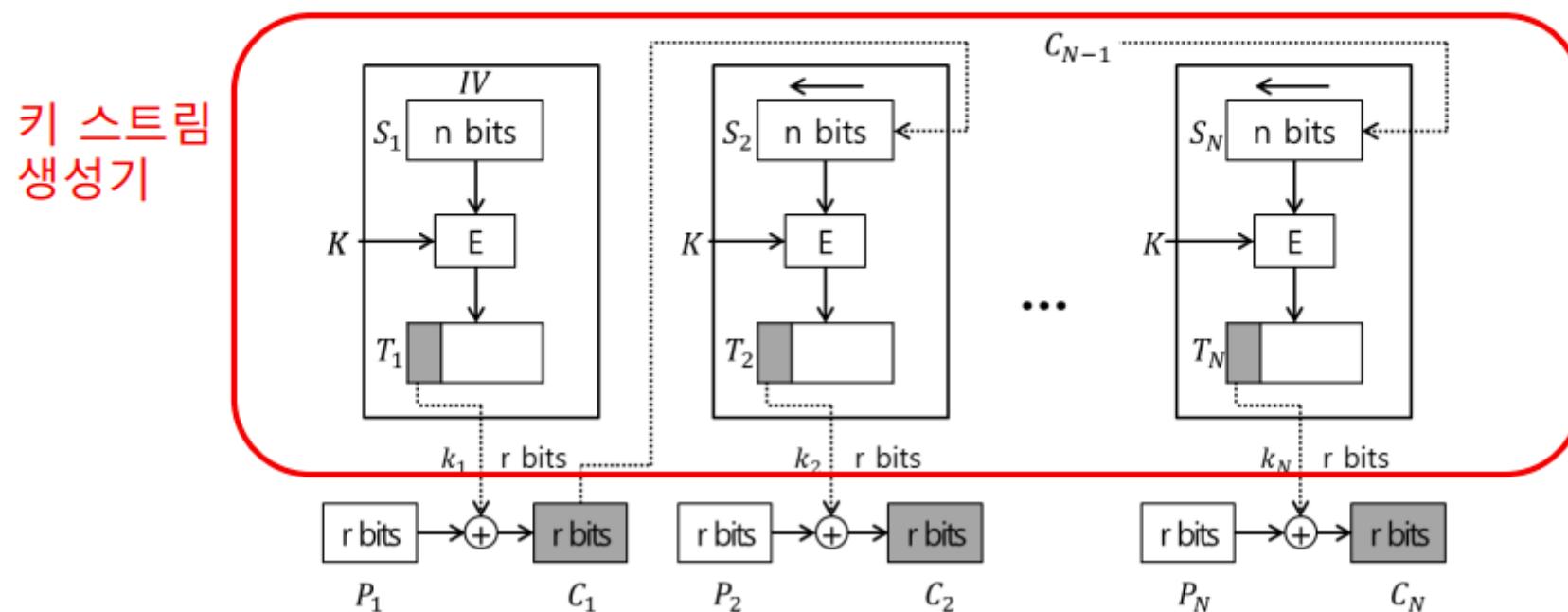


■ CFB, OFB, CTR 모드는 스트림 암호를 만들기 위해서 사용

- 블록 단위보다 작은 단위로 암호화를 진행

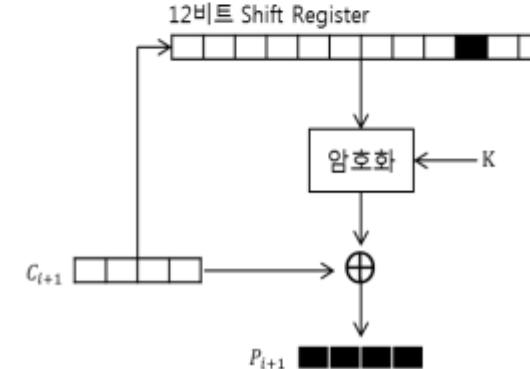
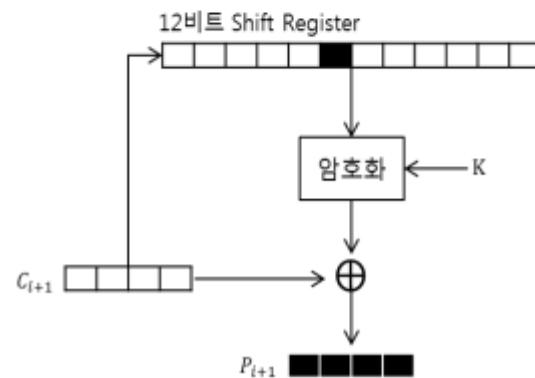
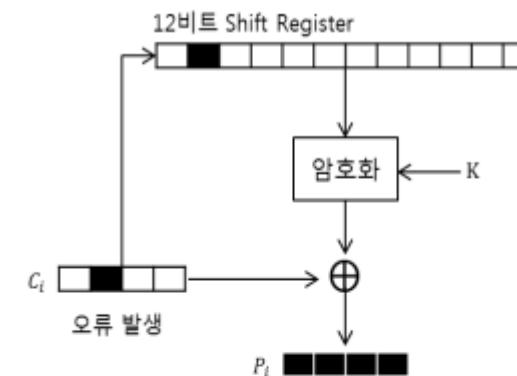
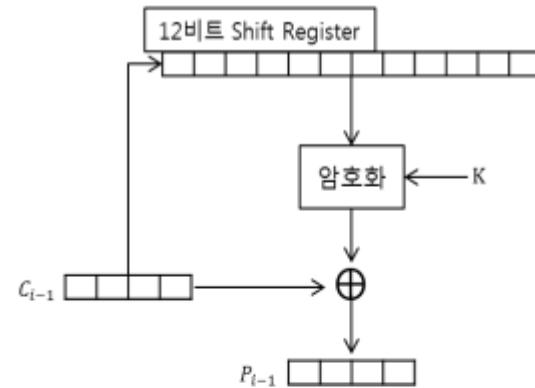
암호화 : $C_i = P_i \oplus \text{SelectLeft}_r\{E_K[\text{ShiftLeft}_r((S_{i-1})|C_{i-1})]\}, S_1 = IV,$

복호화 : $P_i = C_i \oplus \text{SelectLeft}_r\{E_K[\text{ShiftLeft}_r(S_{i-1})|C_{i-1}]\}, S_1 = IV,$



■ 오류 확산 (Error Propagation)

- 자기 동기식 (Self-synchronizing)



Output FeedBack (OFB) 모드

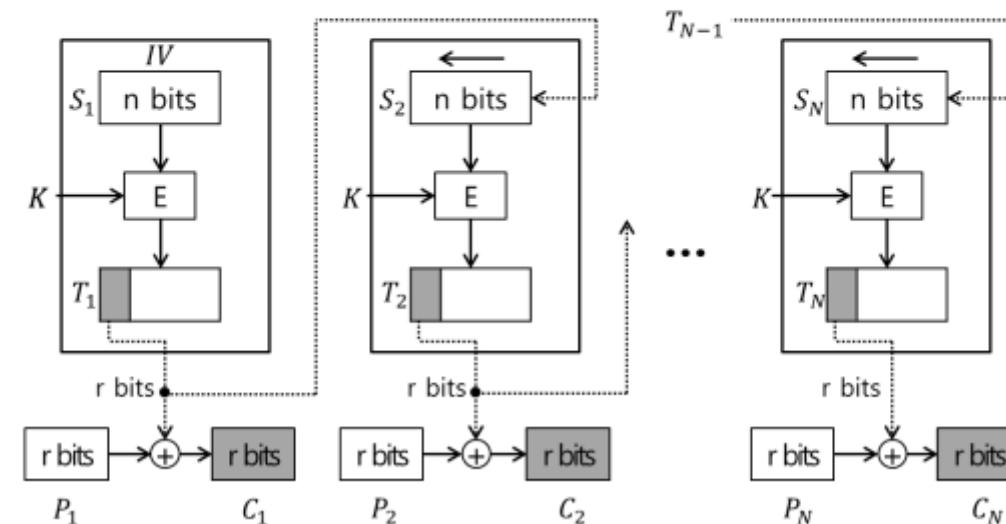
■ 동기식 스트림 암호 (synchronizing stream cipher)

- 암호화와 복호화 과정에서 XOR연산되는 키 스트림이 평문과 암호문에 독립적

암호화 : $C_i = P_i \oplus k_i, i = 1, 2, 3, \dots, N$

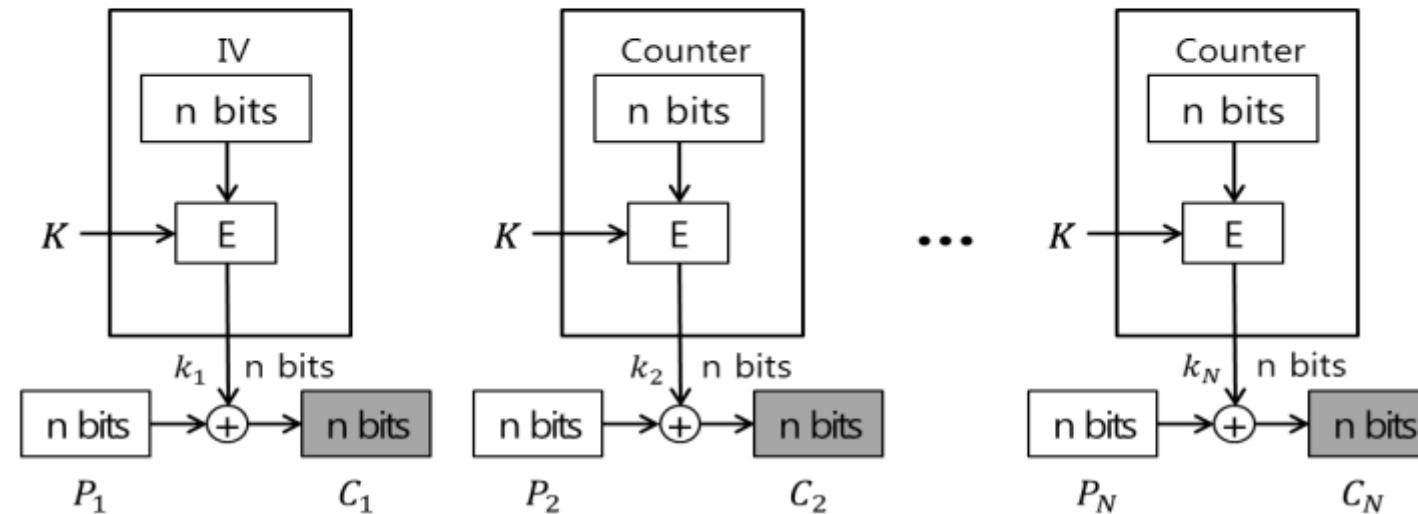
복호화 : $P_i = C_i \oplus k_i, i = 1, 2, 3, \dots, N$

▶ $k_i = SelectLeft_r(E_K(S_i)), S_1 = IV, S_i = ShiftLeft_r(S_{i-1})|k_{i-1} (i = 2, 3, \dots, N)$



- CTR이 암호화됨 → 단 모든 평문 블록마다 CTR은 달라야 함

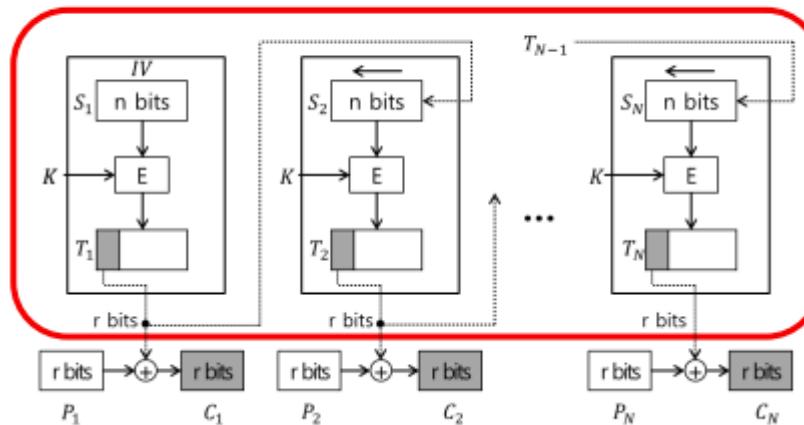
- ▶ 가장 간단한 방법은 $CTR := CTR + 1$
 - ✖ 전처리, 병렬처리 가능
 - 암호화 : $C_i = P_i \oplus E_K(\text{Counter})$, $i = 1, 2, 3, \dots, N$
 - 복호화 : $P_i = C_i \oplus E_K(\text{Counter})$, $i = 1, 2, 3, \dots, N$



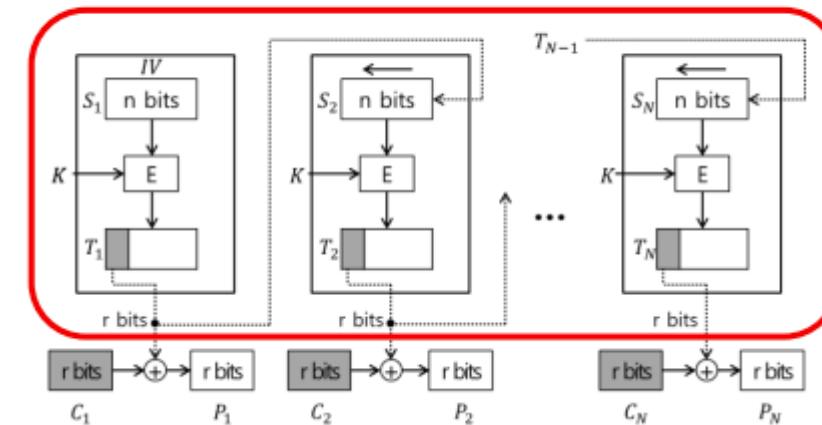
■ 전처리

- OFB

OFB 암호화 과정



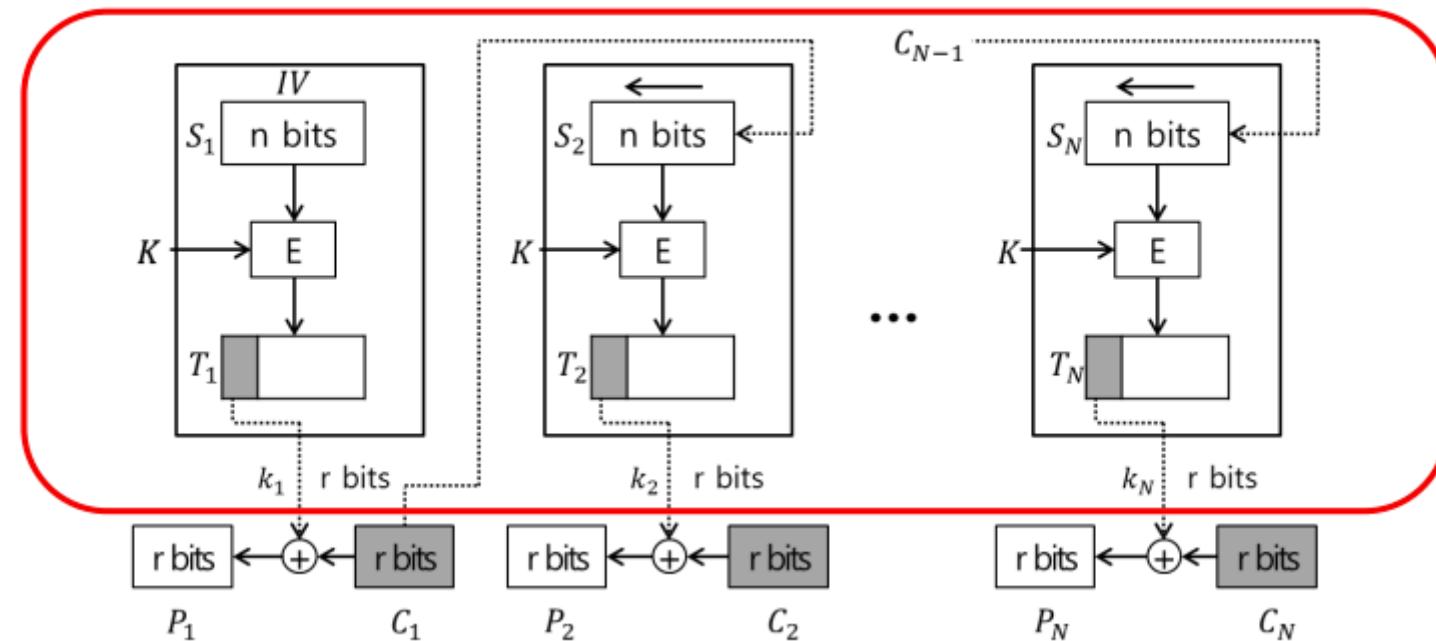
OFB 복호화 과정



■ 병렬처리

- CFB의 복호화시: 암호문을 이용하여 Shift Register를 구성

CFB 복호화 과정



각 운영모드의 특징 비교 (cont'd)

| | ECB | CBC | CFB | OFB | CTR |
|----------|-----|----------------------------|--|------------|------------|
| 블록 패턴 유지 | ○ | × | × | × | × |
| 전처리 가능성 | × | × | × | ○ | ○ |
| 병렬 처리 | ○ | 복호화시 가능 | 복호화시 가능 | × | ○ |
| 오류 확산 | × | (P_i, P_{i+1}) 블록에 영향 | $(P_i, P_{i+\lceil \frac{n}{r} \rceil})$ 블록에 영향 | × | × |
| 암호화 단위 | n | n | $r \leq n$ | $r \leq n$ | $r \leq n$ |

<https://tophix.com/ko/development-tools/encrypt-text>

Thank You



- Lab: <https://mose.kookmin.ac.kr>
- Email: sh.jeon@kookmin.ac.kr