대칭키 암호

3.1 개요

3.2 스트림 암호

3.3 블록 암호

3.4 무결성

3.5 양자 컴퓨터와 대칭 암호

3.6 요약

3.1 개요

■ 개요

- 대칭키 암호는 스트림 암호와 블록 암호로 구분됨
- 스트림 암호
  - 긴 키 스트림(S)을 생성해 평문(P)과 XOR -> 실시간 암호화, 하드웨어에 최적화
  - $C = P \oplus S$

- 블록 암호
  - 고정된 길이(ex, 128비트)의 블록에 대해 가역 변환(E_K)를 제공.
  - 긴 메시지는 운용 모드 (CTR, CBC, GCM 등)를 처리
  - 암호화, 소프트웨어 최적화

3.1 개요

개요

- 대칭키 암호 : 암호화와 복호화에 같은키(비밀키)를 사용하는 방식
- 보낸 사람과 받는 사람이 공통의 비밀키를 미리 공유해야한다.
- 암호화 : $C = E_K(P)$
- 복호화 : $P = D_K(C)$

- 장점 : 계산이 빠름, 구현이 간단, 오래전 부터 연구되어 안정적
- 다점 : 키 분배 문제 => 서로 멀리 덜어진 사용자가 어떻게 비밀키를 안전하게 공유할 것인가

3.1 개요

■ 스트림 암호(stream cipher)

- 전통적인 일회성 암호와 개념이 비슷함
- 일회성 암호와 같은 방식으로 키를 긴 비트열로 늘여서 사용
- 일회성 암호처럼 섀넌의 원칙 중 혼돈 이론만 적용
- A5/1, RC4

■ 블록 암호(block cipher)

- 고전 코드북 암호의 현대 버전
- 블록 암호는 키를 결정하는 특정 코드북과 다양한 다른 코드북 포함
- 블록 암호는 실제 코드북의 전자 버전임
- 혼돈과 확산의 원칙을 모두 사용
- DES, AES, TEA

3.2 스트림 암호

- 길이가 *n*비트인 키 *K*를 가지며, 이 키를 긴 키 스트림(key stream)으로 늘이고 키 스트림은 평문 *P*와 XOR 연산을 하여 암호문 *C* 생성
  - 평문 (P)와 키스트림(S)를 XOR 연산하여 암호문 생성
    $C = P \oplus S$

- 동일한 키 스트림은 XOR 함수를 통해 암호문 *C*를 평문 *P*로 복호화하는 데 사용함

  $P = C \oplus S$

- 예시:

  평문: 1101

  키스트림: 1010

  암호화 :

  복호화:

3.2 스트림 암호 – A5/1

- GSM(Global System for Mobile Communications) 통신을 보호하기 위해 개발된 스트림 암호
  - 용도: 2G GSM 음성 데이터 보호
  - 세개의 LSFR(Linear Feedback Shift Register) 사용하여 키 스트림 생성
    - X register : 19 bit
    - Y register : 22 bit
    - Z register : 23 bit
  - 키 길이 : 64 bit
  - 특징: 저전력 하드웨어 최적화

3.2 스트림 암호 – A5/1

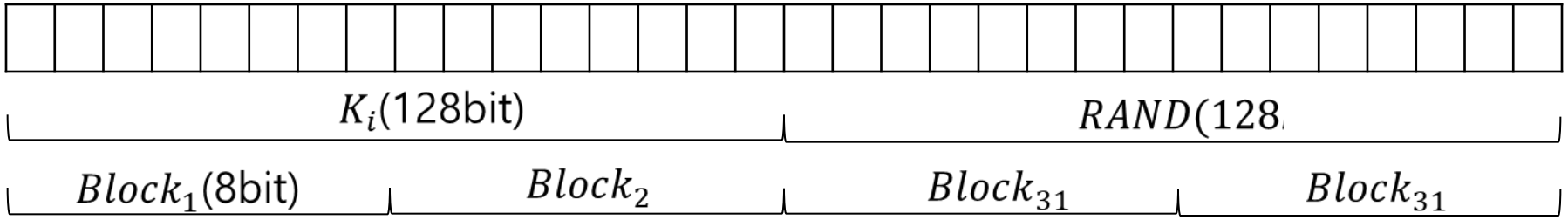
- 세션 키 생성(A8)
  - 입력
    - 가입자 키 K_i(128bit, SIM 내부 보관)
    - 난수 $RAND$(128bit, 기지국 전송)
  - A8 알고리즘
    - 내부에서 $(K_i, RAND)$ → 압축함수(COMP128 등) 처리
    - 출력값: 세션 키 K_c(64bit)

3.2 스트림 암호 – A5/1

COMP128 알고리즘 과정

- 입력
  - K_i(128bit): 가입자 개인 키, Sim 카드 내부 저장
  - RAND(128bit): 네트워크에서 전송하는 난수

  - 두 값을 붙여서 총 256비트 입력 생성
    - 128bit(K_i) + 128bit(RAND) = 256 bit
    - 이 256bit를 다시 32개의 8bit 블록으로 나눔



3.2 스트림 암호 – A5/1

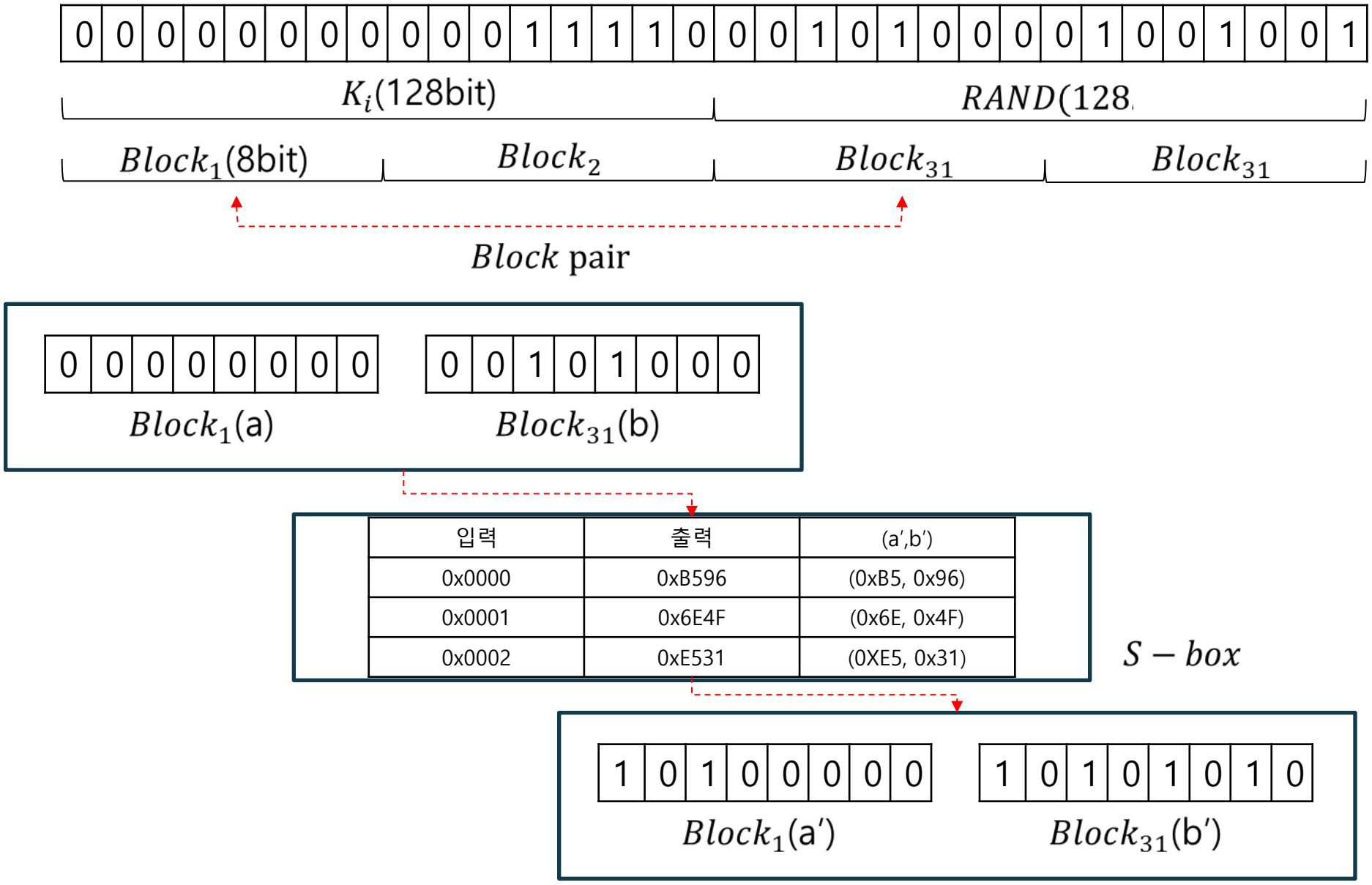
■ COMP128 알고리즘 과정

- 총 8 라운드로 이루어진 함수
- 각 라운드에서 입력 블록들을 섞어 비선형성 확보

- 방식
  - 32개의 값(바이트)/ 32개의 Block를 16쌍(pair)로 묶음
  - 각 쌍에 대해:
    - 두 입력 값 (a,b)를 가져옴
    - (a,b)를 S-box 같은 비선형 치환 테이블에 통과시켜 두 개의 새로운 값 (a′, b′)을 생성
    - 이렇게 해서 32개 값 전체가 갱신됨
  - 라운드 가 끝나면 값들을 다시 섞어 배치(이과정을 8번 반복)
  - 8라운드가 끝나면 32개의 값(=256bit)가 나오지만 상위 128bit만 이용
    - 이 중
      - 상위 32 bit -> A3 결과(RES)
      - 하위 64 bit -> A8 결과(세션 키 Kc)

* S-box(Substitution Box) : 작은 입력 비트를 받아서 전혀 다른 출력 비트로 바꿔주는 룩업 테이블

3.2 스트림 암호 – A5/1


0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1
K_i(128bit)
$RAND$(128
$Block_1$(8bit)
$Block_2$
$Block_{31}$
$Block_{31}$
$Block$ pair
0 0 0 0 0 0 0 0
$Block_1$(a)
0 0 1 0 1 0 0 0
$Block_{31}$(b)
입력 출력 (a′,b′)
0x0000 0xB596 (0xB5, 0x96)
0x0001 0x6E4F (0x6E, 0x4F)
0x0002 0xE531 (0XE5, 0x31)
$S-box$
1 0 1 0 0 0 0 0
$Block_1$(a′)
1 0 1 0 1 0 1 0
$Block_{31}$(b′)

3.2 스트림 암호 – A5/1

■ A5/1 초기화

- K_c : 세션키 (64bit)
- F_N : 프레임 넘버 (22bit)

■ 간략화 된 동작 예

- K_c = 1011 0110 (8bit)
  왼쪽에서 오른쪽으로 주입
- F_N = 0001 10 (6bit)
- X, Y, Z = 00000

- 레지스터 feedback 계산 규칙 가정
  - 모두 left shift => K_c를 LSB에 주입
  - $X = X_5 \oplus X_4 \oplus X_1$
  - $Y = Y_5 \oplus Y_2$
  - $Z = Z_5 \oplus Z_3 \oplus Z_2$

- 1) K_c의 첫 비트 (1)
  - Feedback 계산
    - $X = X_5 \oplus X_4 \oplus X_1 = 0 \oplus 0 \oplus 0 = 0$
    - $Y = Y_5 \oplus Y_2 = 0 \oplus 0 = 0$
    - $Z = Z_5 \oplus Z_3 \oplus Z_2 = 0 \oplus 0 \oplus 0 = 0$
  - $\text{Feedback}(0) \oplus 1$
    - $\text{X} : 0 \oplus 1 \rightarrow 00000 \rightarrow shift\text{&주입} \rightarrow 00001$
    - $\text{Y} : 0 \oplus 1 \rightarrow 00000 \rightarrow shift\text{&주입} \rightarrow 00001$
    - $\text{Z} : 0 \oplus 1 \rightarrow 00000 \rightarrow shift\text{&주입} \rightarrow 00001$

3.2 스트림 암호 – A5/1

- 2) K_c의 두번째 비트 (0)
  - $\mathrm{K_c}$ = 1011 0110
  - $\mathrm{X} = Y = Z = 00001$
  - Feedback 계산
    - $X = X_5 \oplus X_4 \oplus X_1 = 0 \oplus 0 \oplus 0 = 0$
    - $Y = Y_5 \oplus Y_2 = 0 \oplus 0 = 0$
    - $Z = Z_5 \oplus Z_3 \oplus Z_2 = 0 \oplus 0 \oplus 0 = 0$
  - 주입값 $(1) \oplus 0$
    - $\mathrm{X} : 1 \oplus 0 \rightarrow 00001 \rightarrow shift$&주입 $\rightarrow 00010$
    - $\mathrm{Y} : 0 \oplus 0 \rightarrow 00001 \rightarrow shift$&주입 $\rightarrow 00010$
    - $\mathrm{Z} : 0 \oplus 0 \rightarrow 00001 \rightarrow shift$&주입 $\rightarrow 00010$
- 3) K_c의 세번째 비트 (1)
  - $\mathrm{K_c}$ = 1011 0110
  - $\mathrm{X} = Y = Z = 00010$
  - Feedback 계산
    - $X = X_5 \oplus X_4 \oplus X_1 = 0 \oplus 0 \oplus 1 = 1$
    - $Y = Y_5 \oplus Y_2 = 0 \oplus 0 = 0$
    - $Z = Z_5 \oplus Z_3 \oplus Z_2 = 0 \oplus 0 \oplus 0 = 0$
  - 주입값 $(1) \oplus 1$
    - $\mathrm{X} : 1 \oplus 1 \rightarrow 00010 \rightarrow shift$&주입 $\rightarrow 00100$
    - $\mathrm{Y} : 1 \oplus 0 \rightarrow 00010 \rightarrow shift$&주입 $\rightarrow 00101$
    - $\mathrm{Z} : 1 \oplus 0 \rightarrow 00010 \rightarrow shift$&주입 $\rightarrow 00101$
- FN 주입시에도 동일 과정 반복

3.2 스트림 암호 – A5/1

- $Warmning\ up$ 단계
  - 다수결 클로킹 적용 / 이 과정을 100번 반복
- 간략화된 다수결 과정
  - $m = maj(x_2, y_3, z_3)$ 클록비트
    - x_2가 m이면 feedback 주입
    - y_3가 m이면 feedback 주입
    - z_3가 m이면 feedback 주입

- 예시
  - $X = 011100$
  - $Y = 110000$
  - $Z = 001111$
  - $m = maj(x_2, y_3, z_3) = maj(0,0,1) = 0$
  - Feedback 계산
    - $X = X_5 \oplus X_4 \oplus X_1 = 0 \oplus 1 \oplus 0 = 1$
    - $Y = Y_5 \oplus Y_2 = 1 \oplus 0 = 1$
    - $Z = Z_5 \oplus Z_3 \oplus Z_2 = 0 \oplus 1 \oplus 1 = 0$
  - 주입값
    - $X : 1 \rightarrow 011100 \rightarrow shift \rightarrow 111001$
    - $Y : 1 \rightarrow 110000 \rightarrow shift \rightarrow 100001$
    - $Z : 0 \rightarrow 001111 \rightarrow shift \rightarrow 011110$

3.2 스트림 암호 – A5/1

A5/1: 시프트 레지스터

- 레지스터 X 단계

$$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$$

$$x_i = x_{i-1} \ for\ i = 18, 17, 16, \cdots, 1$$

$$x_0 = t$$

- 레지스터 Y 단계

$$t = y_{20} \oplus y_{21}$$

$$y_i = y_{i-1} \ for\ i = 21, 20, 19 \cdots, 1$$

$$y_0 = t$$

- 레지스터 Z 단계

$$t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$$

$$z_i = z_{i-1} \ for\ i = 22, 21, 20, \cdots, 1$$

$$z_0 = t$$

3.2 스트림 암호 – A5/1

A5/1: 키 스트림

- $m = \mathrm{maj}(x_8, y_{10}, z_{10})$

  - $x_8 = m$이면, X 단계 수행
  - $x_8 = m$이면, X 단계 수행
  - $z_{10} = m$이면, Z 단계 수행

  - $s = x_{18} \oplus y_{21} \oplus z_{22}$

3.2 스트림 암호 – A5/1

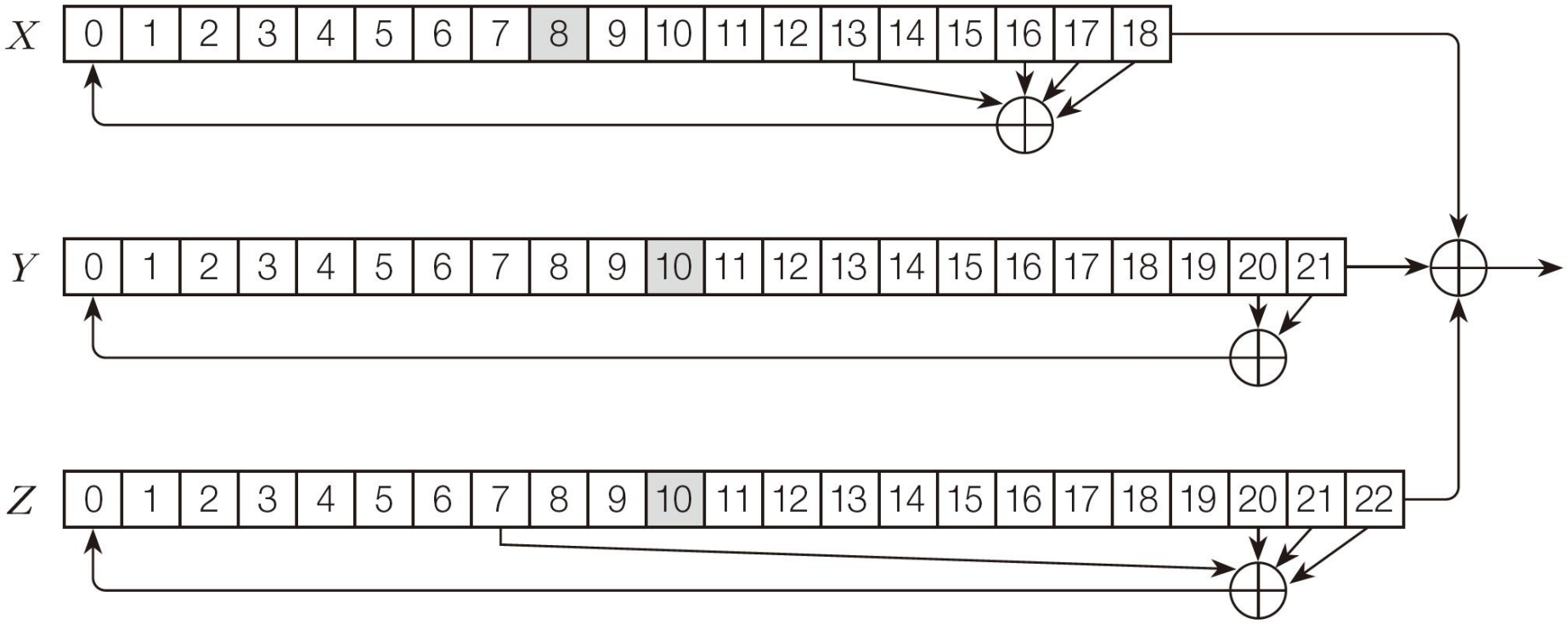


그림 3-1 A5/1 키 스트림 생성

3.2 스트림 암호 – A5/1

- A5/1이 하드웨어에 최적화인 이유
  - LSFR은 Shift와 XOR 연산밖에 없음
  - 레지스터가 동시 동작함으로 병렬성
- 스트림 암호에서 블록 암호로 대거 이동한 이유
  - 시프트 레지스터 암호는 하드웨어에서 효율적이나 소프트웨어로 구현하면 느림
  - 오늘날에는 빠른 프로세서로 소프트웨어에서 더 많은 작업이 수행되므로 블록 암호로 이동

3.2 스트림 암호 – A5/1

■ Simple A5/1 실습

- Simple LSFR 사양

레지스터	길이	피드백 비트	클락비트
X	5bit	[2,4]	2
Y	6bit	[1,5]	3
Z	7bit	[1,5,6]	4

  - $X = X_4 \oplus X_2$
  - $Y = Y_5 \oplus Y_1$
  - $Z = Z_6 \oplus Z_5 \oplus Z_1$
- LSFR 초기화
  1. 모든 레지스터 상태를 0 으로 초기화
  2. 키 로딩(Comp128 K_c : 0xBEEF)
  3. 프레임 로딩(FN : 0x3A)
  4. 워밍업 10회
  5. S stream 비트는 X[4], y[5], z[6]
- 키스트림 8비트 생성
  - 결과 : 1 0 0 1 1 0 0 0

3.2 스트림 암호 – RC4

■ RC4 알고리즘

- 소프트웨어 구현에 최적화
- 각 단계에서 키 스트림 바이트를 생성(비트 단위가 아니라 효율이 좋음)
- 256 바이트 크기의 상태 배열 S를 이용

3.2 스트림 암호 – RC4

■ RC4 알고리즘 -초기화

- 0~255 까지 순열로 상태 배열 S초기화
- 키를 반복적으로 대입하여 배열 S를 섞음

표 3-1 RC4 초기화

```
for i = 0 to 255
    S[i] = i
    K[i] = key[i(mod N)]
next i
j = 0
for i = 0 to 255
    j = (j + S[i] + K[i])(mod 256)
    swap(S[i],S[j])
next i
i = j = 0
```

i	j 계산식	j 값	교환 후 S 배열
0	(0+0+1) mod 5	1	[1,0,2,3,4]
1	(1+0+2) mod 5	3	[1,3,2,0,4]
2	(3+2+3) mod 5	3	[1,3,0,2,4]
3	(3+2+1) mod 5	1	[1,2,0,3,4]
4	(1+4+2) mod 5	2	[1,2,4,3,0]

3.2 스트림 암호 – RC4

■ RC4 알고리즘 –키 스트림 생성

표 3-2 RC4 키 스트림 바이트

$i = (i + 1) \pmod{256}$

$j = (j + S[i]) \pmod{256}$

$\text{swap}(S[i], S[j])$

$t = (S[i] + S[j]) \pmod{256}$

$\text{keystreamByte} = S[t]$

4	(1+4+2) mod 5	2	[1,2,4,3,0]

단계	i	j 계산식	j 값	교환 후 S 배열	t 값	출력 keystream
1	1	(0+S[1]) mod 5 = 2	2	[1,4,2,3,0]	(4+2) mod 5 = 1	4
2	2	(2+S[2]) mod 5 = 4	4	[1,4,0,3,2]	(0+2) mod 5 = 2	0

3.2 스트림 암호 – RC4

■ RC4 알고리즘

- 스스로 수정하는 검색표
- 명쾌하고 단순하며 소프트웨어에서 효율적
- SSL과 WEP 프로토콜 등 많은 응용 분야에서 사용되었지만 32비트 프로세서에도 최적화되어 있지 못함(옛날 8비트 프로세서에 최적화)
- 몇몇 약점이 발견되면서 현재 많이 쓰이지 않음
  - 첫 키 스트림 바이크는 S[1], S[S[1]]만으로 만들어짐
  - 두번째는 S[2]가 개입
  - 같은 맥락으로 256개의 값을 유추 할 수 있음
  - 따라 Drop-N 방식으로 보안 강화
    - N은 256의 배수, 클 수록 보안성이 높다

3.3 블록 암호

■ 반복되는 블록 암호

- 반복되는 블록 암호는 평문을 **고정 크기 블록**으로 나누고 고정 크기 블록 단위로 암호문 생성
- 함수 *F*를 **반복 수행**하여 평문에서 암호문 생성
- 함수 *F*는 이전 회전(round)의 출력과 키 *K*에 의해 결정되며 회전 함수라 함
- 모양 때문이 아니라 여러 번 반복 또는 여러 회에 걸쳐 적용되어 회전 함수라 함
- 블록 암호의 설계 목표는 보안과 효율성
  - 보안 : 키를 모르면 해독 불가
  - 효율성 : 빠르게 동작
- 둘 중 하나에 초점을 두고 개발하는 것은 쉽지만 안전하고 효율적인 블록 암호는 기술적으로 매우 어려움

3.3 블록 암호 – 파이스텔 암호

■ 파이스텔 암호

- 함수 *F*의 특정 회전과 무관하게 복호화를 할 수 있는 것이 특징
- *F*의 출력이 정확한 수의 비트를 생성하면 어떠한 함수 *F*도 파이스텔 암호에서 작동함. 함수 *F*의 역함수가 반드시 존재할 필요는 없다는 점에서 장점임
- 그러나, *F*의 모든 가능한 선택에 대해 안전하지 않을 수 있음
- 공격자가 합법적인 회전 함수로 암호화된 내용을 복호화할 수 있음
- 파이스텔 암호의 보안은 회전 함수 *F*와 키 스케줄로 결정, 분석은 *F*에 초점을 맞춤
- 파이스텔 암호 기술의 결점은 비트 절반이 각 회전에 영향을 받지 않는다는 것
- 이러한 비효율성으로 많은 블록 암호들이 파이스텔 암호가 아님

3.3 블록 암호 – 파이스텔 암호

■ 파이스텔 암호

- 기본 수식
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$
- 초기 입력
  - 평문 $P = (L_0, R_0)$
  - 키 스케줄에 따른 라운드 별 키 준비(키 스케줄[DES, AES]로 파생된 유한개)
- 예시
  - $P = (L_0, R_0) = (13, 6)$
    - $L_0 = 1101_2, R_0 = 0110_2$
  - 라운드 키 : $K_1 = 10, K_2 = 7$ (4bit)
  - 라운드 함수 $F(R, K) = (R + K) mod\ 16$ / 알고리즘마다 F 구조는 다름!

단계	L(10진/2진)	R(10진/2진)	F(R,K)	갱신식
초기	13 / 1101	6 / 0110	—	—
1라운드 후	6 / 0110	13 / 1101	0/0000	$L_1 = R_0, R_1 = L_0 \oplus 0$
2라운드 후	13 / 1101	2 / 0010	4/0100	$L_2 = R_1, R_2 = L_1 \oplus 4$

3.3 블록 암호 – DES

- DES(Data Encryption Standard) 알고리즘
  - IBM 루시퍼 암호(1970년대 초) -> DES의 뿌리
  - NBS(현 NIST)가 상업용 표준 암호성 필요성을 인식
  - NSA의 검토과정
    - 키 길이 128비트 -> 56비트로 축소
    - 일부 S-box 수정 -> 당시 "약화설" 논란
  - 1977년 공식 표준 채택

3.3 블록 암호 – DES

■ DES 알고리즘

- 입력 블록 : 64비트
- 출력 블록 :64 비트
- 키 : 64비트(56비트 사용 + 패리티 8비트)

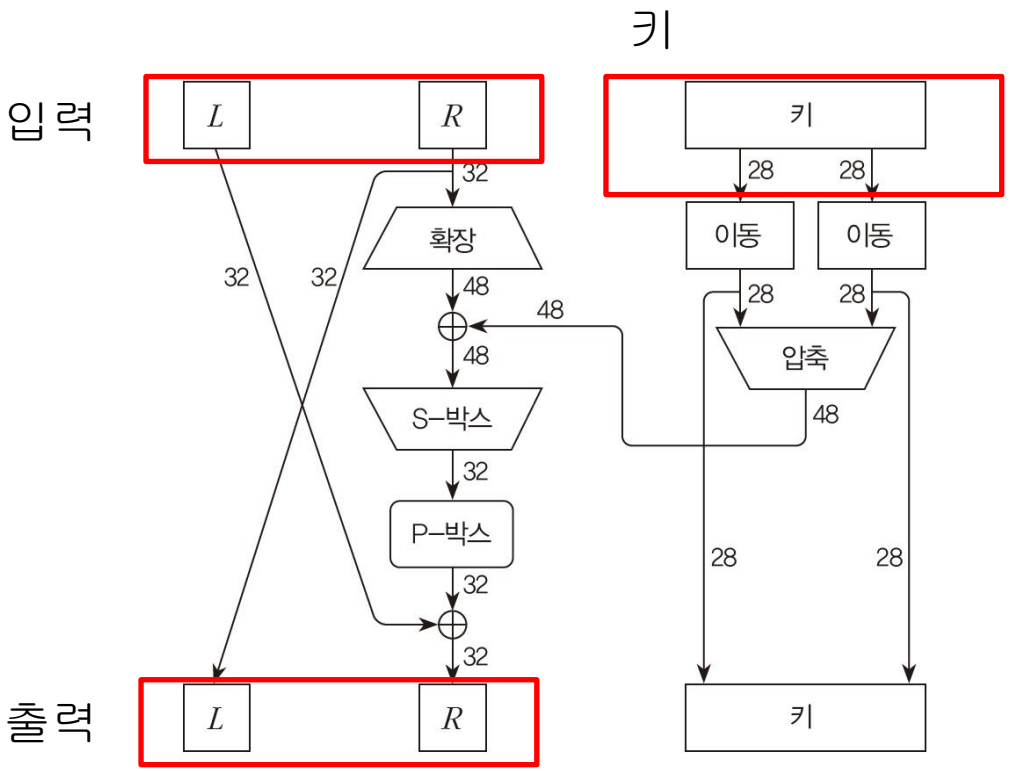


그림 3-2 DES의 단일 회전

3.3 블록 암호 – DES

■ DES 알고리즘

- 파이스텔 구조
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$
  - $i = 1, 2, .., 16$
  - 최종 암호문:(R_{16}, L_{16})

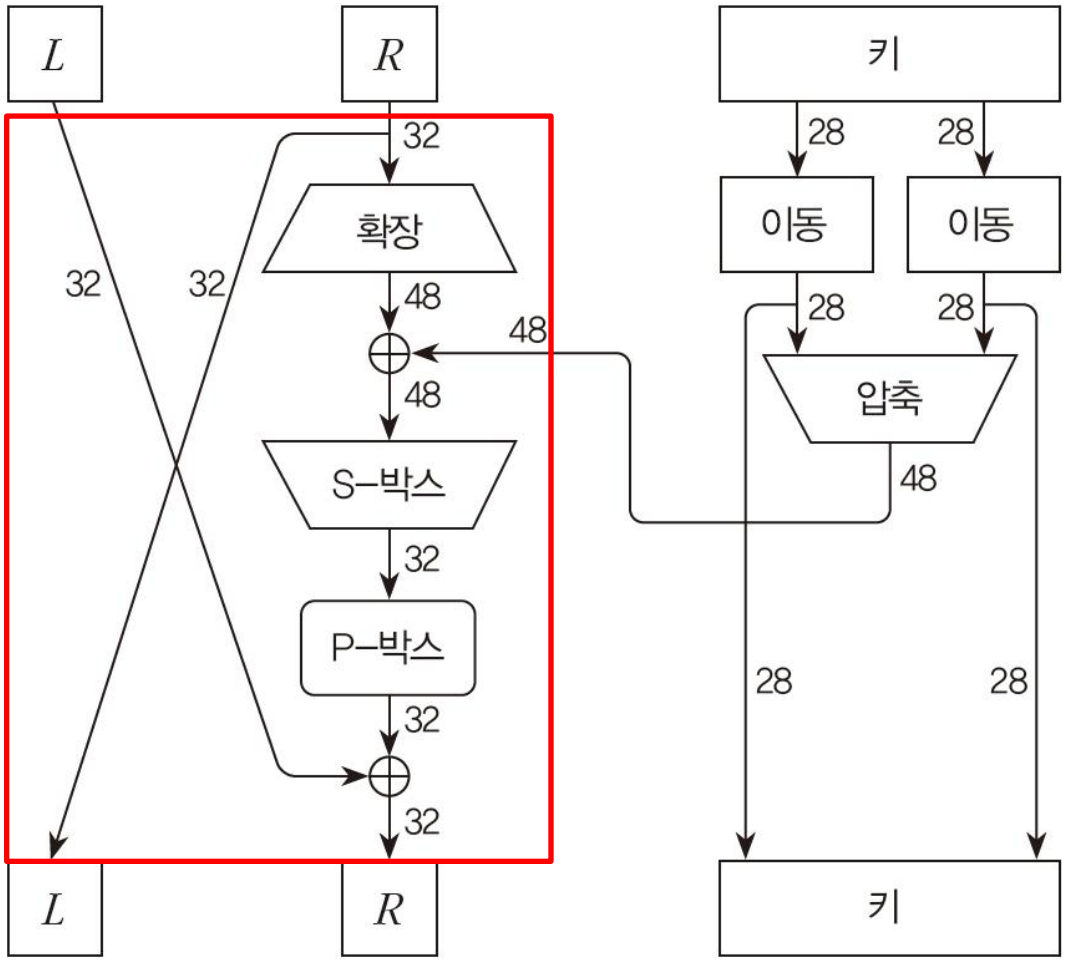


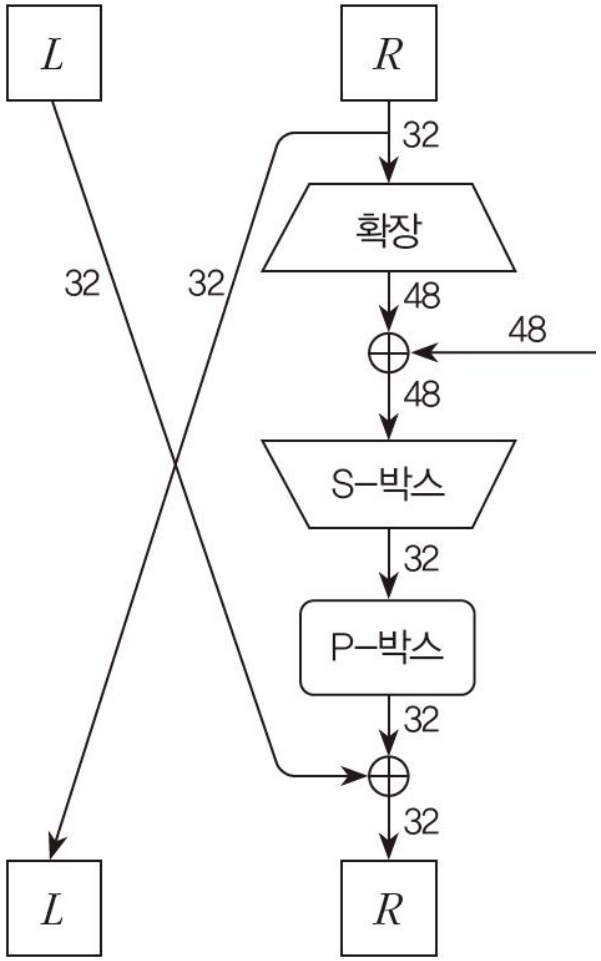
그림 3-2 DES의 단일 회전

3.3 블록 암호 – DES

■ DES 알고리즘

▪ F의 세부 절차

1. 확장 E-Box (32-> 48 비트)
   - R(32비트)를 비트 반복 재배열 하여 48비트 생성
   - 경계 비트 중복 포함(확산 목적)
2. 라운드 키와 XOR(48비트)
   - $E(R_{i-1}) \oplus K_i$
3. S-box 치환(48->32 비트)
   - 48비트를 6비트 x 8 블록으로 분할
   - 각 블록 -> S-box(6->4 매핑) -> 총 32비트 출력
4. P-box 순열(32비트)
   - 32비트를 재배열 하여 다음 라운드 입력값으로 전달



3.3 블록 암호 – DES

■ DES 알고리즘

- S-box 구조
  - 각 S-box:6비트 입력 ->4비트 출력
  - 행 선택 : 첫번째 + 마지막 비트(2비트-> 0~3행)
  - 열 선택 : 중간 4비트 (0~15 열)
  - 출력 = S-box[row][col]
  - 총 8개의 S-box 사용
    -> 병렬 처리로 48비트 -> 32비트 축소

표 3-3 DES S-박스 1(16진수)

	$b_1 b_2 b_3 b_4$															
$b_0 b_5$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

3.3 블록 암호 – DES

■ DES 알고리즘

▪ P-box 순열

- S-Box 출력(32bit)을 다시 섞는 과정
- 목적 : S-box 출력의 비트들이 다음 라운드 전체에 고르게 확산되도록 설계
- 즉, 한 비트 변화가 최종 출력 전체에 퍼지는 효과 -> Avalanche 효과

▪ 키 스케쥴

- 입력 키 64bit -> 패리티 제거 -> 56 bit
- 28 + 28 bit 분할(C0, D0)
- 각 라운드에서 왼쪽 시프트 (1bit 혹은 2bit)
- Pc-2 압축 선택: 56 비트 중 48 비트 추출 -> 라운드 키 K_i
- 총 16개의 키 생성

■ DES 키 스케줄 알고리즘

표 3-4 DES 키 스케줄 알고리즘

$i=1,2,\cdots,n$ 회전 각각에 대하여

$LK=r_i$비트만큼 LK를 왼쪽으로 시프트

$RK=r_i$비트만큼 RK를 왼쪽으로 시프트

보조키 K_i의 왼쪽 절반은 LK의 LP비트로 구성

보조키 K_i의 오른쪽 절반은 RK의 RP비트로 구성

next i

3.3 블록 암호 – 삼중 DES

■ DES의 문제

- 키 공간(길이 : 56비트;가능한 키 개수 $2^{56} \approx 7.2 \times 10^{16}$)
- 하드웨어 발전에 따라 brute-force 가능
  - 단일 CPU 11.4년 -> 16코어 1.14년 -> 보급형 GPU 1.4개월 -> 하이엔드 GPU 8.34d

■ 이중 DES

- 암호화 $C = E_{K_2}\left(E_{K_1}(P)\right)$ / 키 공간 : 56 + 56 = 112비트
- 56비트인 DES의 키 길이가 충분하지 않아 이중 DES로 암호화
- 두 개의 DES 실행으로 효율성 떨어지는 것이 단점이지만 두 개의 56비트 DES 키로 112비트 키를 사용할 수 있는 것은 장점
- 그러나 이중 DES는 단독 DES보다 안전하다고 할 수 없음
  - 실효적 보안 강도는 57비트

3.3 블록 암호 – 삼중 DES

MiTM(Meet-in-the-Middle) 공격아이디어

- Forward 방향
  - 평문 P에서 시작 -> 모든 후보 K_1 후보(2^{56}개) 에 대해 암호화
  - $X = E_{K_1}(P)$
  - 테이블에 저장
- Backward 방향
  - 암호문 C에서 시작 -> 모든 후보 K_2 후보 (2^{56}개) 에 대해 복호화
  - $Y = D_{K_2}(C)$
  - Forward 테이블과 비교
- 중간값 비교
  - 올바른 (K_1, K_2)라면 X=y 가 반드시 성립

3.3 블록 암호 – 삼중 DES

- MiTM(Meet-in-the-Middle) 복잡도 분석
  - Forward: 2^{56}연산
  - Backward: 2^{56} 연산
  - 메모리 요구량
    - Forward 테이블 크기 = 2^{56} 항목
    - 항목당 8바이트(중간값) + 7바이트(키) ≈ 15바이트
  - 실제 보안 강도 = 약 57비트
- 평문-암호문 쌍 필요성
  - 한 쌍만 사용:
    - 후보 ≈ 2^{48} -> 너무 많음
  - 두 쌍 이상 사용:
    - 오탐 필터링 -> 유일한 해

3.3 블록 암호 – 삼중 DES

- MiTM(Meet-in-the-Middle) 예시
  - 설정
    - $P = 20, C = 90$
    - 실제 키 : $K_1 = 30, K_2 = 40$
    - 단순 암호 : $E_K(M) = (M + K) mod\ 256$
  - Forward(K_1후보)
    - $K_1 = 30 \rightarrow X = 50$
    - (모든 K_1에 대해 계산 후 테이블 저장)
  - Backward(K_2 후보)
    - $K_2 = 40 \rightarrow \mathrm{Y} = 50$
    - (모든 K_2에 대해 계산 후 테이블 저장)
  - 중간값 일치
    - Forward에서 X = 50, Backward에서 Y = 50
    - 후보 키쌍 (K1= 30, K2 = 40 발견)

단계	입력	연산	출력
Forward	P=20, K1=30	$E_{30}(20) = 50$	X=50
Backward	C=90, K2=40	$D_{40}(90) = 50$	Y=50
비교	X=50 vs Y=50	일치	키쌍=(30,40)

3.3 블록 암호 – 삼중 DES

■ 삼중 DES

- 삼중 DES는 오직 두 개의 키만 사용하며 암호화-암호화-암호화(EEE) 대신, 암호화-복호화-암호화(EDE) 사용
  - $C = E_{K_3}(D_{K_2}\left(E_{K_1}(P)\right))$
- 두 개의 키만 사용하는 이유는 112비트로 충분하고 세 개의 키 사용한다고 보안이 높아지는 것도 아니기 때문임
- EEE 대신에 EDE를 사용하는 이유는 하위 호환(backwards compatibility) 때문

■ 예시

- 평문 $\mathrm{P} = 50$, 키 $K_1 = 30, K_2 = 40, K_3 = 50$
- $E_K(M) = (M + K) mod\ 256, D_K(C) = (C - K) mod\ 256$
- $E_{K_1}(P) = (50 + 30) mod\ 256 = 80$
- $D_{K_2}(80) = (80 - 40) mod\ 256 = 40$
- $E_{K_3}(40) = (40 + 50) mod\ 256 = 90$
- $D_{K_1}(90) = (90 - 30) mod\ 256 = 60$
- $E_{K_2}(40) = (60 + 40) mod\ 256 = 100$
- $D_{K_3}(80) = (100 - 50) mod\ 256 = 50$