



//부 암호- 대칭키 암호

- 정보보호 기능의 가장 핵심적 기술인 암호를 다룬다. 흥미로운 암호의 역사를 소개하고, 고전적인 암호체계로부터 현대적인 디지털 암호체계에 이르는 기술의 발전을 살펴보고 현대의 고급 암호분석 기법을 소개한다.

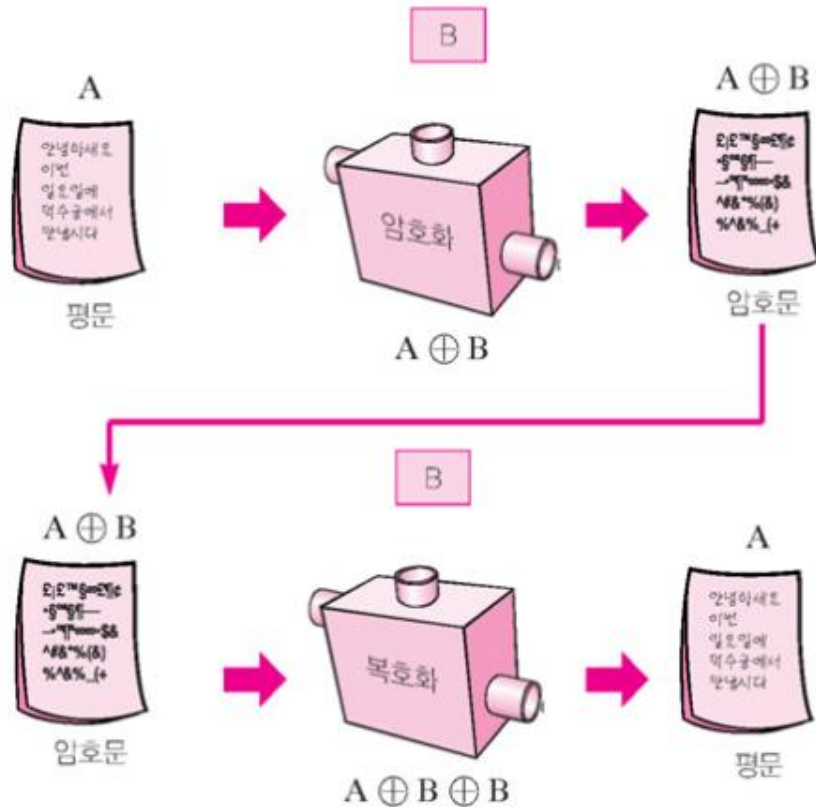
3강. 비밀키 암호

학습목표

- ✓ 비밀키 암호체계
 - ✓ 스트림 암호 A5/1, RC4
 - ✓ 블록 암호 DES
- ✓
- ✓ 블록암호를 사용해 데이터 무결성을 보장하는 방법

Section 01 개요

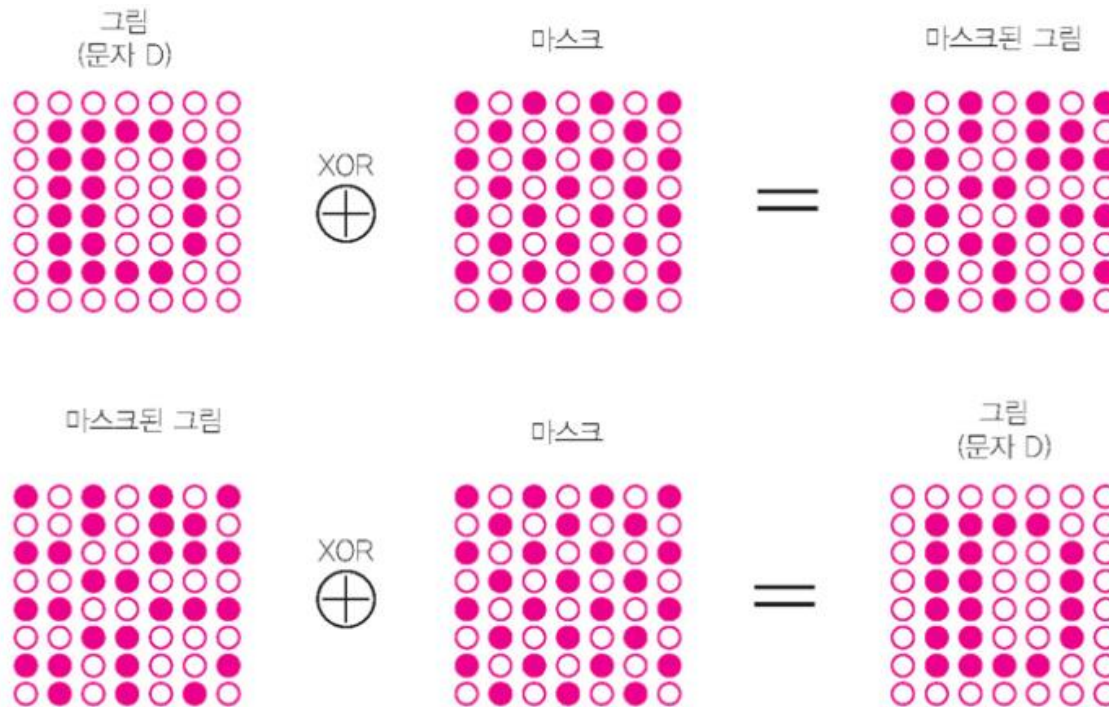
XOR 연산을 이용한 암호화와 복호화



- 평문 A를 키 B로 암호화해서 암호문 $A \oplus B$ 를 얻는다
- 암호문 $A \oplus B$ 를 키 B로 복호화해서 평문 A를 얻는다

Section 01 개요

- 한 비트열에 XOR 연산을 2회 되풀이
 - XOR는 그림을 마스크한다



Section 01 개요

비밀키(대칭키) 암호 분류

- 스트림 암호(Stream cipher) - 일회성 암호 형태
 - 키가 상대적으로 짧다.
 - 키가 긴 키 스트림으로 쪽 뺀다.
 - 키스트림은 일회성 암호 키 같이 사용된다.
 - 암호화 방식 : 평문 XOR 키 스트림
 - 암호화 단위 : 비트
- 블록 암호 – 코드북(codebook) 개념 형태
 - 블록암호 키가 한 권의 코드북을 결정한다.
 - 각 키가 다른 코드북을 결정한다.
 - 혼돈과 확산 모두가 적용된다.

	스트림 암호	블록 암호
장점	암호화 속도가 빠름, 에러 전파 현상 적음	높은 확산, 기밀성, 해시함수 등 다양
단점	낮은 확산	느린 속도, 에러 전달
사례	LFSR, MUX Generator	DES, IDEA, SEED, RC5, AES
주요대상	음성, 오디오 비디오 스트리밍	일반 데이터 전송, 스토리지 저장

Section 02 스트림 암호

- 👤 무선환경이나 스트리밍 서비스 등과 같은 환경
- 👤 현재는 블록암호만큼 일반적으로 사용되지 않음.

예) A5/1

- GSM 휴대폰 체계에 사용
- 블루투스 : E0 암호
- RC4
 - 변경되는 256 byte의 Lookup table을 기반
 - 다양한 분야에서 사용
 - 전송 계층 보안(TLS/SSL)이나 무선랜 표준 WEP(Wired Equivalent Privacy) 등의 여러 프로토콜에서 사용되었던 암호 방식

Section 02 스트림 암호

A5/1

- 3개의 shift register로 구성
- Shift-Right 연산 진행 여부 판단
 - X : 19 bits ($x_0, x_1, x_2, \dots, x_{18}$)
 - Y : 22 bits ($y_0, y_1, y_2, \dots, y_{21}$)
 - Z : 23 bits ($z_0, z_1, z_2, \dots, z_{22}$)

X

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Y

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	y_{21}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Z

z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}	z_{16}	z_{17}	z_{18}	z_{19}	z_{20}	z_{21}	z_{22}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Section 02 스트림 암호

- 각 단계에서 $m = \text{maj}(x_8, y_{10}, z_{10})$
 - $\text{maj}()$: x, y, z 중에서 가장 많은 수를 구한다.
 - 만약 기준과 다른 값 : 연산 진행 X,
 - 기준과 같은 값 : 연산 진행
- 예) $\text{maj}(0, 1, 0) = 0$, $\text{maj}(1, 1, 0) = 1$

X

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Y

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	y_{21}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Z

z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}	z_{16}	z_{17}	z_{18}	z_{19}	z_{20}	z_{21}	z_{22}
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

Section 02 스트림 암호

자세한 동작 원리

- 앞의 계산 값을 저장한 후에 Shift-Right 연산을 진행하고, 저장했던 값을 시프트 연산과정에서 비어있게 되는 0번째 인덱스에 넣는다.
- 시프트 연산이 완료되었다면 각 레지스터의 마지막 인덱스인 $X[18]$, $Y[21]$, $Z[22]$ 의 값을 XOR 연산한 값이 키 스트림의 한 비트가 된다.
- 이 과정을 반복할 때마다 키 스트림의 비트수가 하나씩 증가한다. 키 스트림을 평문과 XOR 연산을 하게 되면 암호문이 나온다.

👤 각 단계에서 : $m = \text{maj}(x_8, y_{10}, z_{10})$

■ 예) $\text{maj}(0, 1, 0) = 0$, $\text{maj}(1, 1, 0) = 1$

■ If $x_8 = m$ then X steps

- $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
- $x_i = x_{i-1}$ for $i = 18, 17, \dots, 1$, $x_0 = t$

■ If $y_{10} = m$ then Y steps

- $t = y_{20} \oplus y_{21}$
- $y_i = y_{i-1}$ for $i = 21, 20, \dots, 1$, $y_0 = t$

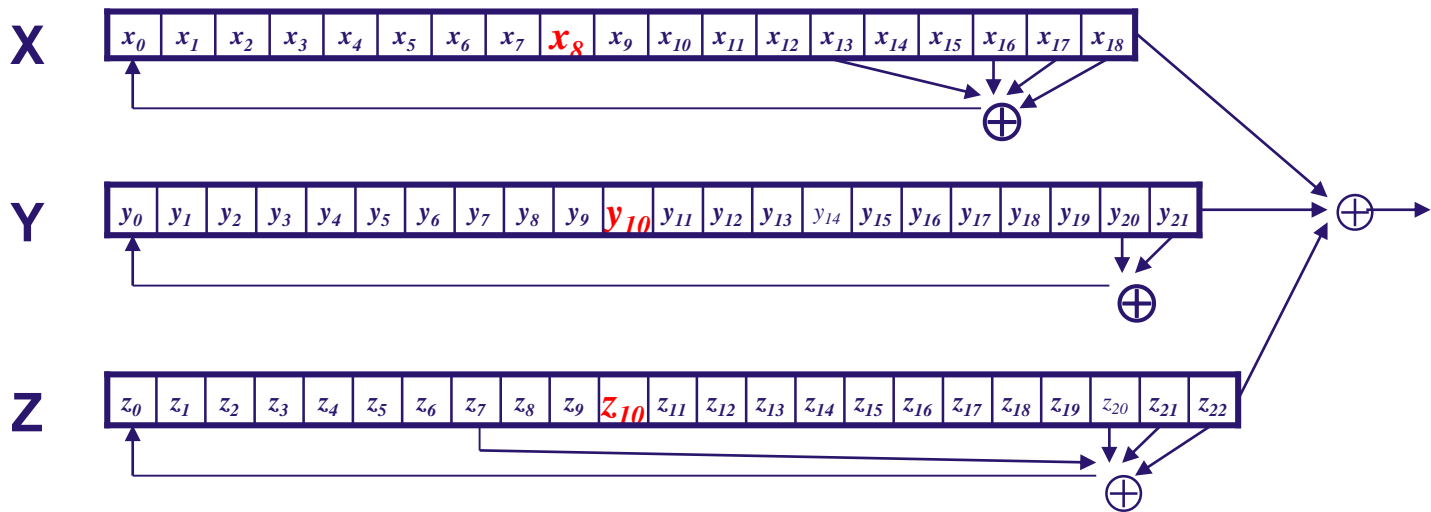
■ If $z_{10} = m$ then Z steps

- $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
- $z_i = z_{i-1}$ for $i = 22, 21, \dots, 1$, $z_0 = t$



Section 02 스트림 암호

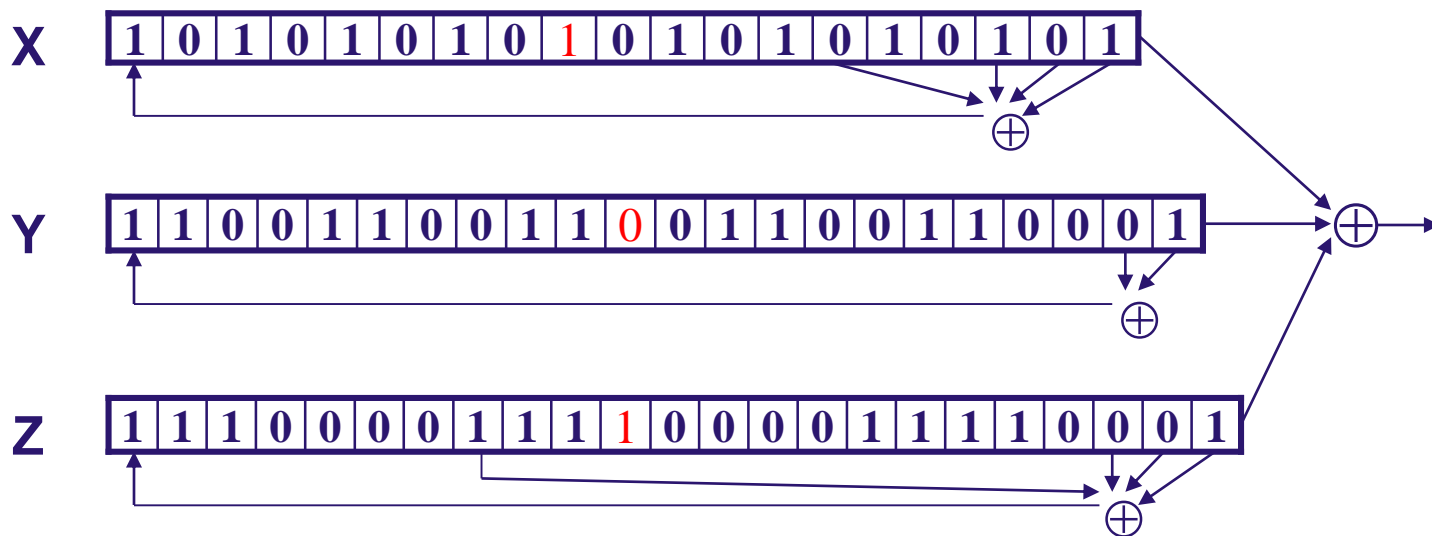
- 각 값은 하나의 비트
- 키는 레지스터들의 초기 값으로 사용
- 각 레지스터는 (x_8, y_{10}, z_{10}) 값에 따라 단계를 진행하거나 혹은 하지 않거나 함
- 키스트림 비트는 레지스터의 우측 비트의 XOR
 - 키스트림 비트는 $x_{18} \oplus y_{21} \oplus z_{22}$



Section 02 스트림 암호

■ 예) $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(1, 0, 1) = 1$

- 레지스터 X는 단계 진행, Y는 하지 않고, Z는 단계 진행
- 키스트림 비트 : $x_{18} \oplus y_{21} \oplus z_{22}$
 $= 0 \oplus 1 \oplus 0 = 1$



Section 02 스트림 암호

Shift register 암호

- shift register 기반 암호는 하드웨어에 효율적
 - 소프트웨어로 구축하기는 어려움
- 과거에는 매우 인기
- 현재는 프로세서들이 매우 빠르기 때문에 s/w로 구축되는 경향
- 일부 분야에서는 shift register가 아직 사용되고 있음.

Section 03 블록 암호

Block Cipher

- 평문과 암호문이 고정된 크기의 블록으로 구성
- 암호문은 평문의 반복되는 회전(round) 함수로 생산
- 회전 함수 입력은 전번 회전 출력과 키로 구성
- 통상적으로 S/W로 구축

Feistel 암호

👤 페이스텔(Feistel) 암호는

- 특정한 암호를 지칭하는 것이 아니고 블록 암호 설계의 한 형태

👤 평문을 좌우 반쪽으로 나눔

- 평문 = (L_0, R_0)

👤 각 회전($i=1, 2, \dots, n$)에서의 계산

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

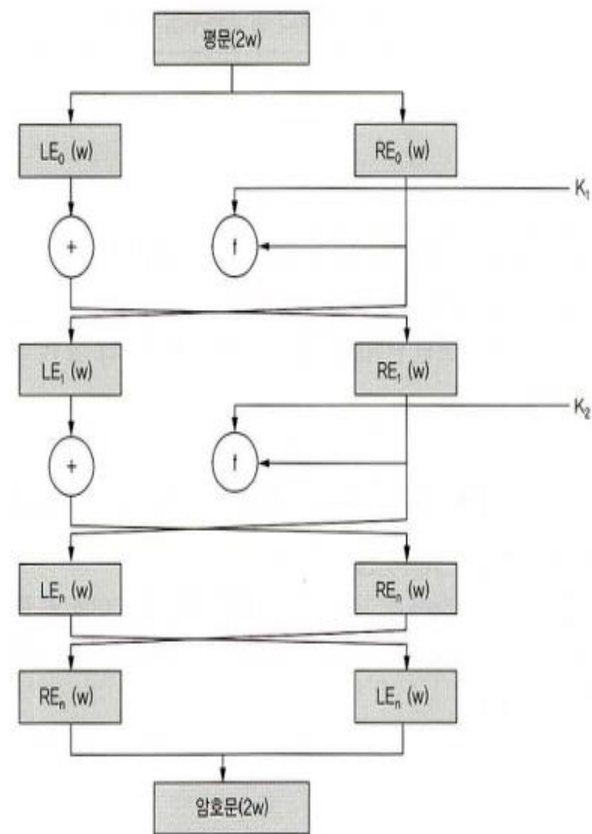
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

- F : 회전 함수, K_i : 보조키

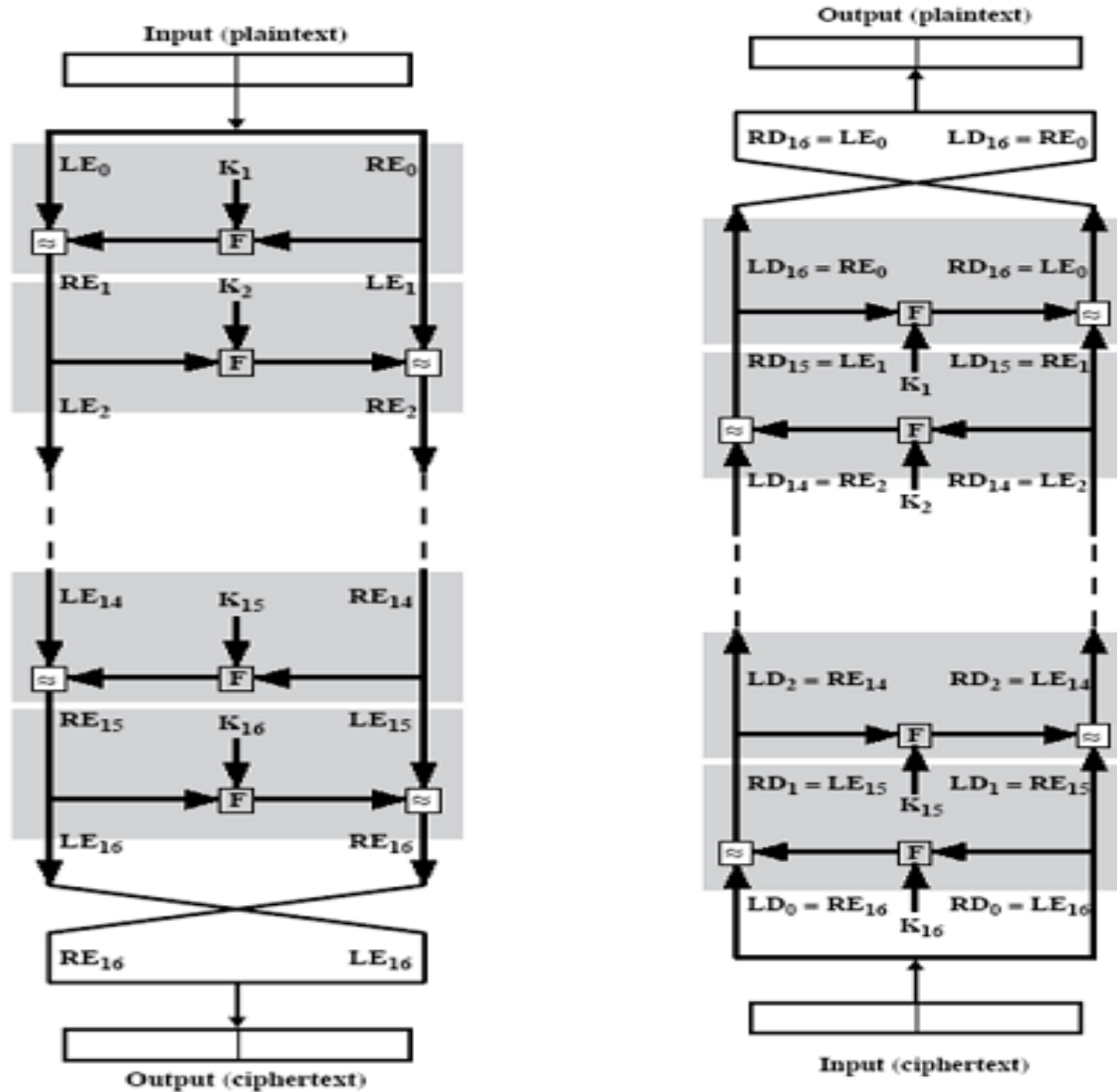
👤 암호문 = (L_n, R_n)

👤 공식은 어떤 함수 F 에 대해서도 성립

👤 그러나 특정 함수 F 에 대해서만 안전



Feistel 암호



Feistel 암호

👤 복호화 : 암호문 = (L_n, R_n)

👤 각 회전 $i=n, n-1, \dots, 1$ 에서 다음을 계산

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

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

- F : 회전 함수, K_i : 보조키

👤 평문 = (L_0, R_0)

Data Encryption Standard

DES

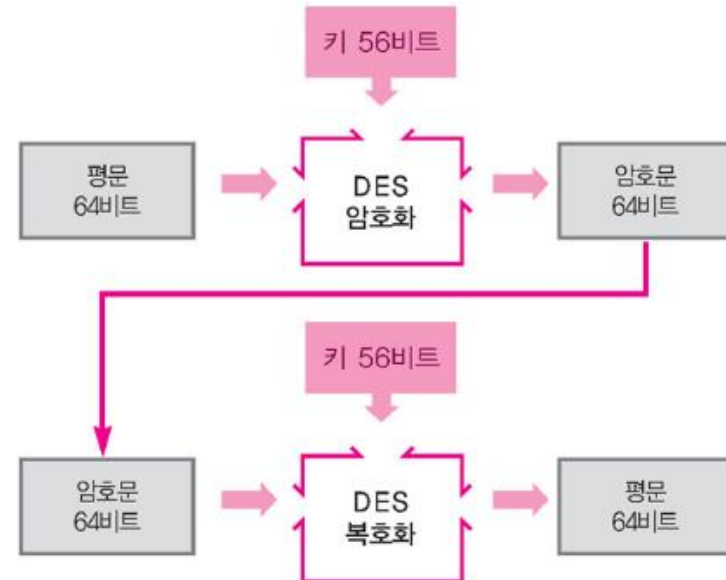
- 1970년대 개발된 블록 암호 알고리즘
- 64비트 평문을 64 비트 암호문으로 암호화하는 방식
- 키 길이 : 56 비트
 - 오류 검출을 위해 8비트 사용, $64=56+8(\text{parity})$
- 1949년도에 Claude Shannon이 제시한 혼돈과 치환이라는 두 가지 개념에 기반
- IBM Lucifer 암호를 기반
- 미 정부 표준
- DES 개발은 논쟁이 있었음
 - NSA가 비밀리에 관련되었음
 - 설계 과정이 비공개
 - 키 길이가 줄었음
 - Lucifer 알고리즘의 교묘한 변경

Data Encryption Standard

👤 DES는 Feistel 암호

- 64 비트 블록 길이
- 56 비트 키 길이
- 16 회전
- 각 회전에서 48 비트의 보조키 사용

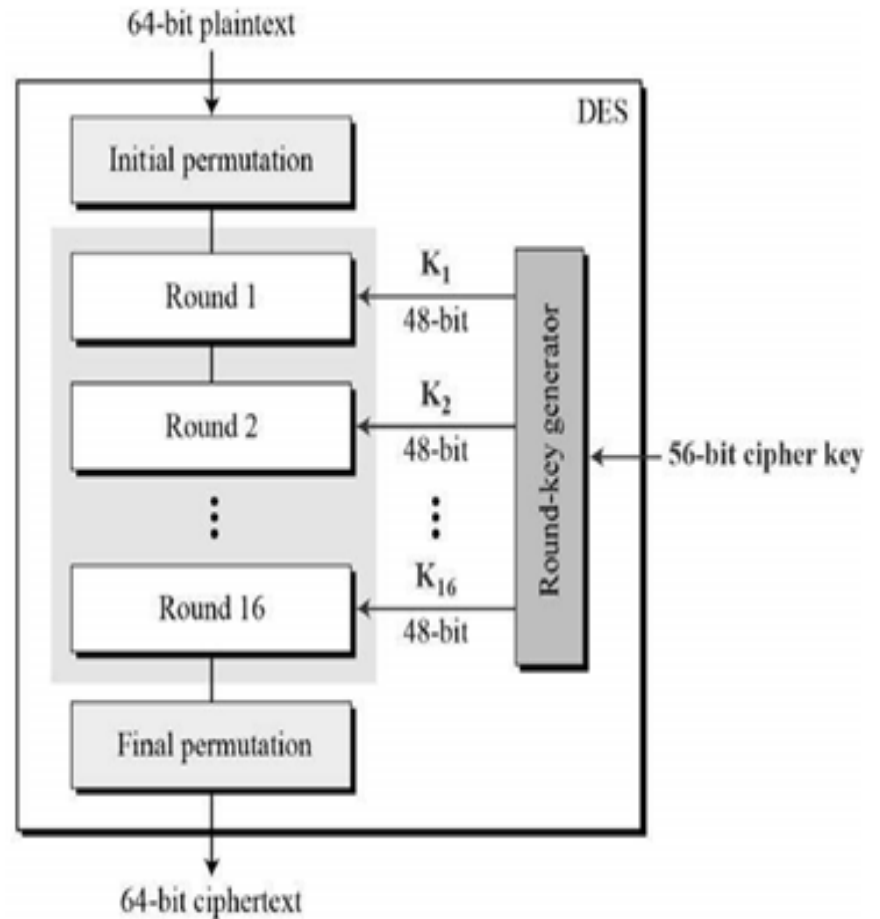
👤 각 회전은 단순



Data Encryption Standard

👤 DES는 Feistel 암호

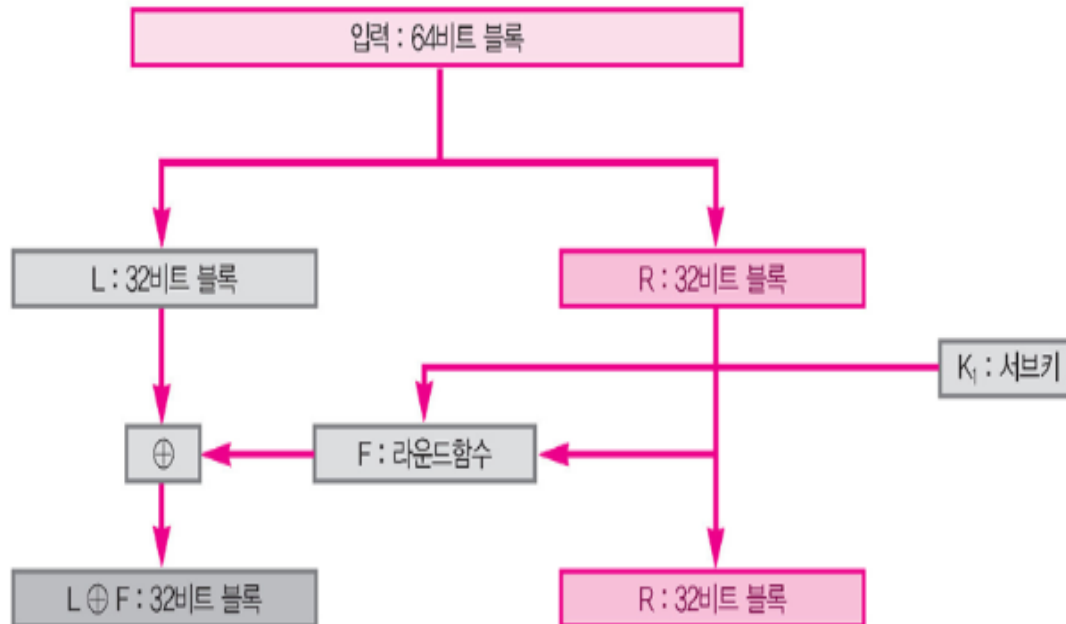
- 16 회전(round)
 - 라운드를 16번 반복
 - Round :
 - 암호화 과정의 한 단계
- 각 회전에서 48 bit의 보조키 사용



Data Encryption Standard

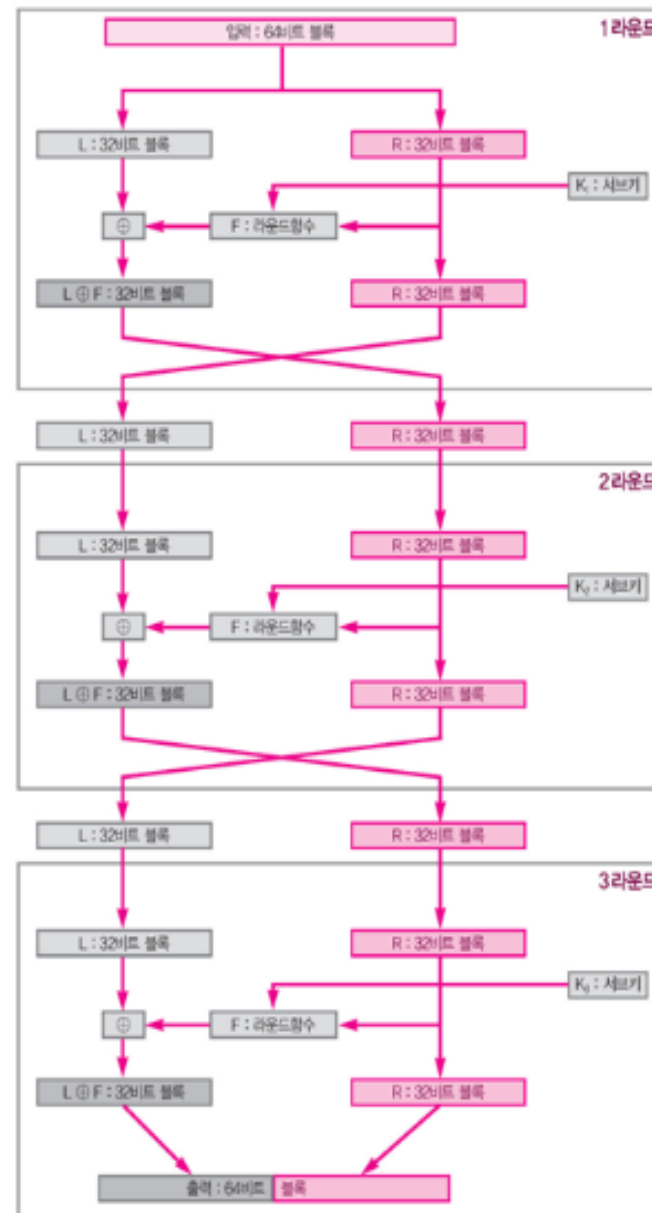
👤 각 회전(round)은 단순

- 64비트 블록을 절반으로 나눠서 왼쪽(L)과 오른쪽(R)로 나눔



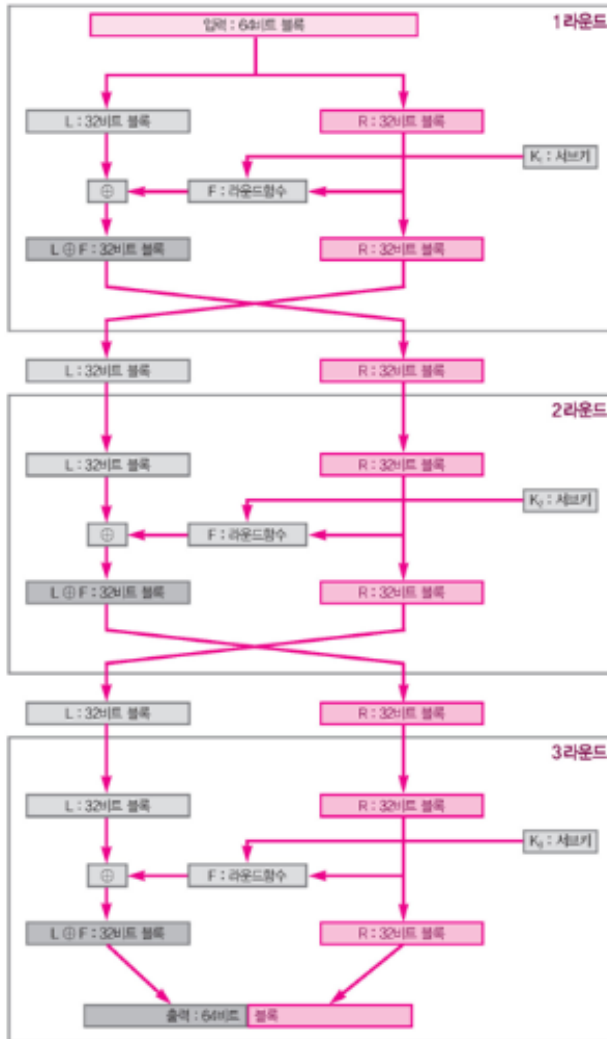
[그림] DES의 제 1라운드 과정

- Feistel 네트워크 암호화(3 라운드)
 - 동일한 하나의 라운드가 3번 반복한다

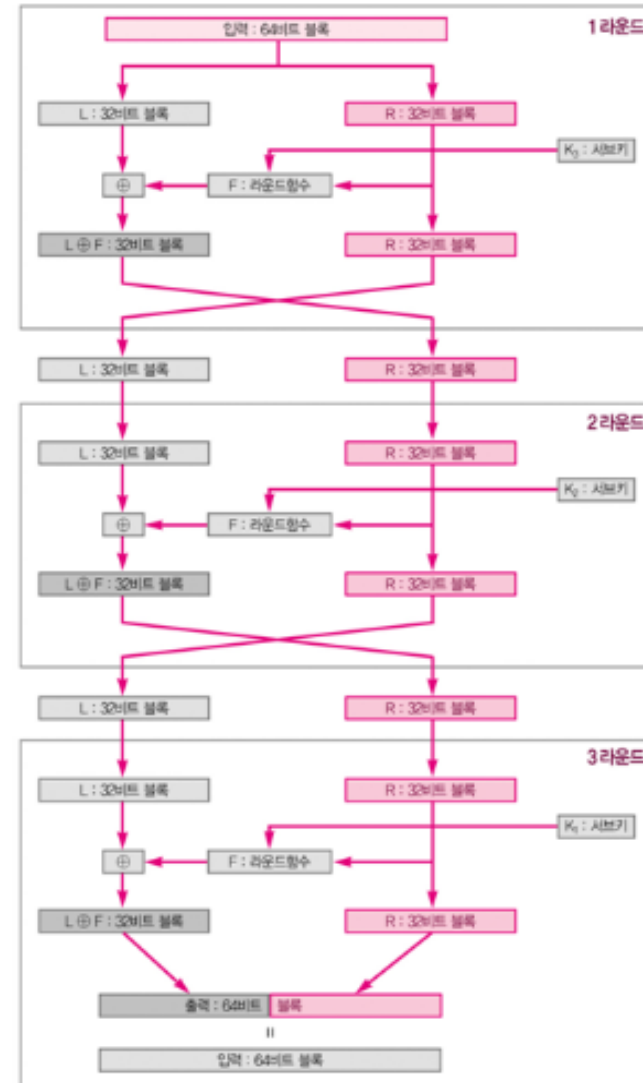


[그림] DES의 Feistel network

👤 Feistel 네트워크의 복호화 (3라운드)



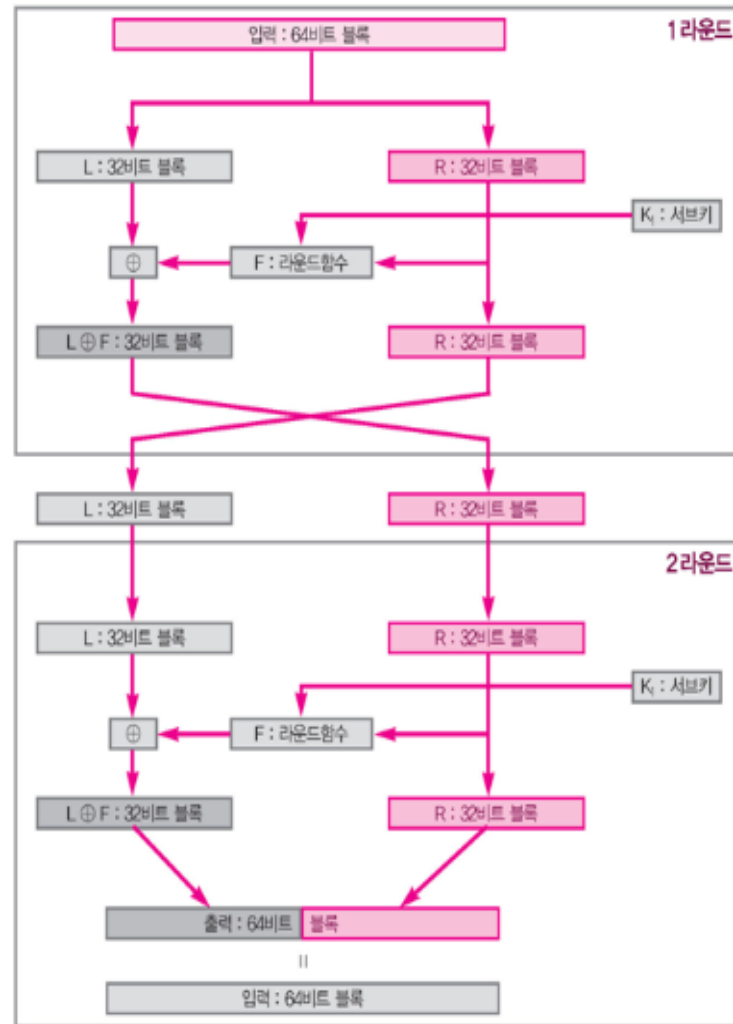
[그림] DES의 암호화



[그림] DES의 복호화

👤 같은 서브 키로 Feistel 네트워크를 2회 통과하면?

- 원래로 돌아간다.

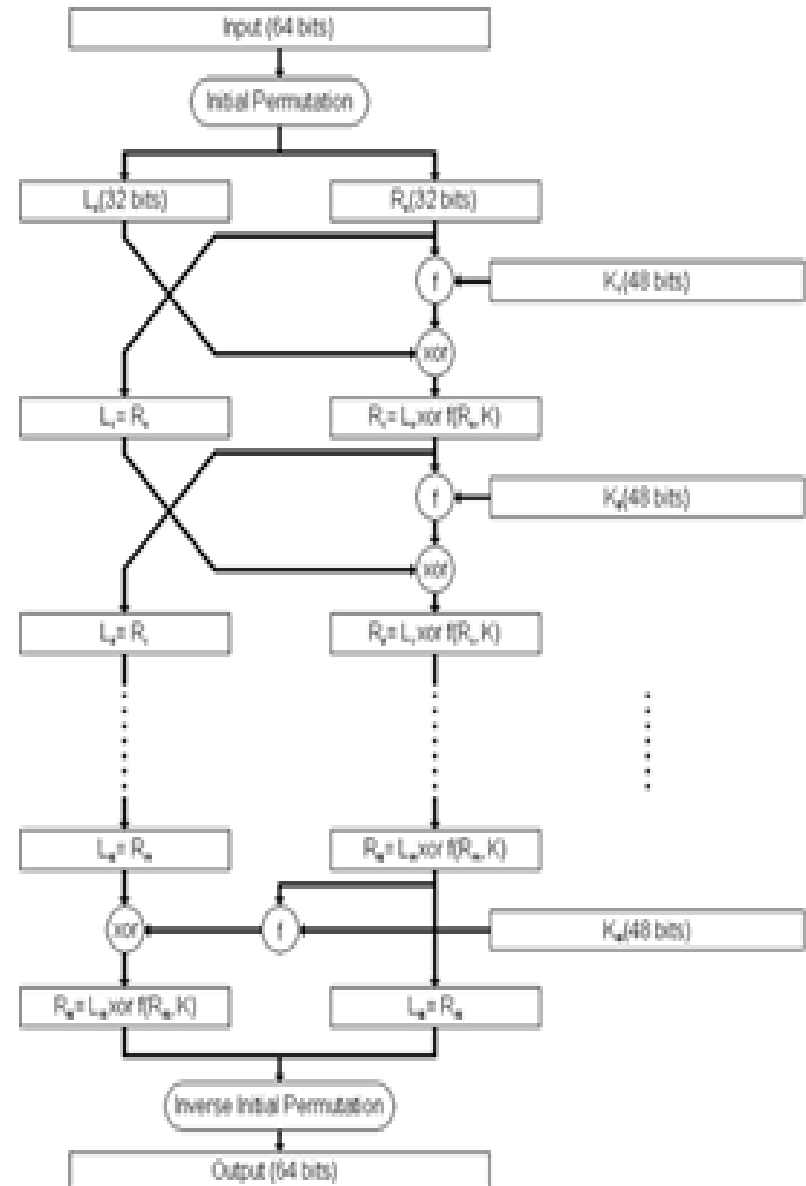


[그림] Feistel network

Data Encryption Standard

DES의 전체 과정(16 라운드)

- 최초 회전 전의 최초 순열 P
- 마지막 회전 후에 반씩을 교환
- 마지막 순열(P의 역)을 (R_{16}, L_{16})에 적용함으로 암호문 생산



Data Encryption Standard

Feistel 네트워크의 성질

- 원하는 만큼 라운드 수를 늘릴 수 있다.
- 라운드 함수 F 에 어떤 함수를 사용해도 복호화가 가능하다
- 암호화와 복호화를 완전히 동일한 구조로 실현 가능

쇄도 효과(avalanche effect)

- 평문 또는 키 값을 조금만 변경시켜도 암호문에는 매우 큰 변화가 생기는 효과

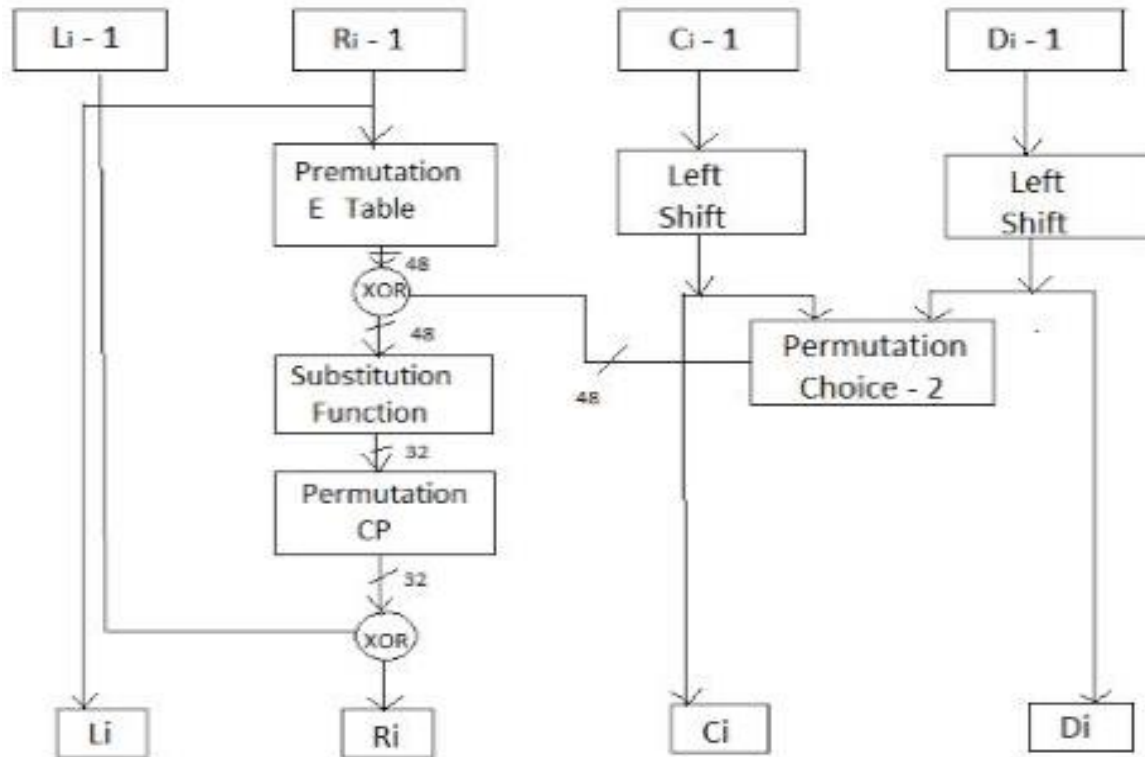
👉 평문과 키 생성 부분을 모두 표현한 것



DES의 한 라운드

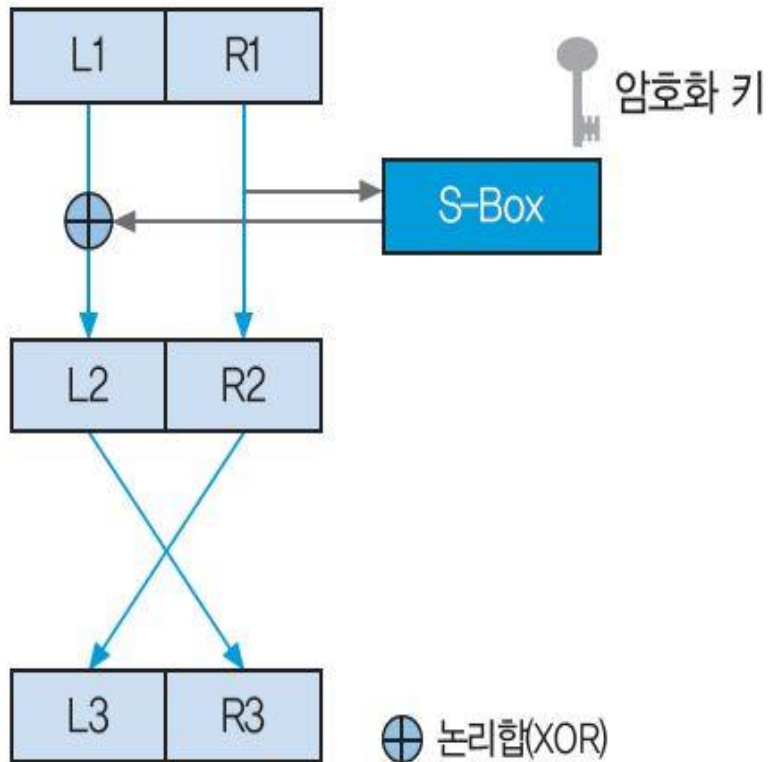
[평문 처리 과정]

[키를 처리하는 과정]

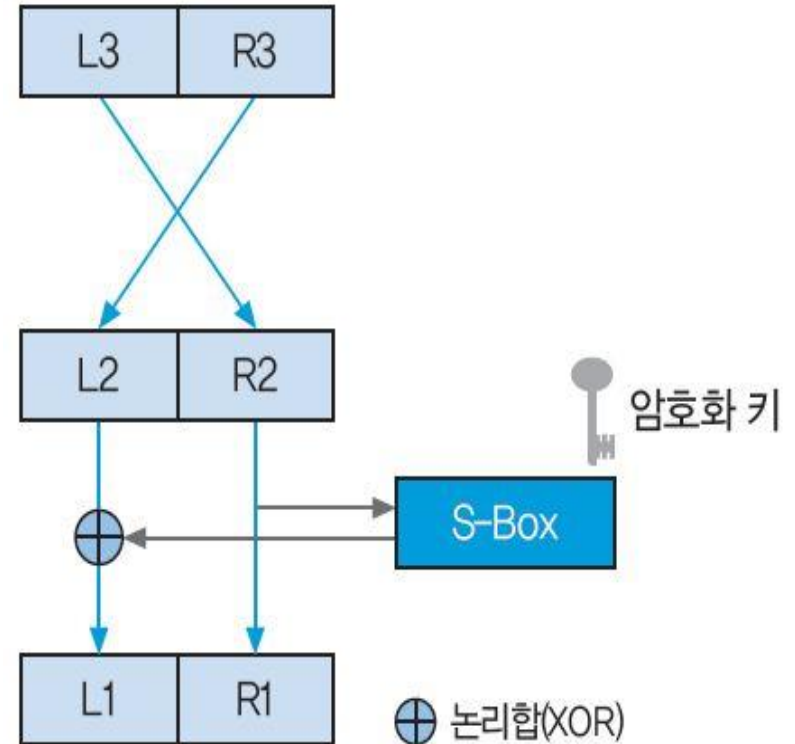


[그림] DES 암호화 과정

DES의 한 라운드



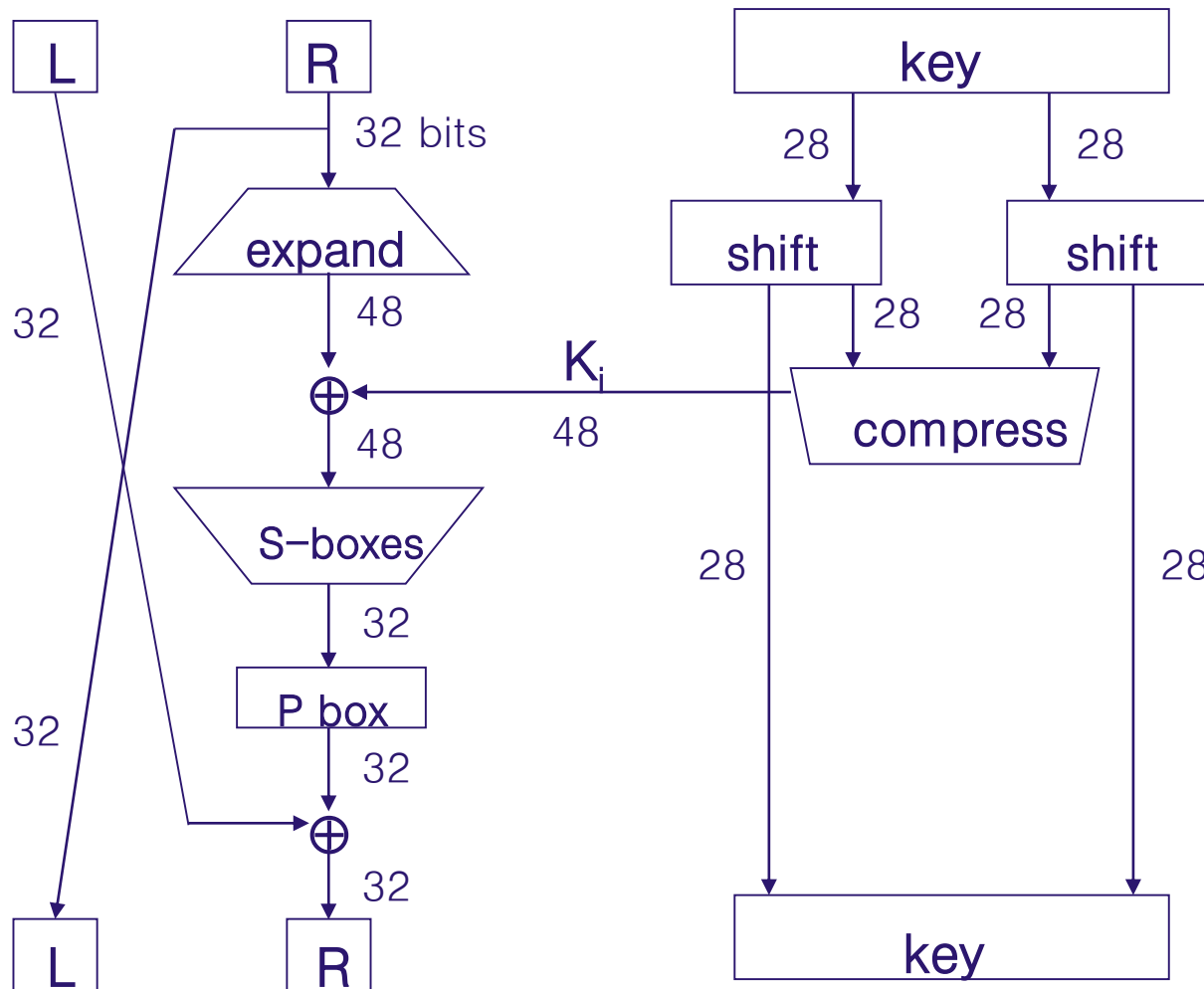
[그림] DES 암호화 과정



[그림] DES 복호화 과정

DES의 한 라운드

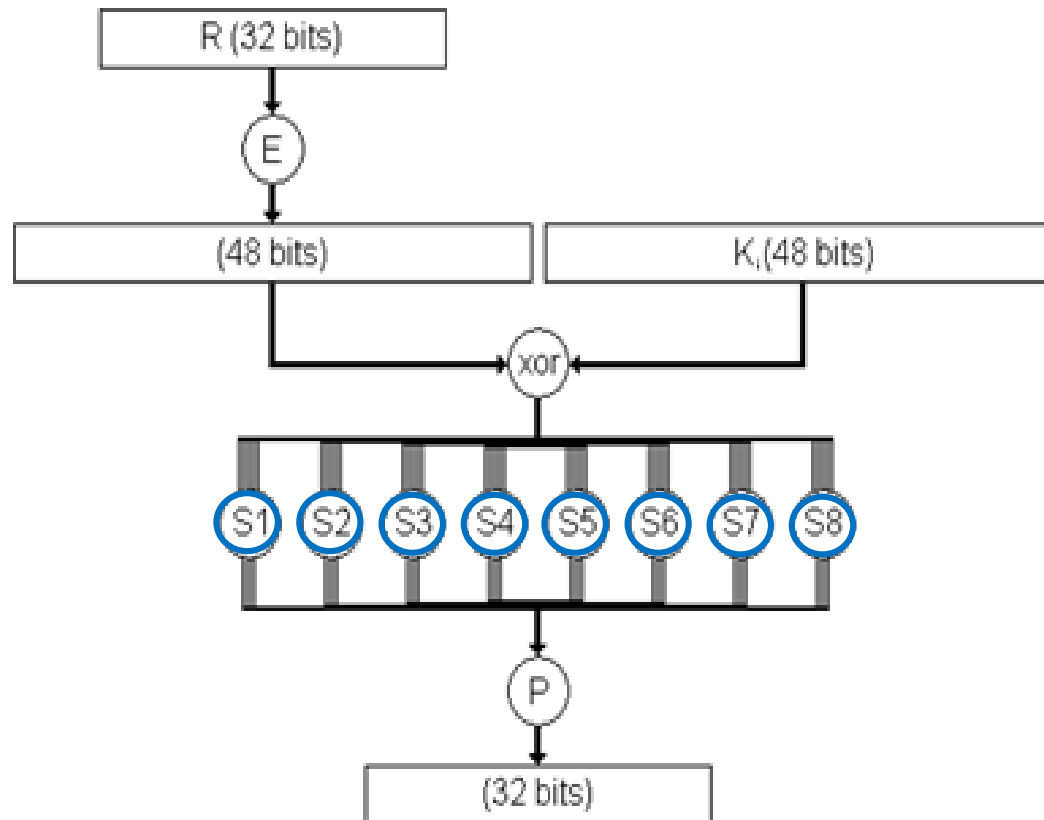
한 라운드의 상세 동작



DES

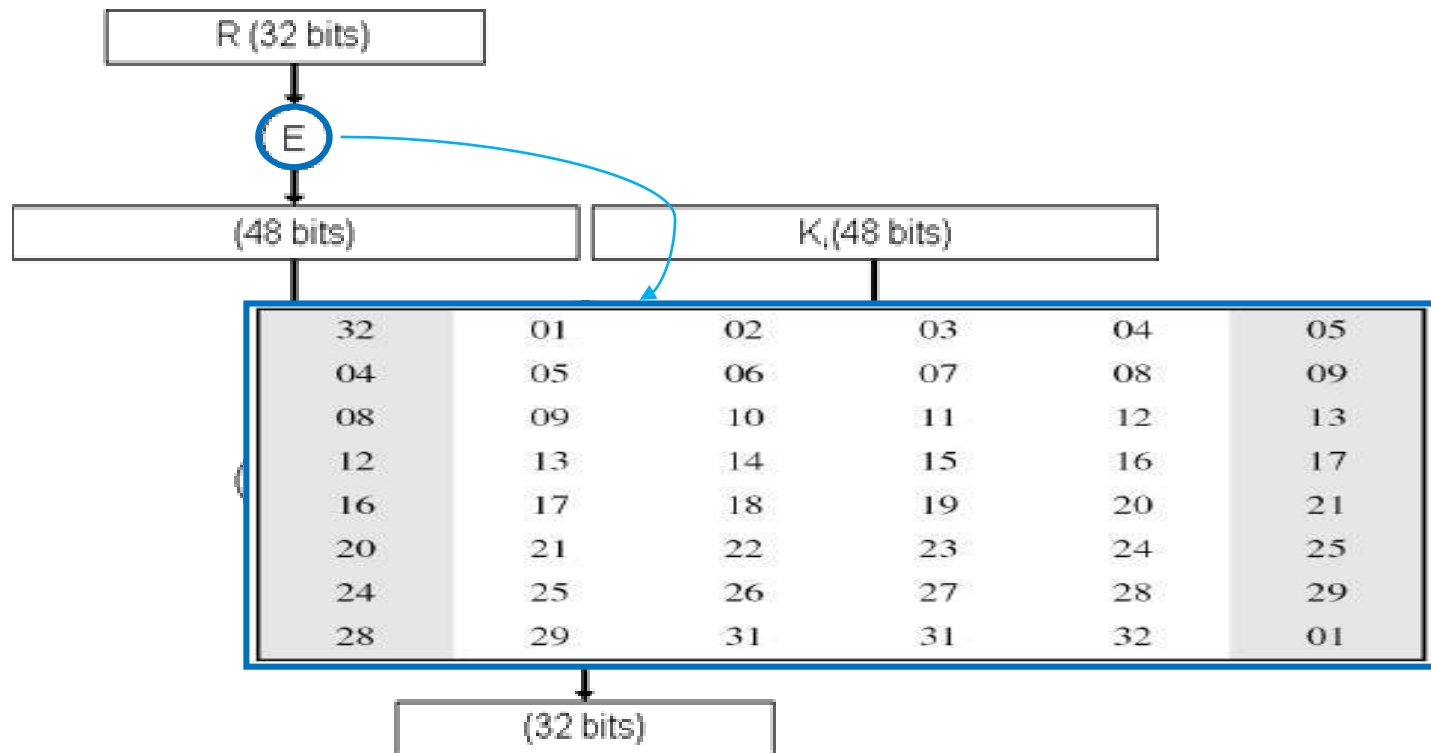
👤 안전성은 주로 “S-box들”에 의존

- 각 S-box : 입력은 6 bit -> 출력은 4 bit로 매핑



Expansion Permutation

- 32 bit 입력 -> 48(6 X 8) bit 출력
- 표를 해석하는 방법 : 제일 앞의 '32'는 입력의 32번째에 있는 비트 값을 표의 '32'가 있는 위치, 즉 제일 처음 부분으로 가져오라는 의미



👤 DES에 나오는 모든 표의 해석 방법

■ 예) 앞의 Expansion Permutation 그림을 예로 들어보자.

- 제일 앞의 '32'는 입력의 32번째에 있는 비트값을 아래 표의 '32'가 있는 위치, 즉 제일 처음 부분으로 가져오라는 의미
- 두 번째 '01'은 입력의 첫 번째 비트에 있는 값을 아래 표의 '01' 위치로 가져오라는 의미

1	2	3	4	5	6	7	8	9	10	...	29	30	31	32					
0	1	1	0	1	1	1	0	0	1	...	1	1	1	0	0	1	0	1	0

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	31	31	32	01

S-box(Substitution Function)

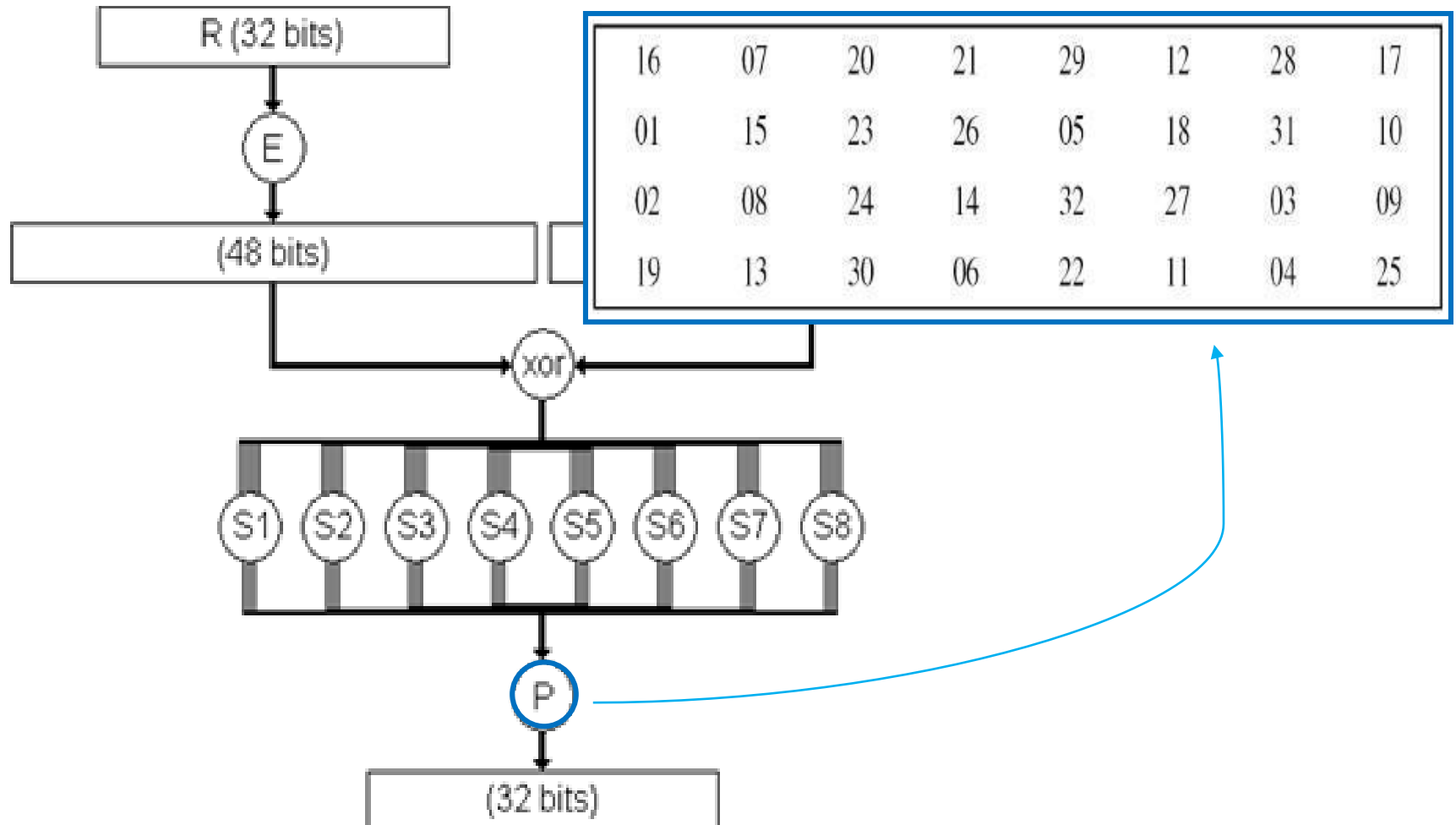
-8개의 S-box가 존재

-각 S-box는 다 다름

שורה	מס' עמודה															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S ₁																
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	3	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	13	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S ₂																
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S ₃																
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S ₄																
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S ₅																
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S ₆																
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S ₇																
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S ₈																
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

P-box

- 앞의 그림들에서 permutation이라고 표시된 부분임



DES의 Subkey

pc-1 (Permuted choice 1)						
49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31
55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3

pc-2 (Permuted choice 2)					
13	16	10	23	0	4
2	27	14	5	20	9
22	18	11	3	25	7
15	6	26	19	12	1
40	51	30	36	46	54
29	39	50	44	32	47
43	48	38	55	33	52
45	41	49	35	28	31

👤 시프트 연산의 경우 라운드마다 시프트값이 다르다.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Bits Shifted in each round

Data Encryption Standard

👤 DES의 확장 순열은?

■ Expansion Permutation

👤 입력 32 비트

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

👤 출력 48 비트

31 0 1 2 3 4 3 4 5 6 7 8
7 8 9 10 11 12 11 12 13 14 15 16
15 16 17 18 19 20 19 20 21 22 23 24
23 24 25 26 27 28 27 28 29 30 31 0

DES S-box

👤 S-box 또는 8 “교환 박스”

👤 각 S-박스 : 입력 6 비트를 출력 4 비트로 매핑

👤 각 박스의 행과 열의 결정방법

- 행 : 입력 6비트에서 처음 비트와 마지막 비트를 결합
- 열 : 가운데 4비트

예) 110011에서 행은 11로 3, 열은 1001로 9

즉, 3행 9열의 값(1011)을 출력

👤 1번 S-박스

입력 비트(0, 5)

↓

입력 비트 (1, 2, 3, 4)

		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

00		1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01		0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10		0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11		1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

DES P-box

입력 32 비트

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

출력 32 비트

15 6 19 20 28 11 27 16 0 14 22 25 4 17 30 9
1 7 23 13 31 26 2 8 18 12 29 5 21 10 3 24

DES 보조키

👤 56 비트 DES 키 : 0, 1, 2, ..., 55

👤 좌 반쪽 키 비트 : LK

49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31

👤 우 반쪽 키 비트 : RK

55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3

DES 보조키

👤 $i=1, 2, \dots, 16$ 번째 회전에서

- LK = (LK를 r_i 만큼 왼쪽으로 회전 이동)
- RK = (RK를 r_i 만큼 왼쪽으로 회전 이동)
- 보조키 K_i 의 왼쪽 절반이 LK 비트

13	16	10	23	0	4	2	27	14	5	20	9
22	18	11	3	25	7	15	6	26	19	12	1

- 보조키 K_i 의 오른쪽 절반이 RK 비트

12	23	2	8	18	26	1	11	22	16	4	19
15	20	10	27	5	24	17	13	21	7	0	3

DES 보조키

- 👤 회전 1, 2, 9, 16에서 r_i 는 1, 나머지 회전 r_i 는 2
- 👤 각 회전 LK의 비트 8, 17, 21, 24 생략
- 👤 각 회전 RK의 비트 6, 9, 14, 25 생략
- 👤 압축 순열은 56 bits of LK 와 RK의 56 비트에서 48 비트 보조 키 K_i 생산
- 👤 위 키 스케줄은 보조키를 생산

DES의 안전성

👤 DES의 안전성 : 다수의 S-box에 의존

- DES의 나머지는 모두 선형

👤 30년간의 강도 높은 분석으로 “백도어”가 없음을 밝혀 냈음

👤 오늘날 공격은 전수키 조사를 사용

👤 RSA사가 주관한 DES의 DES Challenge 결과

- 1997년의 DES Challenge I에서는 96일

- 1997년의 DES Challenge II-1에서는 41일

- 1999년의 DES Challenge II-2에서는 56시간

- 1999년의 DES Challenge III에서는 22시간 15분만에 DES 키가 깨짐

블록 암호 표기

👤 P = 평문 블록

👤 C = 암호문 블록

👤 암호문 C 를 얻기 위해 키 K 로 P 를 암호화

- $C = E(P, K)$

👤 평문 P 를 얻기 위해 키 K 로 C 를 복호화

- $P = D(C, K)$

👤 아래 사항을 주의

- $P = D(E(P, K), K)$ 그리고 $C = E(D(C, K), K)$

블록 암호 표기

👤 현재, 56 비트 DES 키는 너무 작다

👤 하지만 DES가 도처에서 사용 중 : 어떻게 해야 하나?

👤 **삼중 DES** 또는 **3DES** (112 비트 키)

- DES의 비도를 강화한 대칭키 암호

- $C = E(D(E(P, K_1), K_2), K_1)$

- $P = D(E(D(C, K_1), K_2), K_1)$

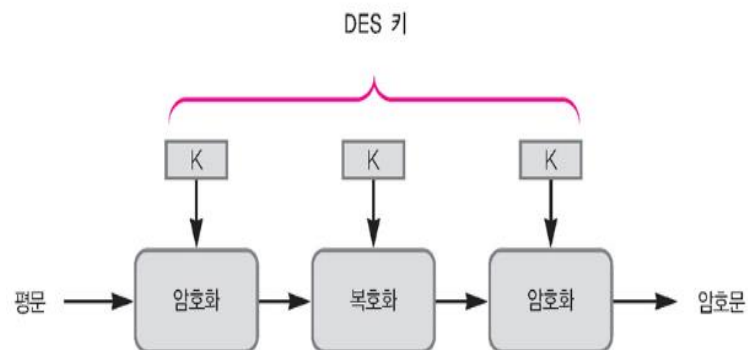
- 왜 2개 키로 암호화-복호화-암호화(EDE) 하는가?

- 단독 DES와 호환성

- 3DES를 DES로 사용가능

- $E(D(E(P, K), K), K) = E(P, K)$

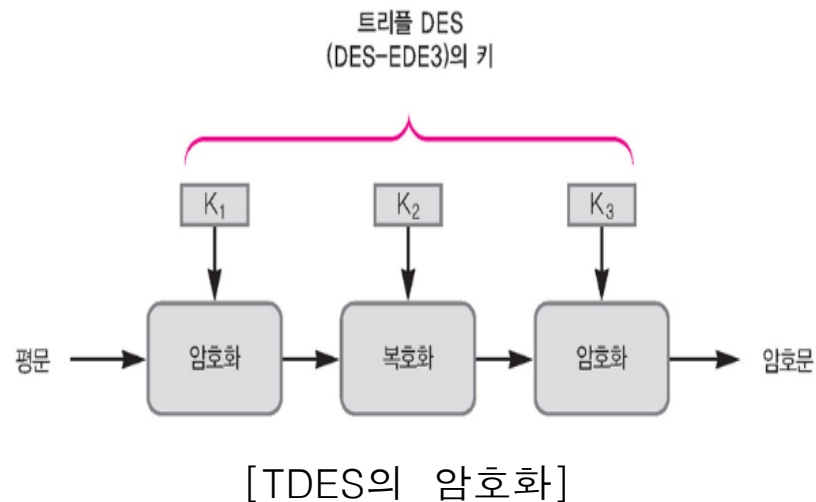
- 112 비트면 안전성을 위해 충분



Triple DES

3DES

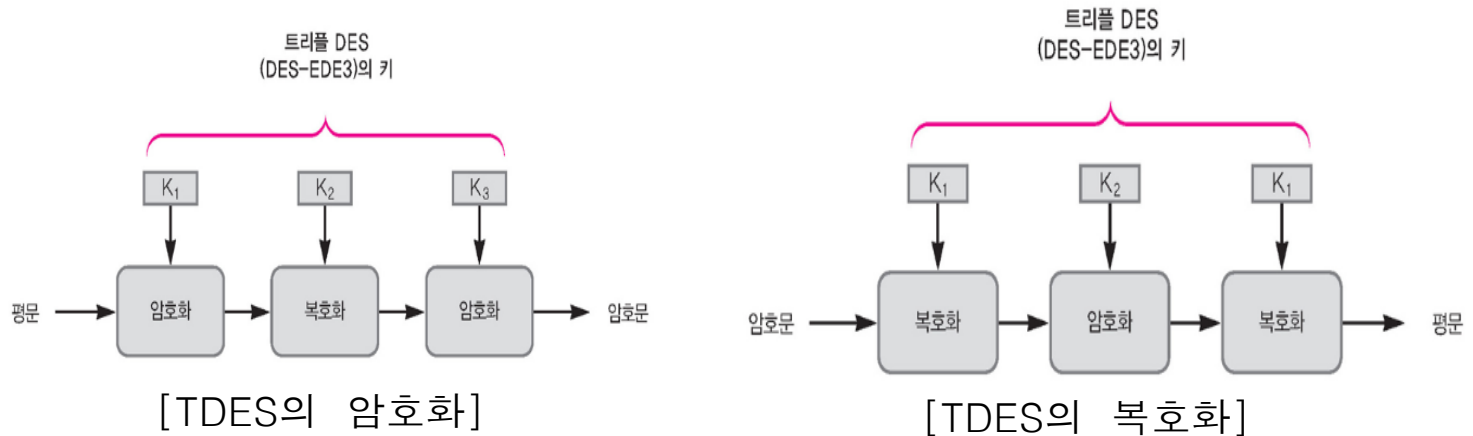
- 금융 분야 응용에 사용할 목적으로 1985년 표준화
- 다른 명칭들 : TDES(triple DES), 3중 DES
- DES보다 강력하도록 DES를 3번 겹치게 한 알고리즘
 - TDES를 단독 DES로도 사용가능하게 하려고 고안한 방법
 - 예) 키가 모두 같을 경우 : DES 암호화를 1번 수행한 것과 같은 효과



Triple DES

■ TDES의 복호화

- 암호화 : 암호화(Encryption) -> 복호화(Decryption) -> 암호화(Encryption)
- 복호화 : D -> E -> D
 - » 키의 순서 : $k_3 \rightarrow k_2 \rightarrow k_1$



■ 은행 등에서 사용, 활용

- 금융권에서 전자지불시스템 등에서 DES-EDE2는 아직 상당히 많이 사용되고 있다

■ AES가 널리 사용될때까지 임시적으로 사용

AES의 선정 과정

👤 DES를 대신하는 새로운 표준

👤 Advanced Encryption Standard 경쟁 (90년대 후반)

- AES를 공모한 것은 미국의 표준화기구인 NIST(National Institute of Standard and Technology)
- 투명한 진행
- 전 세계의 기업과 암호 학자가 AES의 후보로서 다수의 대칭 암호 알고리즘들을 제안
- 2000년에 Rijndael 알고리즘이 선정
 - “Rain Doll” 또는 “Rhine Doll”로 발음

👤 반복되는 블록 암호(DES와 동일)

👤 페이스텔 암호가 아님(DES와 상이)

👤 입력 평문의 길이 : 128 비트, 키 길이 : 128 비트, 196 비트, 258 비트

AES 최종 후보 및 선정

👤 1997년 1월 2일 NIST는 AES의 모집을 개시

👤 조건

- 속도가 빠를 것, 단순하고 구현하기 쉬울 것
- 암호화 자체의 속도뿐만 아니라 키의 셋업 속도도 중요
- 스마트카드나 8비트 CPU 등의 계산력이 작은 플랫폼에서부터 워크스테이션과 같은 고성능의 플랫폼에 이르기까지 빠르게 동작
- 블록 길이 : 128 비트인 대칭 블록 암호
- 다양한 키의 길이 : 128, 192, 256 비트를 지원

👤 공모 조건

- 안전성(security)
 - 대칭키 암호를 해독하려는 공격으로부터 안전해야 한다
 - DES를 해독한 대표적인 공격 방법인 선형 공격(linear cryptanalysis)과 차분 공격(differential cryptanalysis)
- 비용(cost) : 다양한 플랫폼에서 빠른 동작
- 구현 효율성(implementation) : 구현이 단순할 것

AES 최종 후보 및 선정

최초 평가 대상(15개)

■ 12개국에서 모두 15개의 알고리즘이 제안

- CAST256,
- Crypton,
- DEAL,
- DFC,
- E2,
- Frog,
- HPC,
- LOKI97,
- Magenta,
- MARS,
- RC6,
- Rijndael,
- SAFER+,
- Serpent,
- Twofish

AES의 최종 후보

명칭	응모자
MARS	IBM사
RC6	RSA사
Rijndael	Daemen, Rijmen
Serpent	Anderson, Biham, Knudsen
Twofish	Counterpane사

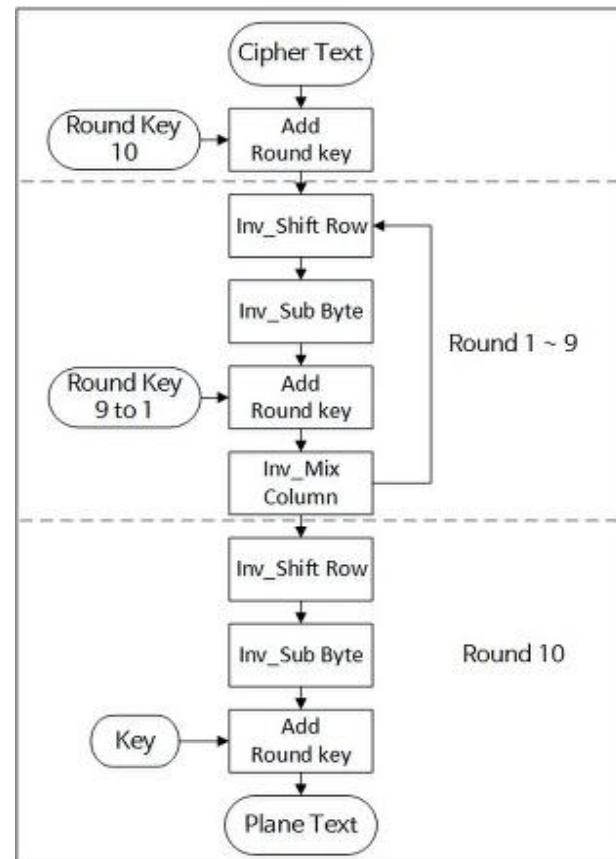
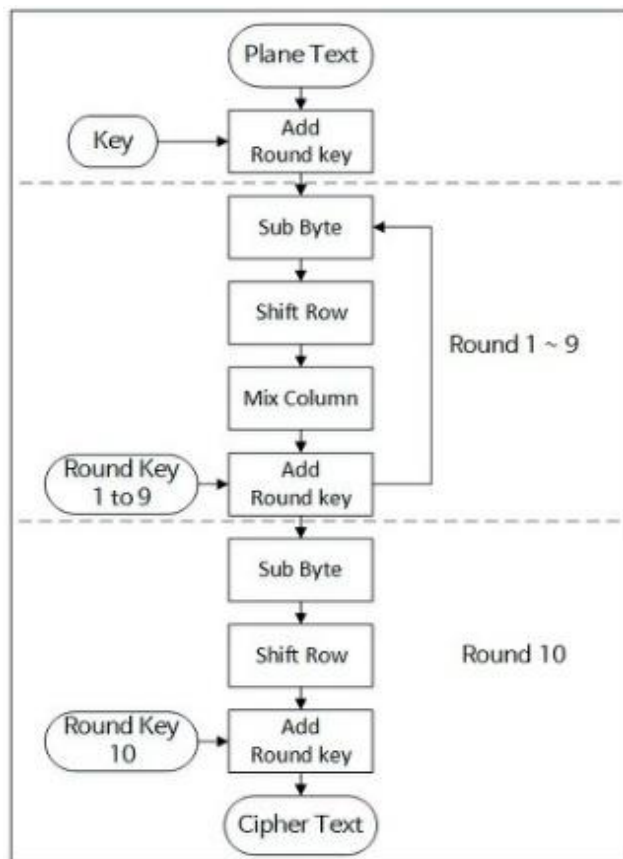
Advanced Encryption Standard

Rijndael

- 2000년 10월에 벨기에의 Rijndael 알고리즘이 AES로 채택
 - Joan Daemen, Vincent Rijmen
- Rijndael의 블록 길이 : 128 비트
- 키의 길이
 - 128비트부터 256비트까지 32비트 단위로 선택가능
- 종류
 - AES-128 : 블록 크기가 128비트로 10번의 라운드 반복
 - AES-192 : 블록 크기가 192비트로 12번의 라운드 반복
 - AES-256 : 블록 크기가 256비트로 14번의 라운드 반복

Advanced Encryption Standard

🔧 AES의 구조 및 알고리즘



[암호화]

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

SubBytes

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

S

[복호화]

Advanced Encryption Standard

AES의 S-box

입력 뒤의 4 비트

입력
앞의 4비트

	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

모드(mode)

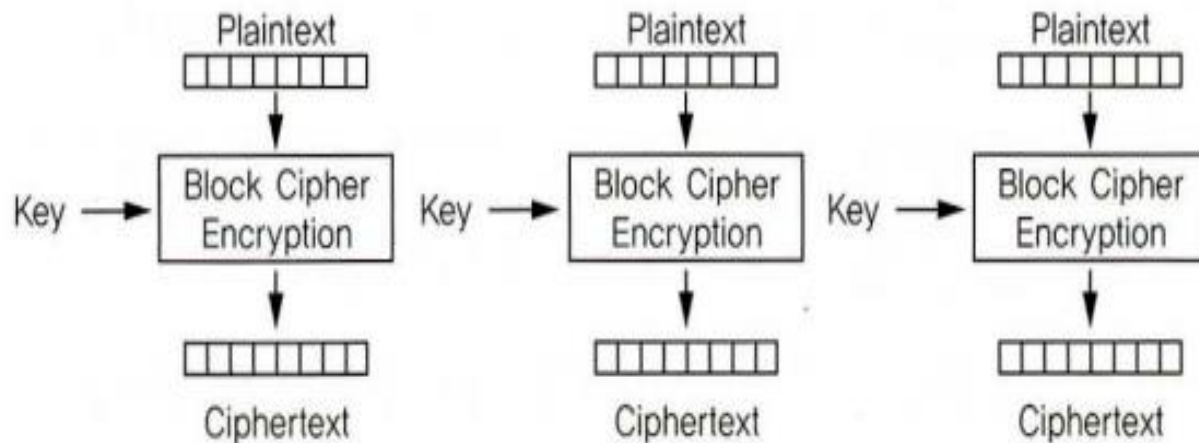
- 블록 암호 알고리즘을 반복 사용하여 평문 전체를 암호화
- 이와 같이 반복하는 방법

주요 모드

- ECB(Electronic Code Book, 전자 부호표) 모드
- CBC(Cipher Block Chaining, 암호 블록 연쇄) 모드
- CFB(Cipher FeedBack, 암호 피드백) 모드
- OFB(Output FeedBack, 출력 피드백) 모드
- CTR(CounTeR, 카운터) 모드

👤 Electronic Code Book (**ECB**) Mode

- 전자 코드북 모드
 - 가장 간단한 모드
 - 각 블록을 독립적으로 암호화
 - 평문 블록을 암호화한 것이 그대로 암호문 블록
 - 평문 블록과 암호문 블록이 일대일 대응표를 갖는다
 - 동일한 내용을 갖는 평문 블록은 이에 대응되는 동일한 암호문 블록으로 변환
- 전자 코드북 모드



■ 패딩(padding)

- 마지막 평문 블록이 블록 길이에 미치지 못할 경우에 추가하여 블록 길이가 되도록 맞춘다

■ ECB 모드의 특징

- 가장 기밀성이 낮은 모드
- 암호문을 살펴보는 것만으로도 평문 속에 패턴 반복성 감지
- 안전하지 않다

 예)

✓ 원래의 지시 :

A-5374의 계좌에서 B-6671의 계좌로 1억 원을 송금하라

✓ 변경된 지시 :

B-6671의 계좌에서 A-5374의 계좌로 1억 원을 송금하라는
정반대의 지시가 됨



- 어느 은행의 송금 의뢰 데이터가 다음 3개의 블록으로 구성
 - 블록1 = 송금자의 은행계좌번호
 - 블록2 = 송금처의 은행계좌번호
 - 블록3 = 송금액
- 송금 의뢰 데이터를 받은 은행은 지정된 금액을 송금자로부터 송금처의 계좌로 이동
- A-5374의 계좌로부터 B-6671의 계좌로 1억 원을 송금하라는 송금 의뢰 데이터를 만들어보자

예) cont'd

- 평문 블록1 = 41 2D 35 33 37 34 20 20 20 20 20 20 20 20 20(송금자 : A-5374)
- 평문 블록2 = 42 2D 36 36 37 31 20 20 20 20 20 20 20 20 20(송금처 : B-6671)
- 평문 블록3 = 31 30 30 30 30 30 30 30 30 20 20 20 20 20 20(송금액 :
1000000000)

- 암호화 하자
 - 암호문 블록1 = 59 7D DE CC EF EC BA 9B BF 83 99 CF 60 D2 59 B9
(송금자:????)
 - 암호문 블록2 = DF 49 2A 1C 14 8E 18 B6 53 1F 38 BD 5A A9 D7 D7
(송금처:????)
 - 암호문 블록3 = CD AF D5 9E 39 FE FD 6D 64 8B CC CB 52 56 8D 79
(송금액:????)

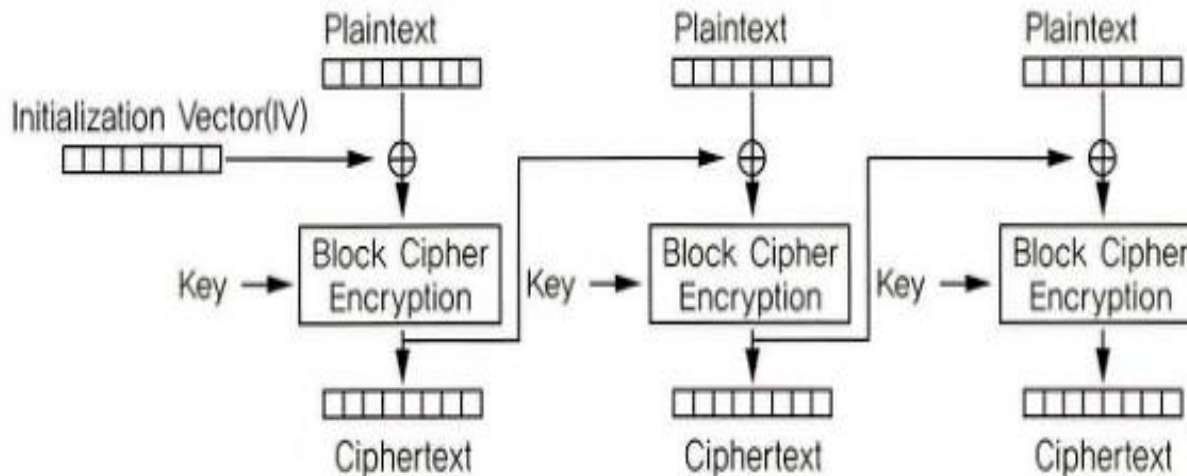
예) cont'd

- 공격자 맬로리가 암호문 블록의 1과 2의 내용을 바꾼다
 - 암호문 블록1 = DF 49 2A 1C 14 8E 18 B6 53 1F 38 BD 5A A9 D7 D7
(송금자 : ????)
 - 암호문 블록2 = 59 7D DE CC EF EC BA 9B BF 83 99 CF 60 D2 59 B9
(송금처 : ????)
 - 암호문 블록3 = CD AF D5 9E 39 FE FD 6D 64 8B CC CB 52 56 8D 79
(송금액 : ????)

- 은행이 이것을 복호화하면 다음과 같이 된다
 - 평문 블록1 = 42 2D 36 36 37 31 20 20 20 20 20 20 20 20 20
(송금처 : B-6671)
 - 평문 블록2 = 41 2D 35 33 37 34 20 20 20 20 20 20 20 20 20
(송금자 : A-5374)
 - 평문 블록3 = 31 30 30 30 30 30 30 30 30 20 20 20 20 20 20
(송금액 : 1000000000)

👤 Cipher Block Chaining Mode(CBC)

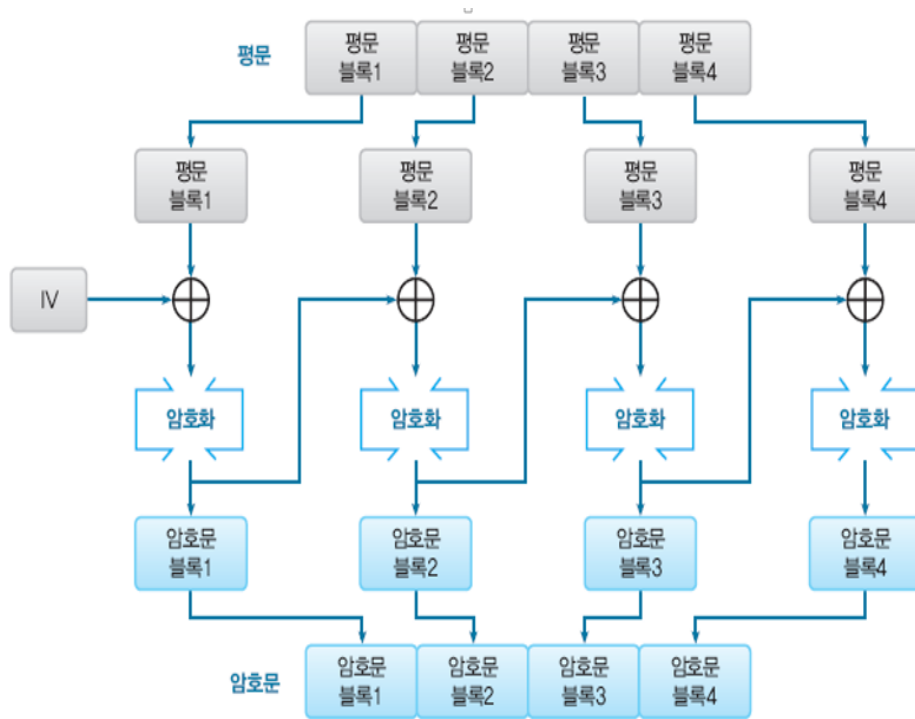
- 암호블록 연결 모드
- 1976년 IBM에 의해 개발
- 암호문 블록을 마치 체인(chain)처럼 연결시키기 때문에 붙여진 이름
- CBC 모드에서는 1 단계 앞에서 수행되어 결과로 출력된 암호문 블록에 평문 블록을 XOR 하고 나서 암호화를 수행
- 각각의 암호문 블록은 단지 현재 평문블록 뿐만 아니라 그 이전의 평문 블록들의 영향도 받게 된다



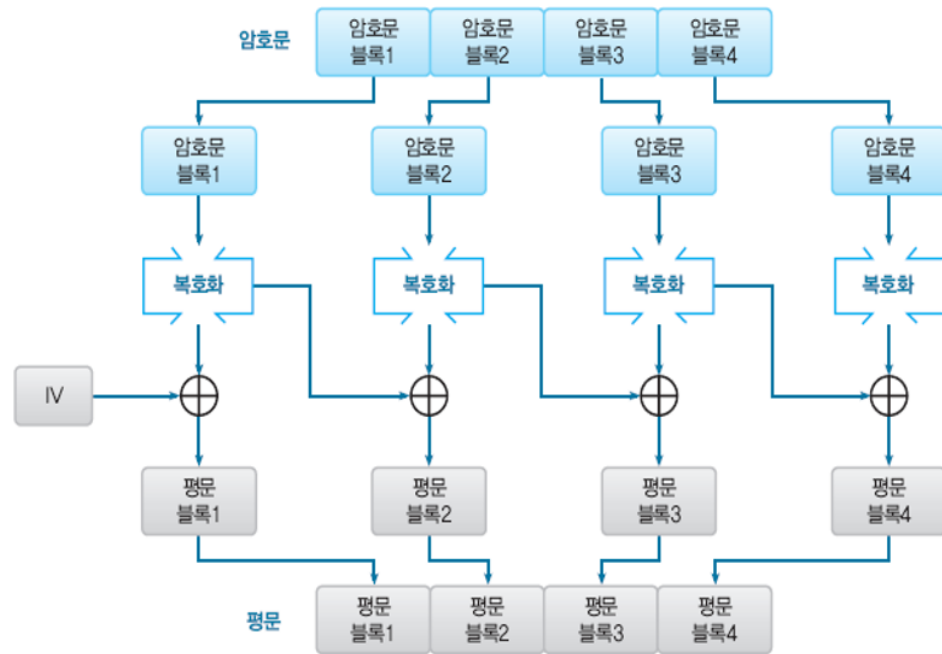
CBC 모드

- ECB 모드보다 안전
- 블록 암호화 운영 모드 중 보안 성이 제일 높은 암호화 방법으로 가장 많이 사용된다.
 - 첫번째 암호문에 대해서는 IV(Initial Vector)가 암호문 대신 사용된다. IV는 제 2의 키가 될 수 있다.
 - 암호문이 블록의 배수가 되기 때문에 복호화 후 평문을 얻기 위해서 Padding을 해야만 한다.
- 암호화가 병렬 처리가 아닌 순차적으로 수행되어야 한다

■ CBC 모드의 암호화와 복호화 과정



(a) CBC 모드에 의한 암호화



(b) CBC 모드에 의한 복호화

그림 5-3 • CBC 모드(암호 블록 체이닝 모드)

👤 CBC 모드의 특징

- 평문 블록은 반드시 「1 단계 앞의 암호문 블록」과 XOR을 취하고 나서 암호화
 - 따라서 만약 평문 블록1과 블록 2의 값이 같은 경우라도 암호문 블록1과 블록2의 값이 같아진다고는 할 수 없고,
 - ECB 모드의 결점이 CBC 모드에는 없다.
- 암호문 블록3을 만들고 싶다면 적어도 평문 블록의 1, 2, 3까지가 갖추어져 있어야만 한다.

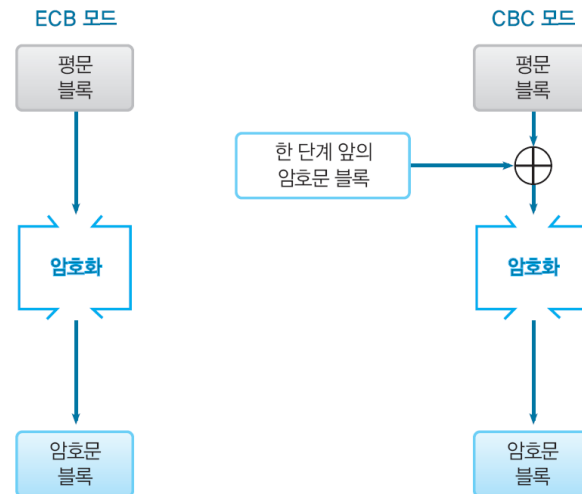
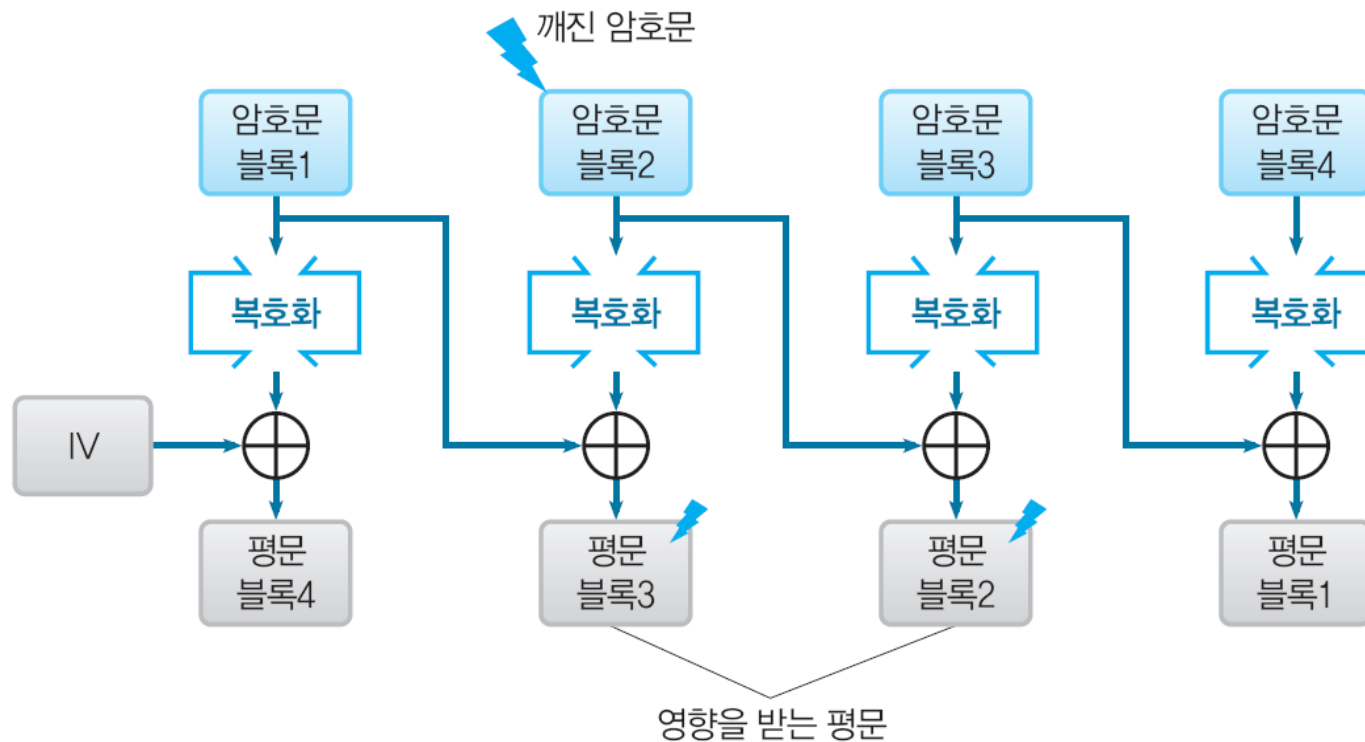


그림 5-4 • ECB 모드와 CBC 모드의 비교

👤 깨진 암호문

- CBC 모드에서 암호문 블록이 파손되면 2개의 평문 블록에 영향을 미친다



■ 패딩 오라클 공격

- 블록 암호의 패딩을 이용한 공격
- 패딩 내용을 조금씩 변화시켜 암호문을 여러 차례 송신
- 수신자가 올바르게 복호화 하지 못할 경우 오류를 관찰하여 평문 정보를 취득
- 패딩을 사용하는 모든 모드에 적용

■ 초기화 벡터는 난수(Random Number)로 부여

■ SSL/TLS의 TLS 버전 1.0

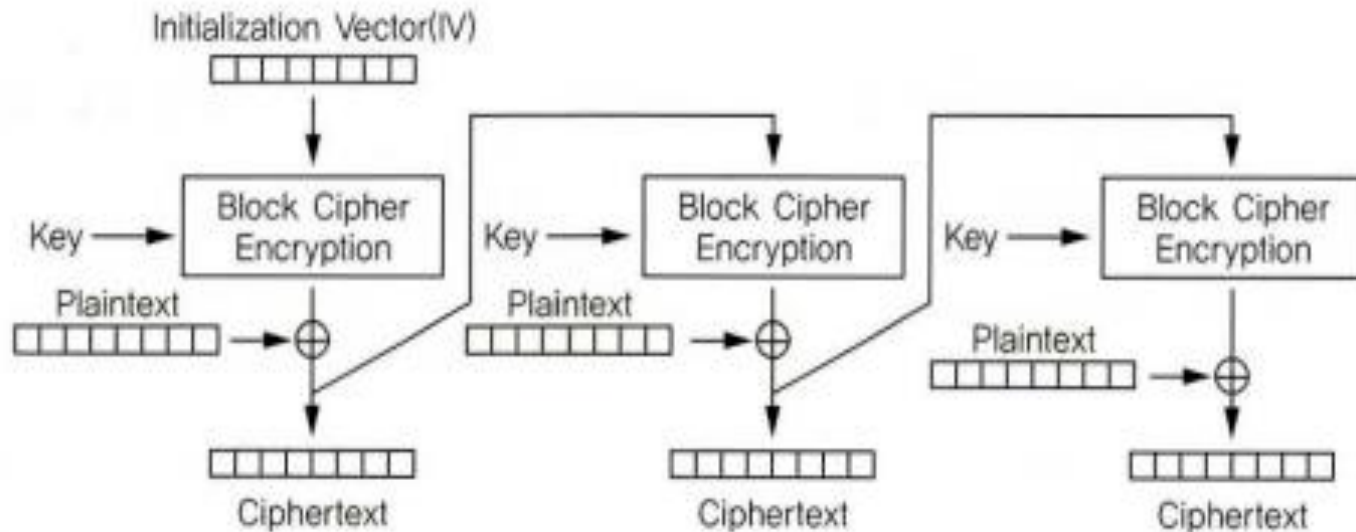
- 초기화 벡터를 이전 CBC 모드로 암호화한 마지막 블록을 사용
- 문제점 발견 후 TLS 버전 1.1 부터는 초기화 벡터를 명시적으로 부여

■ CBC 활용 예

- SSL/TLS : 통신의 기밀성 보호
- 예)
 - 3DES-EDL-CBC
 - AES-256-CBC : 키 길이가 256비트인 경우

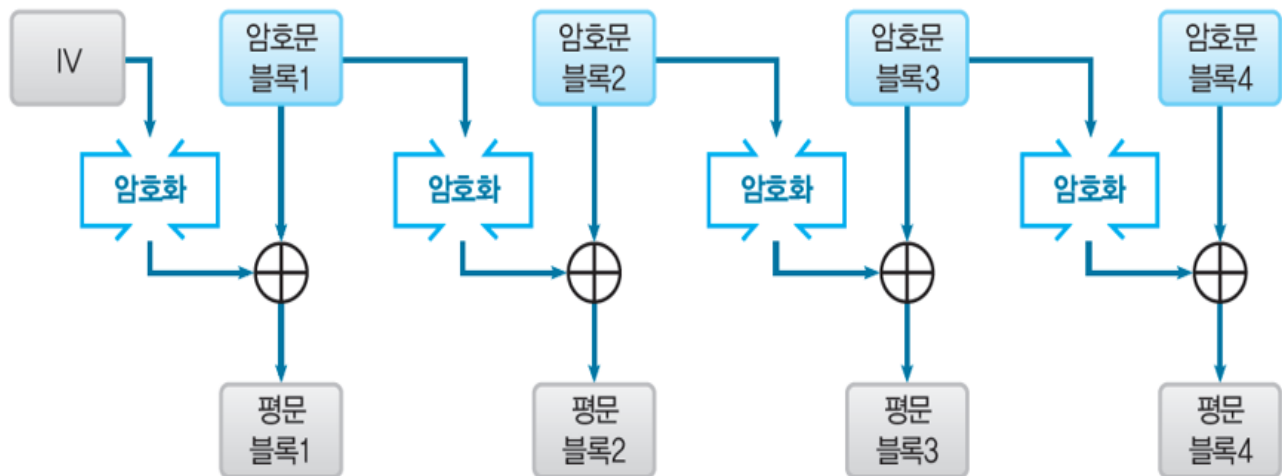
👤 Cipher FeedBack(CFB) Mode

- 암호블록 연결 모드
- CBC의 변형
- 스트림 암호처럼 구성
- 복호화에서 병렬처리 가능
- 평문과 암호문의 길이가 같음
 - Padding 필요 없음



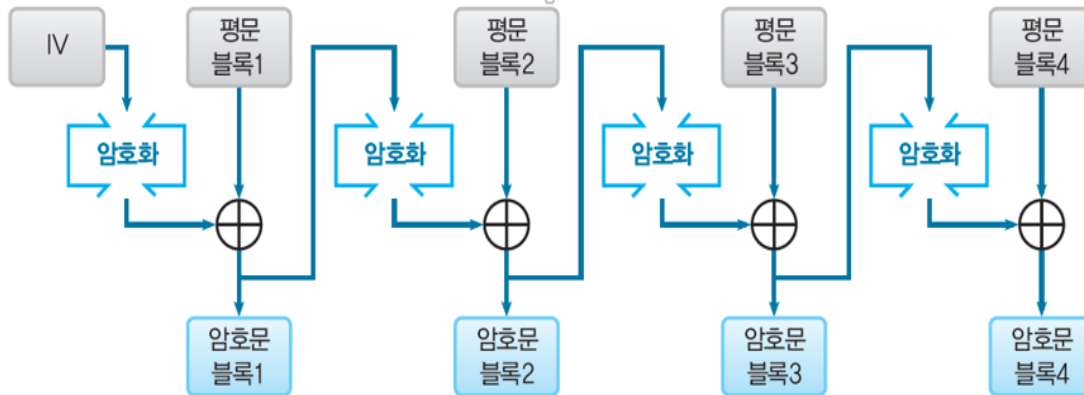
CFB 모드의 복호화

- CFB 모드에서 복호화를 수행할 경우, 블록 암호 알고리즘 자체는 암호화를 수행하고 있다는 것에 주의
- 키 스트림(key stream)은 암호화에 의해 생성

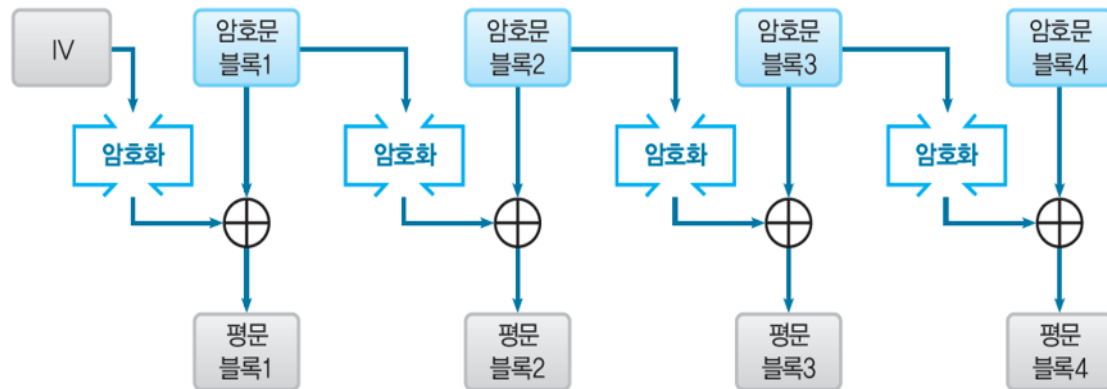


(b) CFB 모드에 의한 복호화

CFB 모드의 암호화와 복호화



(a) CFB 모드에 의한 암호화




(b) CFB 모드에 의한 복호화

CFB 모드와 스트림 암호

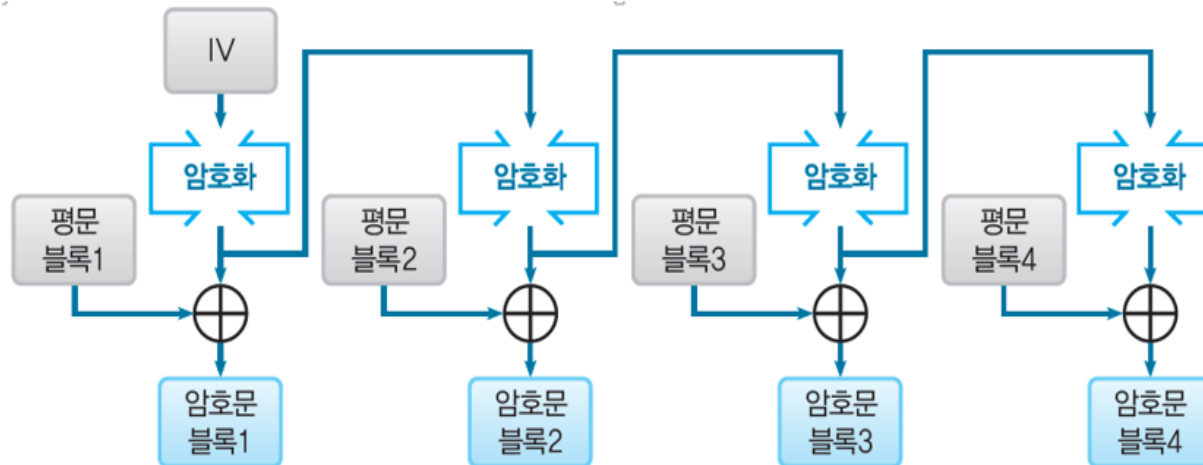
■ 키 스트림(key stream)

- CFB 모드에서 암호 알고리즘이 생성하는 비트 열
- 키 스트림을 생성하기 위한 의사난수 생성기로서 암호 알고리즘을 이용
- 초기화 벡터는 의사난수 생성기의 「seed(종자)」에 해당

- CFB 모드는 블록 암호를 써서 생성한 키를 이용하는 스트림 암호
 -  1 비트씩 암호화할 수도 있기 때문에 스트림 암호로 바꿀 수 있음

👤 Output-FeedBack (OFB) Mode

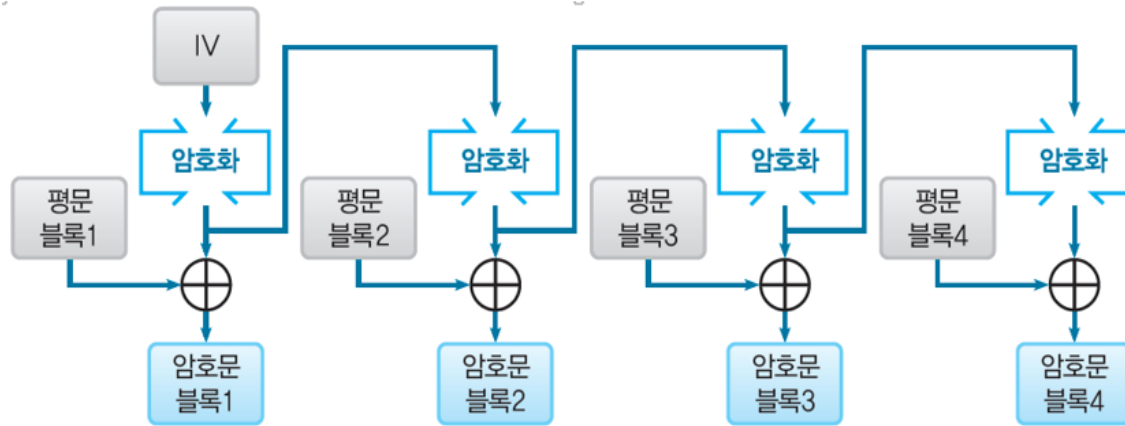
- 출력 피드백 모드
- 스트림 암호처럼 작동
- 평문과 암호문의 길이가 같음
 - Padding 필요 없음.
- 디지털화한 아날로그 신호에 많이 사용.



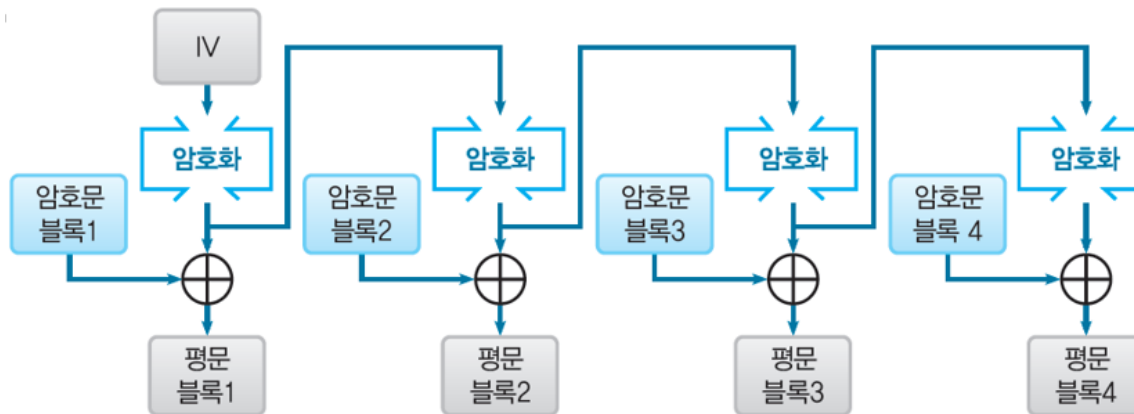
(a) OFB 모드에 의한 암호화

운영 모드

👤 OFB 모드에 의한 암호화와 복호화



(a) OFB 모드에 의한 암호화



(b) OFB 모드에 의한 복호화

👤 CFB 모드와 OFB 모드 비교

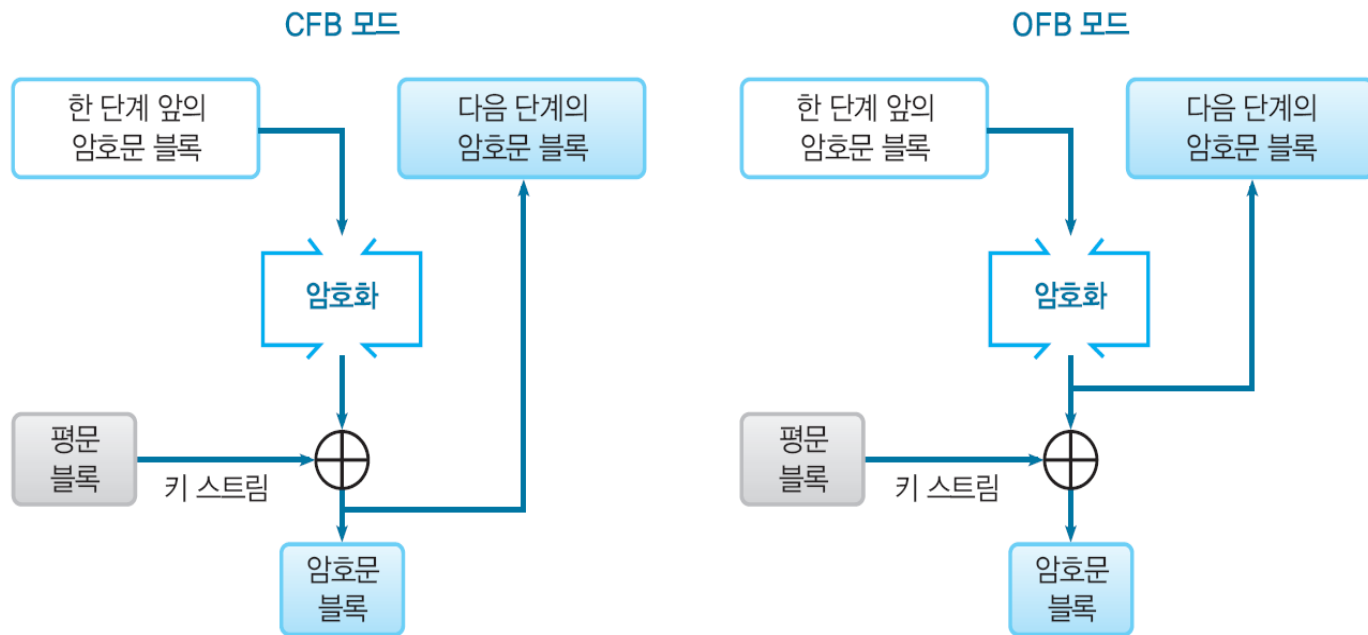
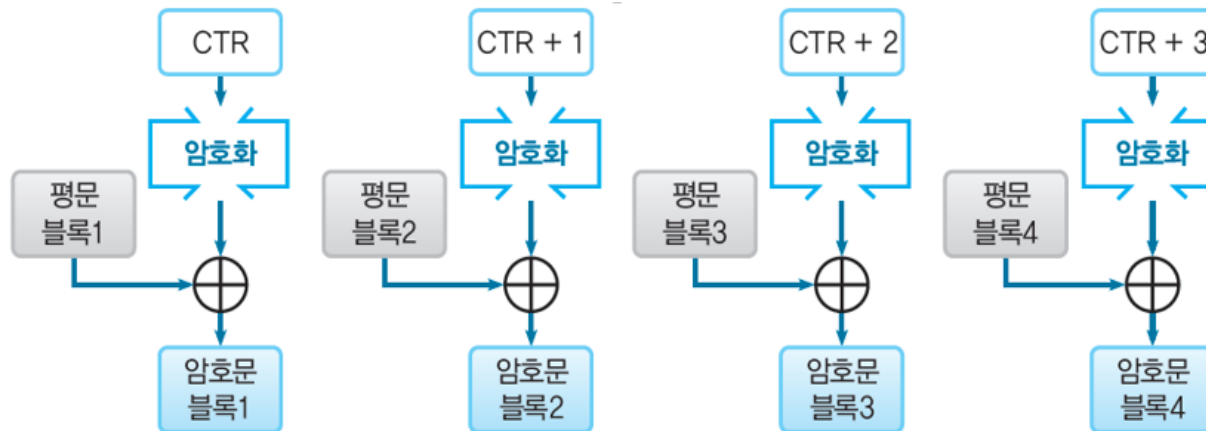


그림 5-14 • CFB 모드와 OFB 모드의 비교

카운트 (CTR, CounTeR) 모드

- 스트림 암호처럼 작동
- 1씩 증가해가는 카운터를 암호화해서 키 스트림을 만들어 내는 스트림 암호

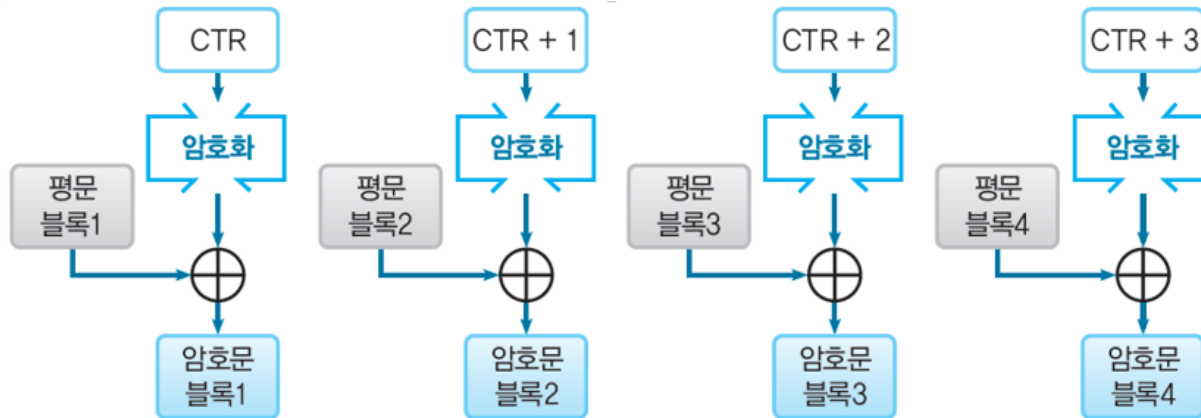
암호화



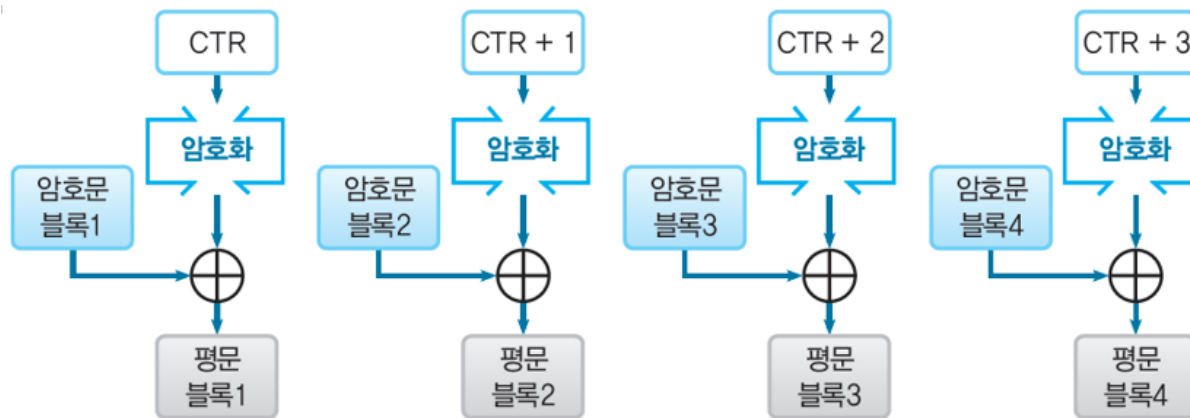
(a) CTR 모드에 의한 암호화

👤 CTR 모드의 암호화와 복호화

- 암호화와 복호화는 완전히 같은 구조



(a) CTR 모드에 의한 암호화




(b) CTR 모드에 의한 복호화

그림 5-15 • CTR 모드(카운터 모드)

카운터 초기값

- 암호화 때마다 다른 값(nonce, 비표)을 기초로 해서 작성

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 01



비표 블록 번호

카운터 만드는 법

- 평문 블록1용의 카운터(초기값)

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 01

- 평문 블록2용의 카운터

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 02

- 평문 블록3용의 카운터

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 03

- 평문 블록4용의 카운터

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 04 ::

👤 OFB 모드와 CTR 모드의 비교

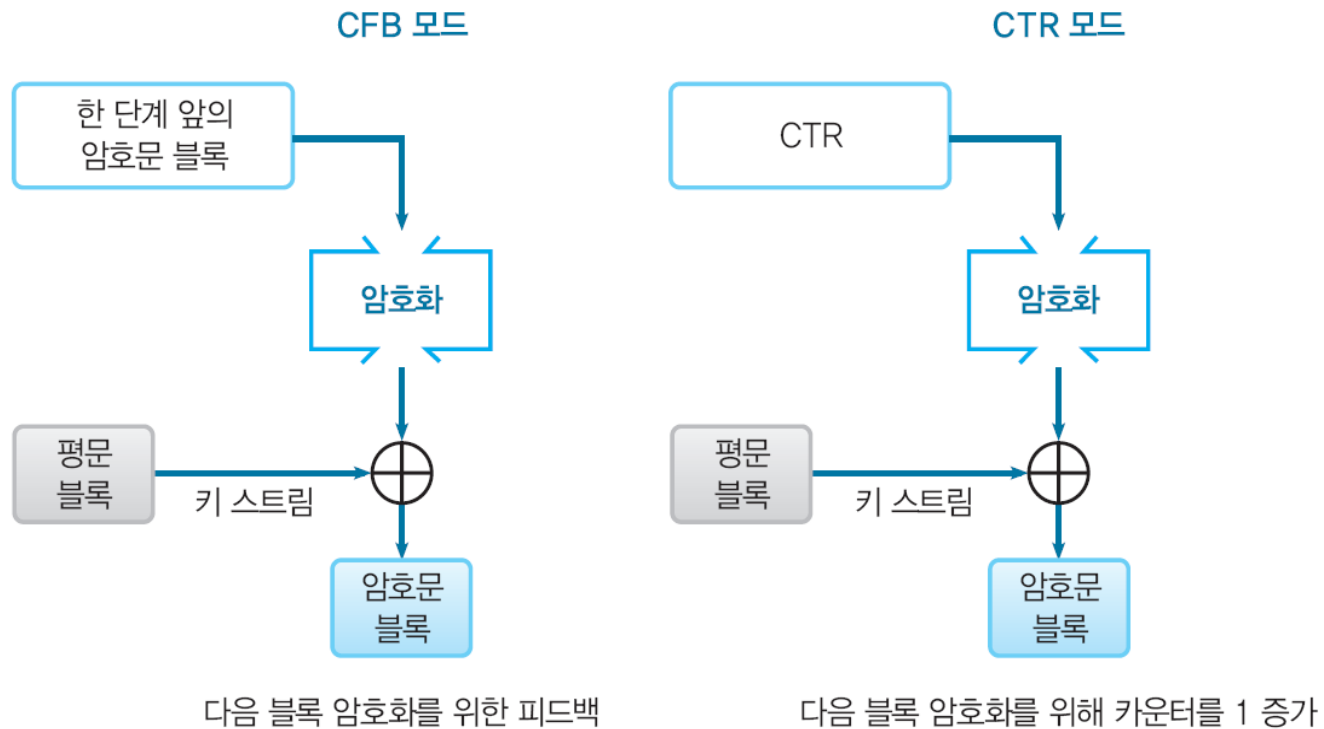


그림 5-16 • OFB 모드와 CTR 모드의 비교

CTR 모드의 특징


- CTR 모드의 암호화와 복호화는 완전히 같은 구조
- 프로그램으로 구현하는 것이 매우 간단
- OFB 모드와 같은 스트림 암호의 특징
- CTR 모드에서는 블록을 임의의 순서로 암호·복호화할 수 있다
- 병렬 처리가 가능한 시스템에서는 CTR 모드를 이용하여 자료를 고속으로 처리

무결성

- 인가되지 않은 데이터의 수정을 방지하거나 적어도 탐지하는 것
- 예) 인터넷 은행에서 자금 이동
 - 비밀성은 제공이 용이하나 무결성은 심각

암호화는 비밀성을 제공

- 데이터를 보안성이 없는 채널에서 전송
- 보안성이 없는 미디어에서 안전하게 저장

 암호화 자체만으로 무결성을 확신할 수 없음. (일회성 암호와 ECB 공격 등)

메시지 인증 코드 (MAC)

- Message Authentication Code
- 데이터 무결성을 위해 사용
- 무결성은 비밀성과 동일하지 않음

MAC은 CBC의 나머지로 계산

- Compute CBC 암호화를 계산하여 암호문 블록의 마지막만 저장