2025/07/25, 2025 보안 기초 세미나

암호학과 네트워크 보안

-5장 현대 대칭-키 암호 소개,
8장 현대 대칭-키 암호를 이용한 암호화 기법-

송 민 희(23011716@sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- 보충
- 현대 대칭-키 암호 소개
  - 현대 블록 암호 개요
  - 치환 군으로서의 블록 암호
  - 현대 블록 암호의 구성요소
  - 현대 블록 암호
  - 블록 암호에 대한 공격
  - 현대 스트림 암호
- 현대 대칭-키 암호를 이용한 암호화 기법
  - 현대 블록 암호의 사용
  - 스트림 암호의 사용
  - 키 관리와 키 생성

목 차

- 보충
- 현대 대칭-키 암호 소개
  - 현대 블록 암호 개요
  - 치환 군으로서의 블록 암호
  - 현대 블록 암호의 구성요소
  - 현대 블록 암호
  - 블록 암호에 대한 공격
  - 현대 스트림 암호
- 현대 대칭-키 암호를 이용한 암호화 기법
  - 현대 블록 암호의 사용
  - 스트림 암호의 사용
  - 키 관리와 키 생성

보충

- 현대 블록 암호 개요
  - 특징
    - 어느 경우이던지 n-비트 블록의 각 비트는 0과 1 중 하나의 값을 가질 수 있으므로 n-비트 평문/암호문의 개수는 2^n 개임
    - 대치 구조가 전치 구조 보다 보안성이 우수한 보안 설계 요소로 사용됨
      - 대치 구조 암호는 비선형적으로 암호문의 예측이 어려움
      - 전치 구조 암호는 선형적으로 암호문의 예측이 쉬움

보충

- 현대 블록 암호의 구성요소 (1/3)
  - P-박스 (1/3)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (1/3)
      - 단순 P-박스
        - 보안성
          - 가장 기본적인 P-박스 형태로 복잡한 비트 상호작용은 없으나, 다른 구성요소와 결합하여 높은 보안성을 제공할 수 있음
        - 성능
          - 동일한 수의 비트를 입출력 하므로 비트 이동 연산만 수행함
          - 비교적 효율적인 처리 속도를 제공함

보충

- 현대 블록 암호의 구성요소 (2/3)
  - P-박스 (2/3)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (2/3)
      - 확장 P-박스
        - 보안성
          - 입력 비트보다 많은 출력 비트 생성으로 확산효과를 극대화 함
        - 성능
          - 입출력시 처리할 비트 수가 증가함
          - 더 많은 연산과 메모리 사용으로 처리 속도가 저하됨
      - 축소 P-박스
        - 보안성
          - 입력 비트보다 적은 출력 비트 생성으로 확산 효과가 약화됨
        - 성능
          - 입출력시 처리할 비트 수가 감소함
          - 연산량 감소로 처리 속도가 증가됨

보충

- 현대 블록 암호의 구성요소 (3/3)
  - P-박스 (3/3)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (3/3)
      - 보안 관점에서의 순위
        - 확장 P-box > 단순 P-box > 축소 P-box
        - 확장 P-box는 비트 수 증가로 확산 효과를 극대화 함
        - 축소 P-box는 비트 수 감소로 정보 손실과 확산효과에 제한이 발생함
      - 성능 관점에서의 순위
        - 축소 P-box > 단순 P-box > 확장 P-box
        - 축소 P-box는 처리 할 비트 수가 감소하므로 가장 빠른 성능을 제공함
        - 확장 P-box는 처리할 비트 수가 증가하므로 가장 낮은 성능을 제공함

보충

- 현대 블록 암호의 사용 (1/3)
  - 운영 모드간 비교 (1/3)

운영모드	패딩 필요	IV사용	블록간 관계	병렬 처리	스트림 암호로 변경	동기식 / 비동기식	데이터 단위 크기
ECB	O	X	독립적	암호화, 복호화	X	-	n
CBC	O	O	암호문 의존	복호화	X	-	n
CFB	X	O	암호문 의존	복호화	O	비동기식 스트림 암호	$r \leq n$
OFB	X	O	이전 키스트림 의존	불가	O	동기식 스트림 암호	$r \leq n$
CTR	X	O	독립적	암호화, 복호화	O	동기식 스트림암호	n

보충

- 현대 블록 암호의 사용 (2/3)
  - 운영 모드간 비교 (2/3)
    - 데이터 저장 관점에서 적절한 운영모드 (1/2)
      - CTR 모드
        - 정확한 크기로 데이터 저장이 가능함
          - 패딩이 필요하지 않으므로 효율적이고 정확한 크기를 사용할 수 있음
        - 원하는 블록에 접근이 가능함
          - 각 블록이 고유한 카운터를 이용해 독립적으로 처리됨
        - 처리 속도가 빠름
          - 패딩이 필요하지 않음
          - 스트림 암호로 사용할 수 있음
          - 암호화와 복호화 모두 병렬 처리가 가능함

보충

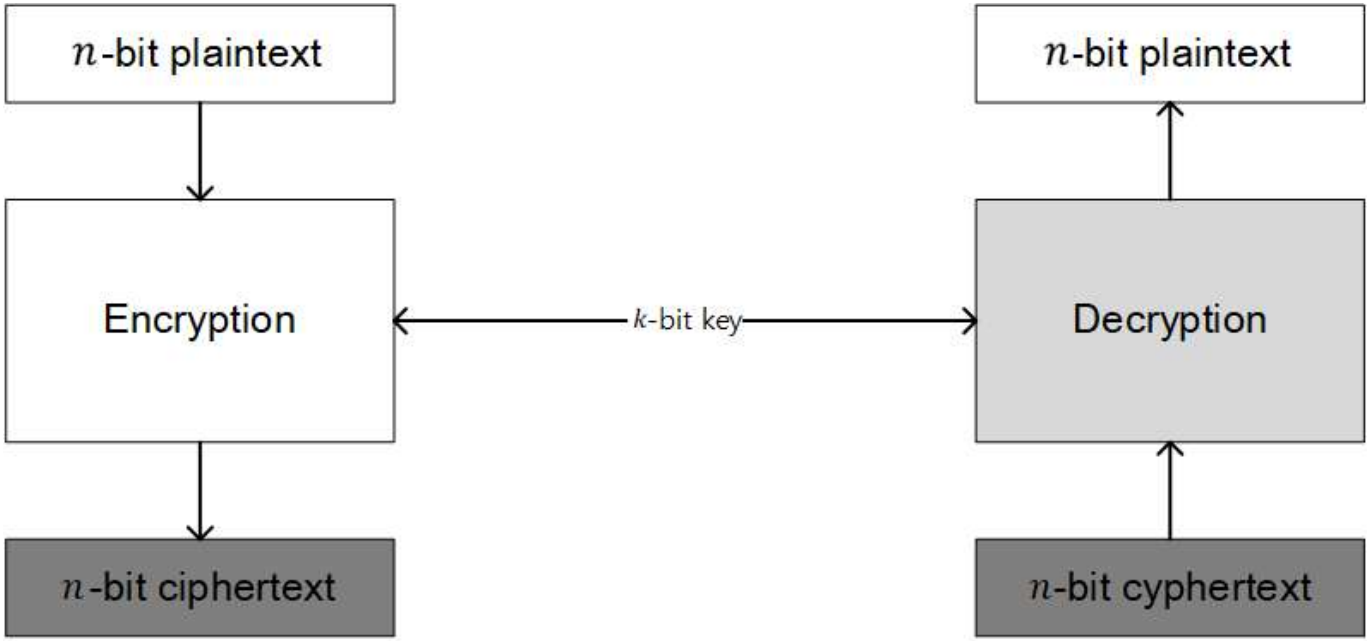
- 현대 블록 암호의 사용 (3/3)
  - 운영 모드간 비교 (3/3)
    - 데이터 전송 관점에서 적절한 운영모드 (2/2)
      - OFB 모드
        - 전송을 위한 실시간 처리에 적합함
          - 스트림 암호로 사용할 수 있음
          - 패딩이 필요하지 않음
        - 오류 전파 없음
          - 암호문에 오류가 생겨도 해당 블록에만 영향을 미침

목 차

- 보충
- 현대 대칭-키 암호 소개
  - 현대 블록 암호 개요
  - 치환 군으로서의 블록 암호
  - 현대 블록 암호의 구성요소
  - 현대 블록 암호
  - 블록 암호에 대한 공격
  - 현대 스트림 암호
- 현대 대칭-키 암호를 이용한 암호화 기법
  - 현대 블록 암호의 사용
  - 스트림 암호의 사용
  - 키 관리와 키 생성

현대 대칭-키 암호 소개

- 현대 블록 암호 개요 (1/4)
  - 정의
    - n-비트 평문 블록을 암호화 하거나 n -비트 암호문 블록을 복호화 하는 것



현대 대칭-키 암호 소개

- 현대 블록 암호 개요 (2/4)
  - 특징
    - k-비트의 키를 사용함
    - 암호화, 복호화 알고리즘 모두 동일한 비밀 키를 사용함
    - n –비트를 기준으로 메시지의 길이를 조정하여 사용함
      - 메시지의 길이가 n-비트 보다 작다면 n-비트를 만들기 위해 덧붙이기(Padding)를 사용함
      - 메시지의 길이가 n-비트 보다 길다면 메시지는 n-비트 블록 단위로 분할하여 사용함
      - 보통 n은 64(2^6), 128(2^7), 256(2^8), 512(2^9) 비트를 사용함
  - 예제 5.1

100개의 문자 8비트 ASCII 코드가 사용되고 64비트 블록 암호를 이용하여 암호화하기를 원한다면, 100개의 문자로 구성된 메시지는 몇 비트가 덧붙이기 되어야하는가?

$$|M| + |pad| = 0 \; mod \; 64 \rightarrow |pad| = -800 \; mod \; 64 \rightarrow 32 \; mod \; 64 \qquad \therefore \; 32\text{비트}$$

현대 대칭-키 암호 소개

- 현대 블록 암호 개요 (3/4)
  - 대치(Substitution) 암호
    - 정의
      - 평문 비트의 값이 임의의 값으로 대체되는 암호
    - 예시
      - 00110000 → 11000110
  - 전치(Transposition) 암호
    - 정의
      - 평문 비트 각각의 값이 재배열되는 암호
    - 예시
      - 00110000 → 10001000

현대 대칭-키 암호 소개

- 현대 블록 암호 개요 (3/4)
  - 특징
    - 어느 경우이던지 n-비트 블록의 각 비트는 0과 1 중 하나의 값을 가질 수 있으므로 n-비트 평문/암호문의 개수는 2^n 개임
    - 대치 구조가 전치 구조 보다 보안성이 우수한 보안 설계 요소로 사용됨
      - 대치 구조 암호는 비선형적으로 암호문의 예측이 어려움
      - 전치 구조 암호는 선형적으로 암호문의 예측이 쉬움

현대 대칭-키 암호 소개

- 현대 블록 암호 개요 (4/4)
  - 예제 5.2

64비트 블록 암호라 가정하자 ($n = 64$). 만약 암호문에서 1의 개수가 10이라면, 공격자가 중간에 가로챈 암호문으로부터 평문을 도출해 내기 위해서 필요한 시행착오(Trial-and-error Test)는 다음 두가지 경우 각각 몇 번인가?

a. 암호가 대치 암호로 설계된 경우
b. 암호가 전치 암호로 설계된 경우

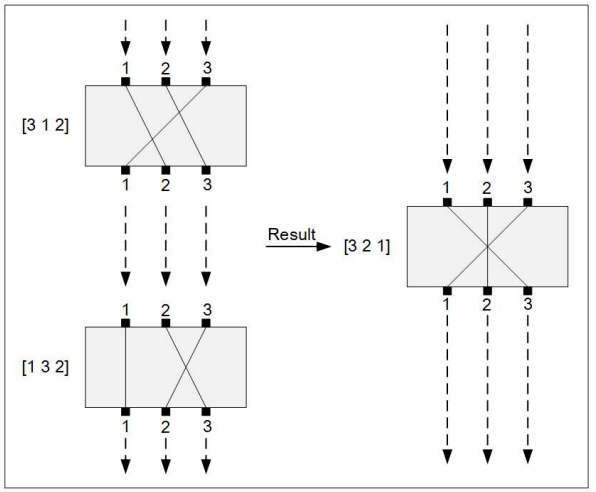
a. 대치 암호의 경우 1의 개수를 알 수 없으므로 공격자는 64비트 블록에 대하여 2^{64}번의 시행착오가 필요함
$\therefore 2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$

b. 전치암호의 경우, 1의 개수가 유지되므로 공격자는 1이 10개인 평문만 조사 필요함
$\therefore \frac{64!}{(10!)(54!)} = 151{,}473{,}214{,}816$

현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 치환 *군(Permutation Group)
    - 모든 치환을 모아놓은 집합
    - 두 치환에 대한 합성으로 정의된 연산들로 만들어진 집합
    - 군의 닫혀 있는 성질을 통해 합성 연산이 암호의 안전성을 강화할 수 있는 지를 확인 할 수 있는 집합
    - 전체 길이 키 암호와 부분 키 암호에서의 군
      - 전체 길이 키 암호가 군 $G = < M, \circ >$이라면, 부분 키 암호는 부분군 $H = < N, \circ >$
        - 단, M이 대응되는 집합이고, $\circ$은 동일한 연산
        - 단, N은 M의 부분 집합이고, $\circ$은 동일한 연산
      - e.g., $2^{64}!$ 개의 대응을 갖는 군으로 2^{56}개의 대응을 갖는 부분군을 구성할 수 없으므로, 56비트 키를 갖는 다중 DES는 군이 아님

*군
네 개의 성질(혹은 공리)를 만족하는 이항 연산이 정의된 원소들의 집합

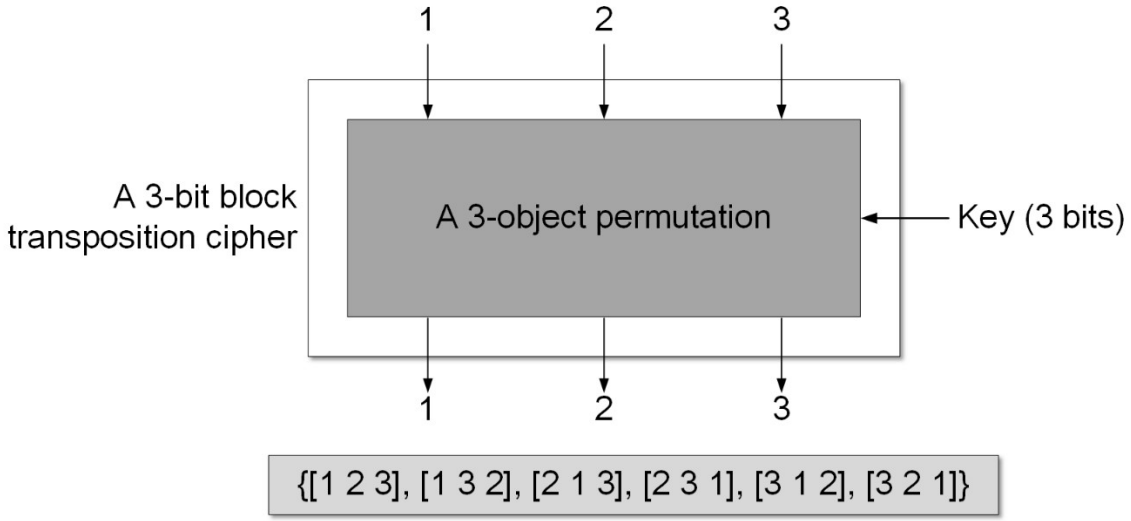


현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (1/6)
    - 정의
      - 입력 값을 출력 값으로 대응시키는 모든 가능한 경우를 키로 대응시키기 위하여 충분한 키 길이를 가지는 키 암호
        - 실제 환경에서는 더 짧은 길이의 키를 사용
    - 종류
      - 전체 길이 키 전치 블록 암호(Full-size Key Transposition Block Ciphers)
      - 전체 길이 키 대치 블록 암호(Full-size Key Substitution Block Ciphers)

현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (2/6)
    - 전체 길이 키 전치 블록 암호(Full-size Key Transposition Block Ciphers) (1/2)
      - 각 비트의 위치를 재배열함
      - n-비트의 블록을 $n!$개의 치환표로 구성된 집합으로 모델화할 수 있음
      - 효율적인 키 길이는 $[\log_2 n!]$ 비트임

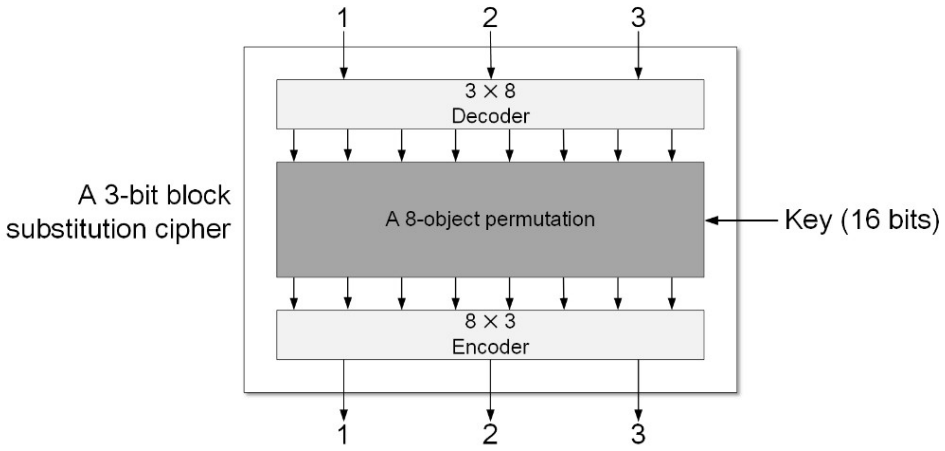


현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (3/6)
    - 전체 길이 키 전치 블록 암호(Full-size Key Transposition Block Ciphers) (2/2)
      - 예제 5.3

블록 길이가 3비트인 3비트 블록 전치 암호의 모델과 치환표의 집합을 보이시오.

- 치환표의 집합은 3!=6의 원소를 가지며, 키 길이는 $\lceil \log_2 6 \rceil = 3$ 임
- 3비트 키는 $2^3 = 8$개의 서로 다른 대응을 정의할 수 있지만, 8개 중 6개의 키만 사용함



{[1 2 3 4 5 6 7 8], [1 2 3 4 5 6 7 8], . . .}

현대 대칭-키 암호 소개

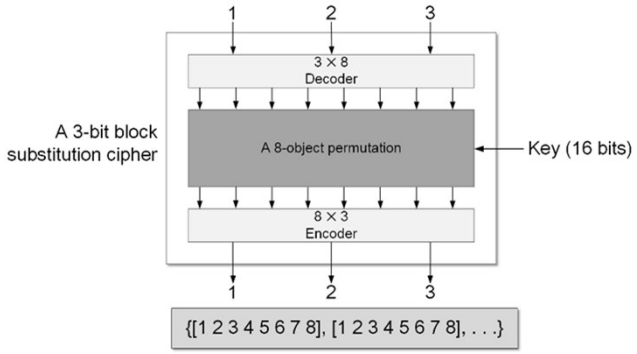
- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (4/6)
    - 전체 길이 키 대치 블록 암호(Full-size Key Substitution Block Ciphers) (1/2)
      - 각 비트의 값을 0 또는 1로 대체함
      - 대치암호 자체로는 치환표로 모델화할 수 없음
      - 디코딩(Decoding)과 인코딩(Encoding)을 통해 치환표로 모델화할 수 있음
        - 모델화 과정
          1. 입력 값에 전치를 적용할 수 있도록 디코딩
          2. 디코딩 된 값에 전치 적용
          3. 전치 적용된 값을 인코딩하여 원래 대치와 같은 효과부여

현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (5/6)
    - 전체 길이 키 대치 블록 암호(Full-size Key Substitution Block Ciphers) (2/2)
    - 예제 5.4

블록 길이가 3비트인 3비트 블록 대치 암호의 모델과 치환표의 집합을 보이시오.

- 세 비트 입력 평문은 0과 7 사이의 정수 값을 가짐
- 입력 값은 한 비트만 1의 값을 갖는 8비트 스트링으로 디코딩 됨
  e.g., 000은 00000001로 디코딩, 101는 00100000로 디코딩
- 3비트 블록 대치 암호의 치환표 집합의 원소의 개수는
  $8! = 40{,}320$ 개임
- 3비트 블록 대치 암호의 키 길이는 $\lceil \log_2 40{,}320 \rceil = 16$비트로,
  16비트 키는 $2^{16} = 65{,}536$개의 서로 다른 대응을 정의할 수 있으나,
  40,320개의 키만 사용함



현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 전체 길이 키 암호(Full-size Key Cipher) (6/6)
    - 전체 길이 키 n-비트 전치/대치 블록 암호는 치환으로 모델화 가능함
    - 키 길이는 전치 블록 암호와 대치 블록 암호 서로 다름
      - 전치암호
        - 블록 크기 : n-비트
        - 가능한 전치 수 : $n!$
        - 필요한 키 길이 : $\lceil \log_2 n! \rceil$ 비트
      - 대치암호
        - 입력 조합 수 : 2^n개
        - 가능한 대치 수 : $(2^n)!$
        - 필요한 키 길이 : $\lceil \log_2 (2^n)! \rceil$ 비트
    - 전체 길이 키는 길이가 너무 커 실제 암호에서는 사용할 수 없음

현대 대칭-키 암호 소개

- 치환 군으로서의 블록 암호
  - 부분 길이 키 암호(Reduced-key Cipher)
    - 전체 가능한 치환 중 일부만을 키로 사용하는 블록 암호
    - 동일한 암호의 다단계(Multi-stage) 버전의 안전성 증명
      - e.g., DES(Data Encryption Standard)의 사용
        - 이론적 최대 키 길이 : $\log_2(2^{64}!) \approx 2^{70}$비트
        - 실제 DES 키 길이 : 2^{56}개의 대응만을 사용하는 56비트
        - 2^{64}!대응을 갖는 군으로 2^{56}개의 대응을 갖는 부분군을 구성 불가능
          ∴ 56비트 키를 갖는 다중 DES는 군이 아님
  - 실제 키와 이론 키 길이 차이의 중요성
    - 이론 키의 길이가 충분히 크더라도 실제 키의 길이가 작아지면 공격 표면이 높아져 공격자가 암호를 알아낼 가능성이 증가함
    - 키 길이가 줄어들면 성능은 높아지나 보안성이 낮아지는 Trade-Off 현상이 발생함

현대 대칭-키 암호 소개

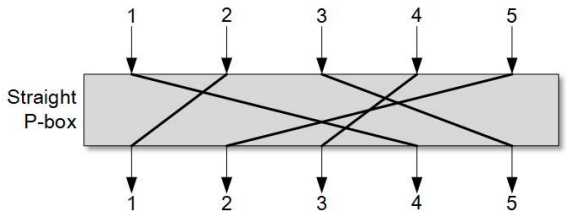
- 치환 군으로서의 블록 암호
  - 키 없는 암호 알고리즘
    - 키를 입력으로 받지 않고, 고정된 연산으로만 구성된 암호 알고리즘
    - 단독으로는 암호화에 사용되지 않으며, 키가 있는 암호 알고리즘의 구성 요소로 사용
    - 키가 없는 전치 암호
      - 입력 비트들의 위치를 바꾸는 위치 기반 사상
      - 하드웨어의 경우 고정 회로로 내장
      - 소프트웨어의 경우 테이블 등으로 표현 가능
      - e.g., P-박스
    - 키가 없는 대치 암호
      - 입력 값에 대해 사전에 정의된 출력 값으로 대응시키는 치환 사상
      - 테이블, 수학 함수 등으로 대응 방식 정의 가능
      - e.g., S-박스

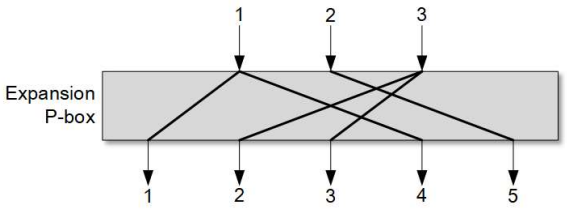
현대 대칭-키 암호 소개

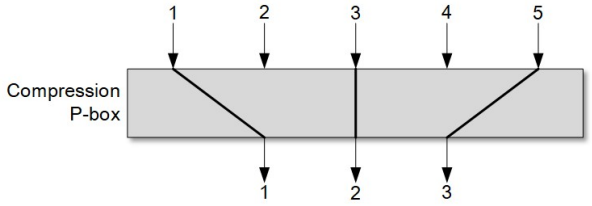
- 현대 블록 암호의 구성요소
  - 현대 블록 암호의 특징
    - 부분적 대응만을 허용하는 키와 다양한 구성요소를 결합하여 만듦
  - 구성요소
    - P-박스
    - S-박스
    - 배타적 논리합
    - 순환 이동
    - 스왑
    - 분할과 결합

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스(P-box, Permutation Box) (1/6)
    - n-비트를 입력 받고 m-비트를 출력하는 치환 함수임
    - 키가 존재하지 않으며, 대응은 보통 사전에 정의 됨
    - P-박스의 종류
      - 단순(Straight) P-박스
      - 확장(Expansion) P-박스
      - 축소(Compression) P-박스

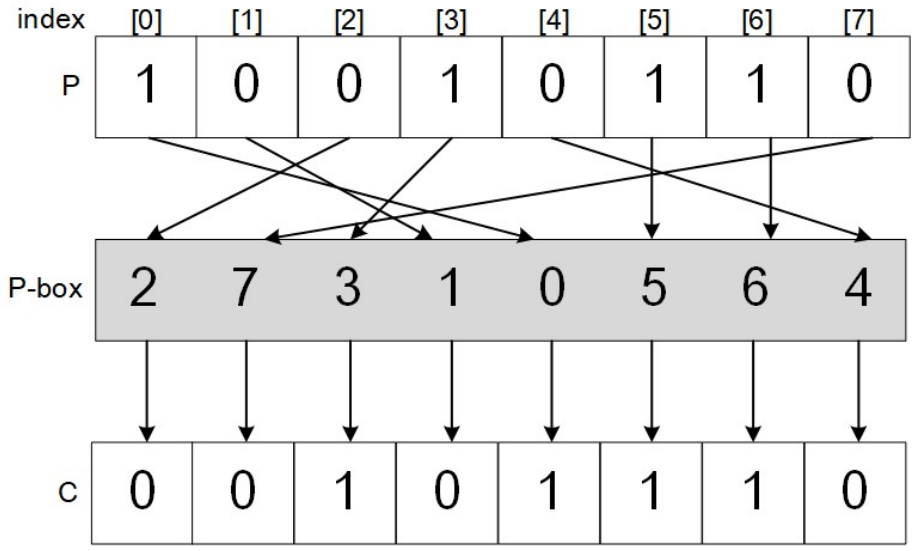






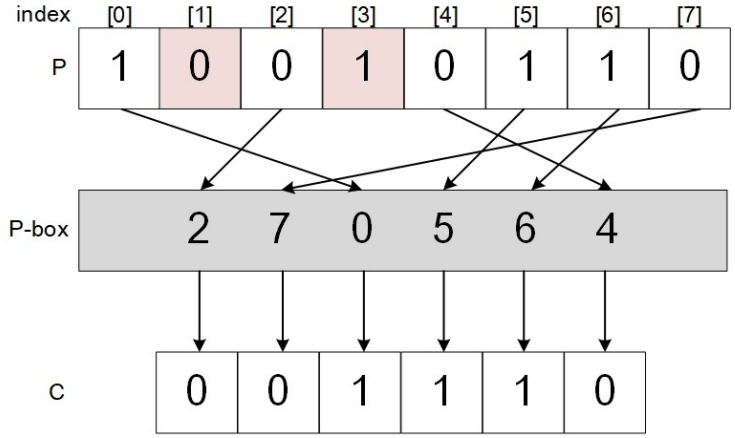
현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (2/6)
    - 단순 P-박스
      - n-비트를 입력 받고 n-비트를 출력하는 치환 함수
      - $n!$개의 대응이 존재함
      - e.g., 8비트 단순 P-박스의 작동 방식



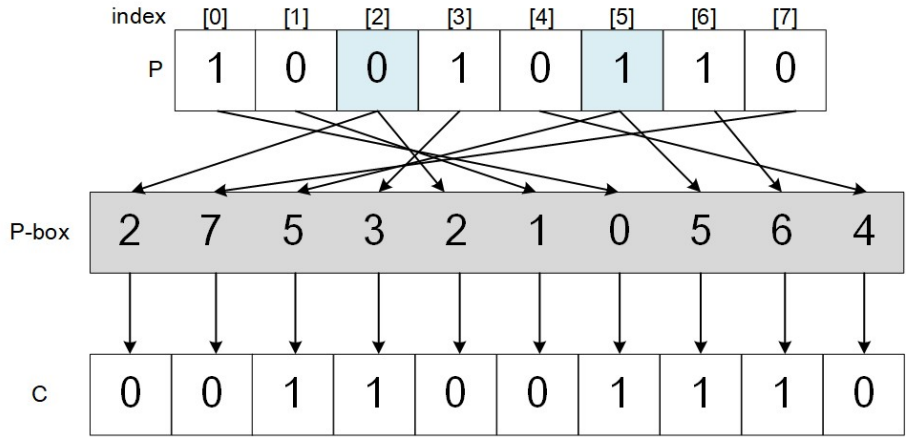
현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (3/6)
    - 축소 P-박스
      - n-비트를 입력 받고 m-비트를 출력하는 치환 함수 $(n > m)$
      - 치환 표는 m개의 성분을 가지며 $1 \sim n$까지의 값을 가짐
        - 특정 비트는 소실되어 출력되지 않음
        - 다음 단계를 위한 비트의 수를 줄이기 위해 사용함
      - e.g., 8*6 축소 P-박스의 작동 방식



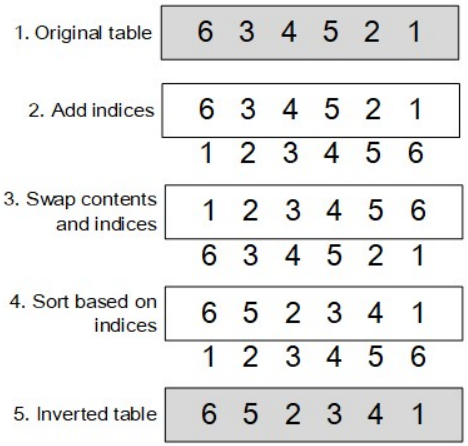
현대 대칭-키 암호 소개

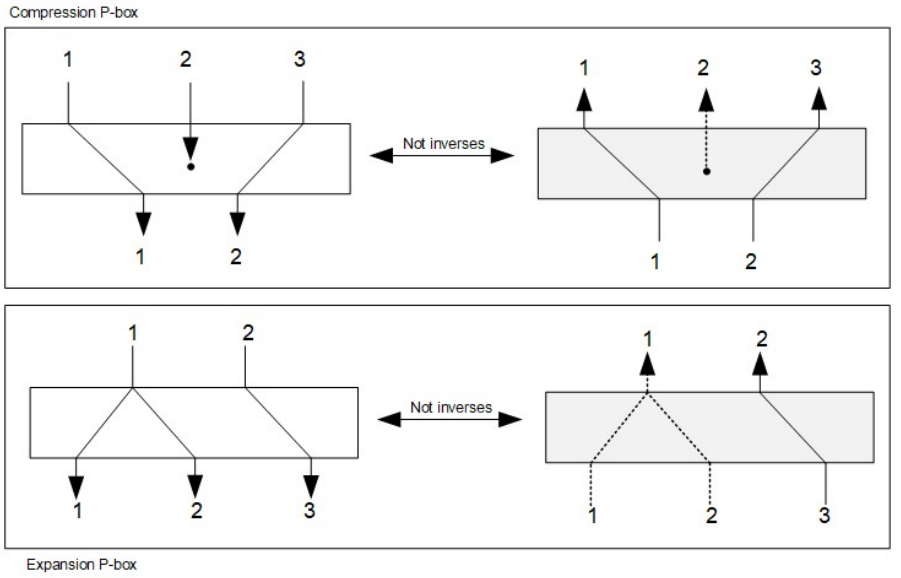
- 현대 블록 암호의 구성요소
  - P-박스 (4/6)
    - 확장 P-박스
      - n-비트를 입력 받고 m-비트를 출력하는 치환 함수 $(n < m)$
      - 치환표는 m개의 성분을 가지며 $m - n$개의 성분이 반복적으로 사용함
        - 하나의 입력 비트는 하나 이상의 출력 비트에 대응 됨
        - 다음 단계를 위한 비트의 수를 늘리기 위해 사용함
      - e.g., 8*10 확장 P-박스의 작동 방식



현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (5/6)
    - 역함수가 존재하는 경우
      - 서로 간의 역원이 되는 경우
      - e.g., 단순 P-박스
    - 역함수가 존재하지 않는 경우
      - 소실, 중복 된 비트를 찾는 단서가 없는 경우
      - e.g., 축소 P-박스, 확장 P-박스
    - 치환표의 역 구성





현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (6/8)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (1/3)
      - 단순 P-박스
        - 보안성
          - 가장 기본적인 P-박스 형태로 복잡한 비트 상호작용은 없으나, 다른 구성요소와 결합하여 높은 보안성을 제공할 수 있음
        - 성능
          - 동일한 수의 비트를 입출력 하므로 비트 이동 연산만 수행함
          - 비교적 효율적인 처리 속도를 제공함

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (7/8)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (2/3)
      - 확장 P-박스
        - 보안성
          - 입력 비트보다 많은 출력 비트 생성으로 확산효과 극대화할 수 있음
        - 성능
          - 입출력시 처리할 비트 수가 증가함
          - 더 많은 연산과 메모리 사용으로 처리 속도가 저하됨
      - 축소 P-박스
        - 보안성
          - 입력 비트보다 적은 출력 비트 생성으로 확산 효과가 약화됨
        - 성능
          - 입출력시 처리할 비트 수가 감소함
          - 연산량 감소로 처리 속도가 증가됨

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - P-박스 (8/8)
    - P박스의 종류에 대한 Trade-off 관점에서의 비교 (3/3)
      - 보안 관점에서의 순위
        - 확장 P-box > 단순 P-box > 축소 P-box
        - 확장 P-box는 비트 수 증가로 확산 효과를 극대화함
        - 축소 P-box는 비트 수 감소로 정보 손실과 확산효과에 제한이 발생함
      - 성능 관점에서의 순위
        - 축소 P-box > 단순 P-box > 확장 P-box
        - 축소 P-box는 처리 할 비트 수가 감소하므로 가장 빠른 성능을 제공함
        - 확장 P-box는 처리할 비트 수가 증가하므로 가장 낮은 성능을 제공함

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - S-박스(S-box, Substitution Box) (1/5)
    - n-비트를 입력 받고 m-비트를 출력하는 치환 함수
      - n과 m의 크기가 같을 필요는 없음
      - 역함수가 존재하는 경우에는 $n = m$임
    - 일반적으로 키가 없는 S-박스를 사용하며, 키가 있는 경우도 있음
    - S-박스 종류
      - 선형(Linear) S-box
        - 입력 비트들의 XOR 연산만으로 구성된 함수임
        - 단순하고 예측 가능함
        - 암호학적으로 보안이 취약함
      - 비선형(Nonlinear) S-box
        - 입력 비트들의 AND, 곱, 역함수 등의 비선형 연산으로 구성된 함수임
        - 구조가 복잡하며 분석 어려움
        - 암호학적으로 보안 강함

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - S-박스(S-box, Substitution Box) (2/5)
    - 예제 5.5

3비트를 입력받아 2비트를 출력하는 S-박스가 다음을 만족한다.

$$y_1 = x_1 \oplus x_2 \oplus x_3 \qquad y_2 = x_1$$

- $a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1} = 1$ 이고, $a_{2,2} = a_{2,3} = 0$이기 때문에, 위의 S-박스는 선형임
- 아래의 행렬식처럼 표현할 수 있음

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - S-박스(S-box, Substitution Box) (3/5)
    - 예제 5.6

3비트를 입력받아 2비트를 출력하는 S-박스가 다음을 만족한다.

$$y_1 = (x_1)^3 + x_2 \qquad y_2 = (x_1)^2 + x_1x_2 + x_3$$

- 해당 예제에서 곱셈과 덧셈은 $^*GF(2)$에서 정의됨
- 입력 값과 출력 값 사이에 선형 관계식이 존재하지 않으므로 위의 S-박스는 비선형임

$^*GF(2)$
두 원소 {0, 1}만 가지며, 덧셈은 XOR,
곱셈은 AND로 정의되는 이진 갈로아체

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - S-박스(S-box, Substitution Box) (4/5)
    - 예제 5.7

다음의 표는 4×2 길이의 S-박스에 대한 입출력 관계식을 정의한다. 입력 값의 왼쪽 최상위 비트는 행을 정의하고 입력 값의 오른쪽 하위 두 비트는 열을 정의한다. 두 개의 출력 비트는 선택된 행과 열이 교차되는 성분의 값이다.

Leftmost bit

	00	01	10	11
0	00	10	01	11
1	10	00	11	01

Rightmost bit

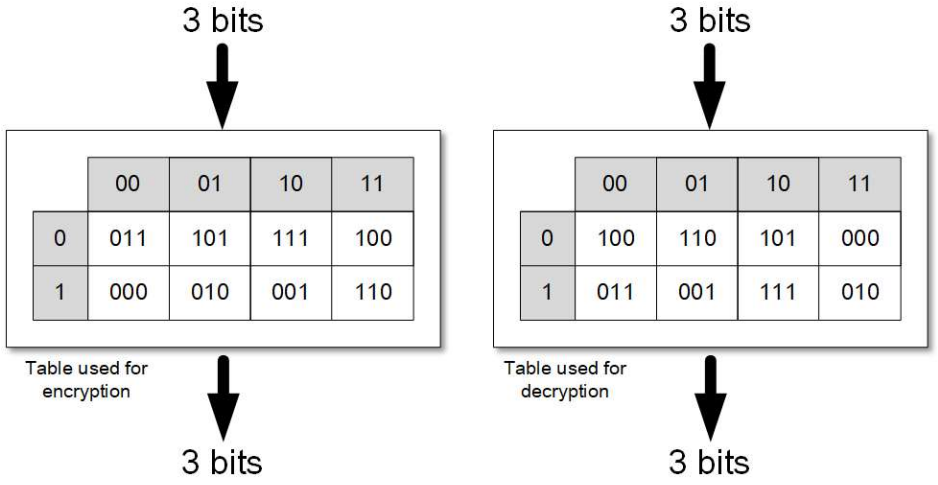
Output bits

- 입력이 010 이면 01을 출력함
- 입력이 101 이면 00을 출력함

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - S-박스(S-box, Substitution box) (5/5)
    - 예제 5.8

아래 그림은 역함수가 존재하는 S-박스의 예제를 보여준다. 표 중 하나는 암호 알고리즘에서 사용되며, 다른 하나는 복호 알고리즘에서 사용된다. 각 표에서 출력 값의 왼쪽 최상위 비트는 행을 정의하고 다음 두 비트는 열을 정의한다. 출력은 입력 값의 행과 열이 교차되는 값이다.



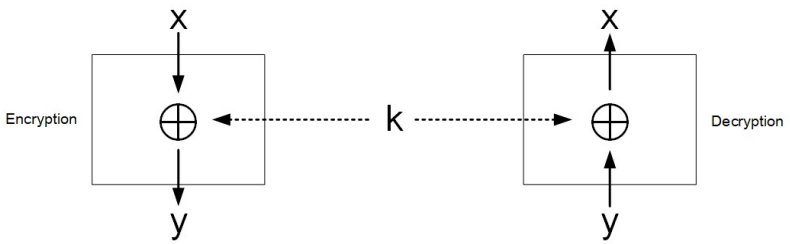
- 두 표는 서로 역함수 관계임

현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 배타적 논리합(Exclusive-Or) (1/2)
    - 성질
      - 닫힘
        - 두 개의 n-비트 워드의 배타적 논리합의 결과가 또 다른 n-비트 워드
      - 결합 법칙
        - $x \oplus (y \oplus \mathrm{z}) \leftrightarrow (x \oplus y) \oplus z$
      - 교환 법칙
        - $x \oplus y \leftrightarrow y \oplus x$
      - 항등원의 존재성
        - $x \oplus (00 \cdots 0) = x$
      - 역원의 존재성
        - $x \oplus x' = (00 \cdots 0)$

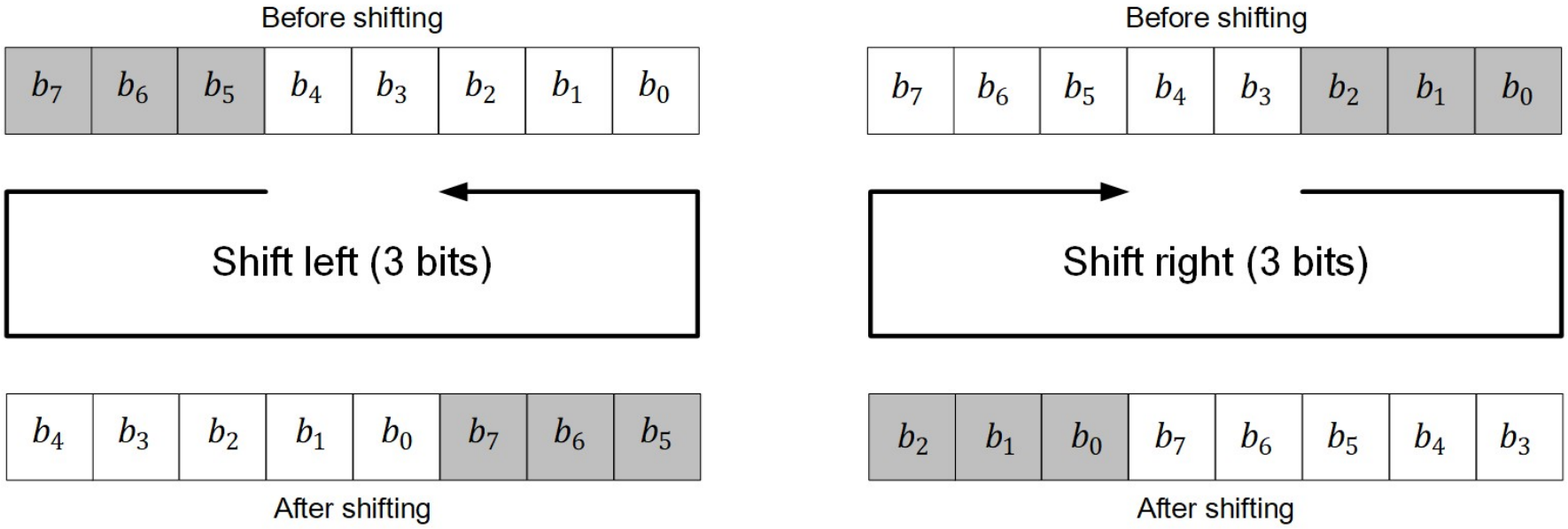
현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 배타적 논리합(Exclusive-Or) (2/2)
    - 보수(Complement)
      - 한 워드에서 각각의 비트를 반대 값으로 바꾸는 단일 연산
      - 성질
        - $x \oplus' x = (11 \cdots 1)$
        - $x' \oplus (11 \cdots 1) = x$
    - 역(Inverse)
      - 암호 알고리즘의 구성요소가 단일연산인 경우, 각 구성요소의 역 존재
        - 하나의 입력과 하나의 출력을 가지기 때문
      - 배타적 논리합은 이진 연산이므로 두 입력 중 하나의 입력이 고정되어야 역의 존재 유의미
        - e.g., $x \oplus k = y$, $y \oplus k = x$ (k는 고정 값)



현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 순환 이동 연산(Circular Shift Operation) (1/2)
    - n-비트 워드를 왼쪽이나 오른쪽으로 k-비트만큼 이동시키는 이항 연산
    - 최상위 k-비트는 반대쪽으로 밀려 최하위 k-비트로 전환됨
    - e.g., $n = 8, k = 3$인 경우 순환 이동 연산

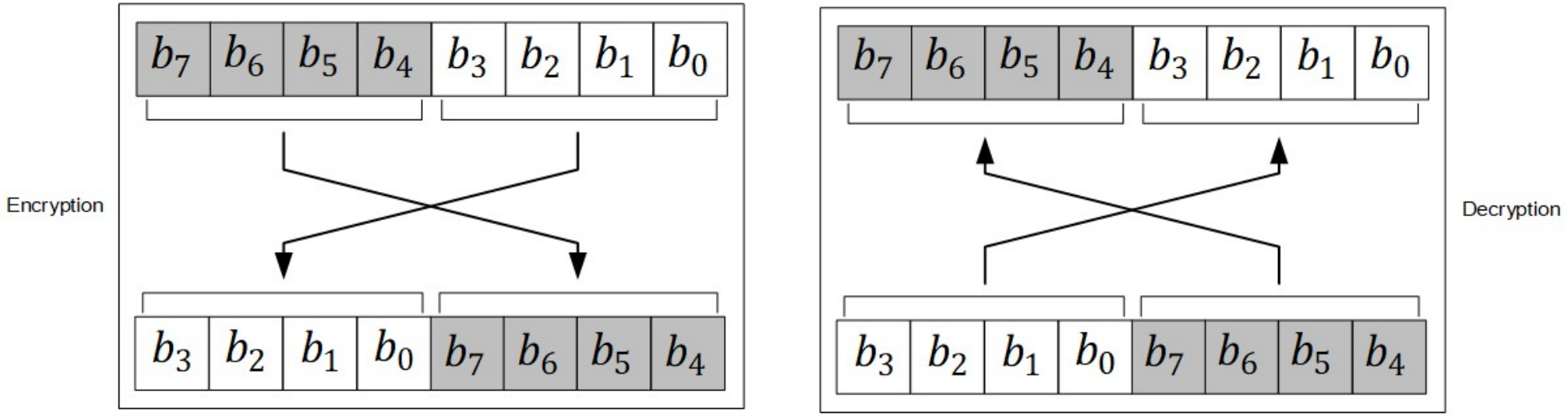


현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 순환 이동 연산(Circular Shift Operation) (2/2)
    - 특징
      - 보통 키가 없으며 k의 값은 고정되고 사전에 정의됨
      - 왼쪽 순환이동 연산은 오른쪽 순환 이동 연산의 역임
    - 성질
      - 이동되는 양은 모듈로 n임
        - $k = 0$이거나 $k = n$이면 자기 자신과 동일함
      - 합성 연산에서 순환 이동 연산은 군임
        - 한 번 이상의 워드를 이동하는 것은 오직 한번만 이동하는 것과 같음

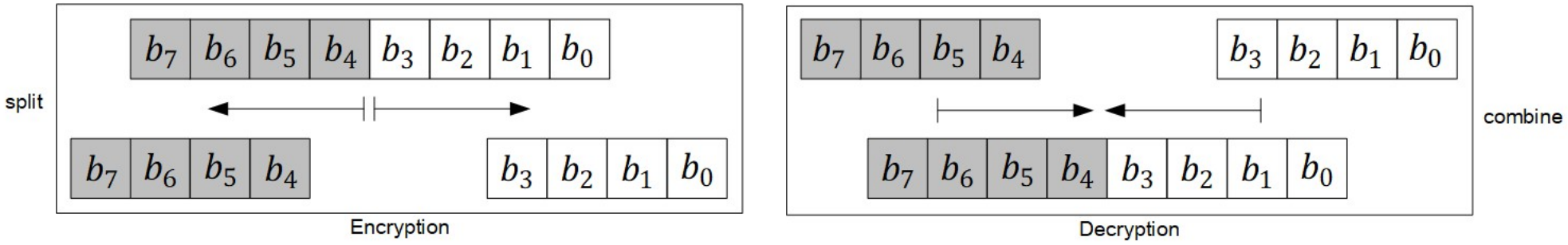
현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 스왑 연산(Swap Operation)
    - $k = n/2$ 인 이동 연산
    - n이 짝수일 때만 연산이 옳게 수행됨
    - 자기 자신을 역으로 가짐
    - e.g., 8비트 워드에서의 스왑연산



현대 대칭-키 암호 소개

- 현대 블록 암호의 구성요소
  - 분할과 결합
    - 분할 연산(Split)
      - 두개의 동일한 길이의 워드를 생성하기 위해 n-비트 워드의 중앙을 분할
    - 결합 연산(Combine)
      - 두 개의 동일한 길이의 워드를 n-비트 워드를 연접(Concatenate)
    - 특징
      - 두 연산은 서로 역관계임
      - 서로를 소거시키기 위한 한 쌍으로 사용함
      - e.g., $n = 8$인 경우의 연산



현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (1/14)
    - 정의
      - 대치, 치환 등의 구성요소를 결합한 복합적인 암호
    - 확산과 혼돈
      - 확산(Diffusion)
        - 암호문과 평문 사이의 관계를 숨기기 위해 사용함
        - 암호문에 대한 통계 테스트를 통하여 평문을 찾기 어렵게 하기 위해 사용함
        - 평문의 단일 비트가 바뀌면, 암호문의 특정 비트나 모든 비트가 변경될 수 있음
      - 혼돈(Confusion)
        - 암호문과 키의 관계를 숨기기 위해 사용함
        - 암호문을 이용하여 키를 찾기 어렵게 하기 위해 사용함
        - 키의 단일 비트가 변하면 암호문의 거의 모든 비트가 변경 됨

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (2/14)
    - 확산과 혼돈의 결합 필요성
      - 확산만 적용 시
        - 암호문의 패턴은 숨겨지지만 키와의 관계가 선형으로 드러남
        - 선형/차분 분석에 취약함
      - 혼돈만 적용 시
        - 평문의 패턴이 암호문에 반영되어 통계적 구조가 드러남
        - 통계적 분석(빈도 분석)에 취약함
      - 확산과 혼돈 모두 적용 시
        - 평문과 암호문의 통계적 구조와 키 관계를 모두 복잡하게 만듦
        - 공격자가 암호문으로부터 평문과 키 모두 추정하기 어려움

현대 대칭-키 암호 소개

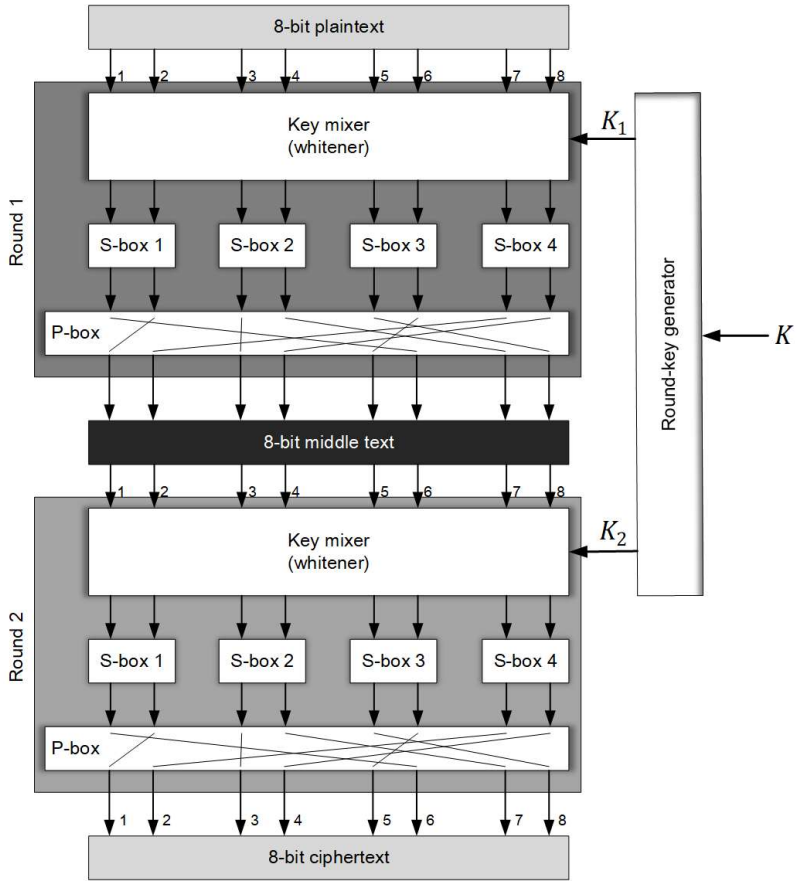
- 현대 블록 암호
  - 합성 암호(Product Cipher) (3/14)
    - 라운드(Round)
      - 합성암호가 반복적으로 사용되는 단계
      - 각 라운드에 사용될 서로 다른 키를 생성하기 위하여 키 스케줄(키 생성 알고리즘)을 사용 함
      - 중간 상태 값(Middle Text) : 각 라운드를 수행하는 중간에 변화되는 상태의 값

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (4/14)
    - 2라운드 합성 암호
      - 라운드 수행 과정
        1. 8비트 문자열은 문자열의 정보를 숨기기 위하여 키와 섞임
           - 일반적으로 8비트 키와 8비트 문자열의 배타적 논리합이 수행됨
        2. 1의 결과 값은 2비트씩 분할되고, 각각의 2비트 값은 4개의 S-박스로 입력됨
           - 이 과정에서 각각의 2비트 값은 S-박스의 구조에 따라 변화함
        3. 각각의 비트를 치환하기 위해 S-박스의 출력 값은 P-박스를 통과함
           - 다음 라운드에서 각 박스는 서로 다른 입력 값을 받음

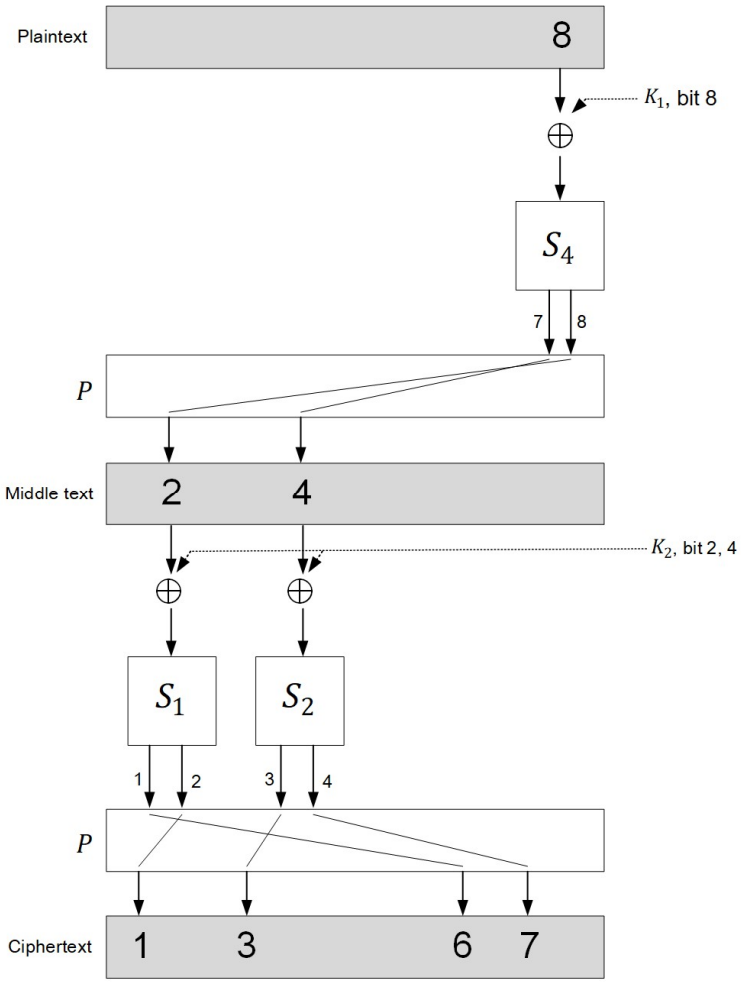
현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (5/14)
    - 2라운드 합성 암호
      - 확산
        - 하나의 비트값이 최종적으로 여러 개의 비트에 영향을 미침
          - e.g., 8번째 비트값은 1,3,6,7 번째 비트에 영향을 미침



현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (6/14)
    - 2라운드 합성 암호
      - 혼돈
        - 여러 개의 비트가 키의 여러 비트에 의해 영향을 받음
          - e.g., 1,3,6,7번째 비트는 키에서 3개의 비트에 의해 영향을 받음



현대 대칭-키 암호 소개

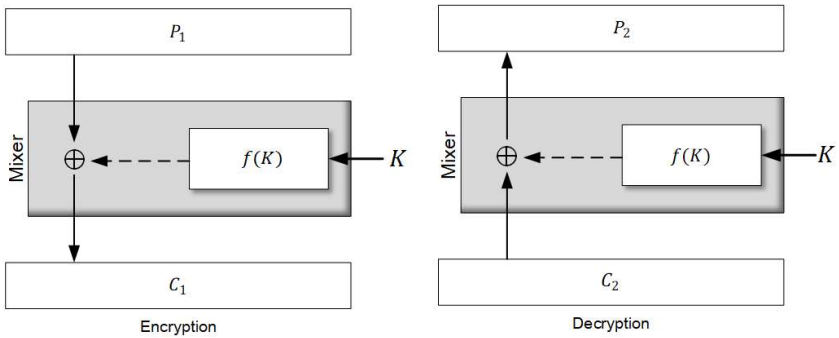
- 현대 블록 암호
  - 합성 암호(Product Cipher) (7/14)
    - 실제 환경에서 사용하는 암호의 특성
      - 확산과 혼돈을 향상시키기 위하여 실제 암호는 더 크고 많은 구성요소를 사용함
      - 사용한 구성요소의 개수와 라운드 수가 증가할 수록 암호문과 키의 관계를 찾기 어려움

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (8/14)
    - 종류
      - Feistel 암호
        - 입력 블록을 좌/우로 분할하고, 좌우를 교환하며 한 쪽은 암호 함수 처리 후 다른 쪽과 XOR 연산하는 암호
        - 역함수가 존재하는 구성요소와 역함수가 존재하지 않는 구성요소를 모두 사용함
        - e.g., DES 암호
      - non-Feistel 암호
        - 입력 전체 블록에 대해 순차적이고 역함수가 존재하는 연산을 적용하는 암호
        - 역함수가 존재하는 구성 요소 만을 사용함
        - e.g., AES 암호

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (9/14)
    - Feistel 암호
      - 구성요소
        - 자기 자신을 역으로 갖는 것
        - 역함수가 존재하는 것
        - 역함수가 존재하지 않는 것
      - 첫번째 사상
        - 입력받은 키 K와 역함수가 존재하지 않는 함수 $f(K)$에 대해 $f(K)$와 평문 간 배타적 논리합을 수행함
        - 배타적 논리합 결과를 암호문으로 사용함



현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (10/14)
    - Feistel 암호
      - 혼합기(Mixer)
        - 키 K에 대하여 역함수가 존재하지 않는 함수$f(K)$와 배타적 논리합 연산의 결합
        - 자기 자신을 역함수로 가짐
          - 배타적 논리합 연산의 두 가지 성질(역의 존재성, 항등원의 존재성)을 사용함
        - 암호화 : $C_1 = P_1 \oplus f(K)$
        - 복호화 : $P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = \mathrm{P}_1 \oplus f(K) \oplus f(K) = P_1 \oplus (00 \ldots 0) = P_1$

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (11/14)
    - Feistel 암호
      - 예제 5.9

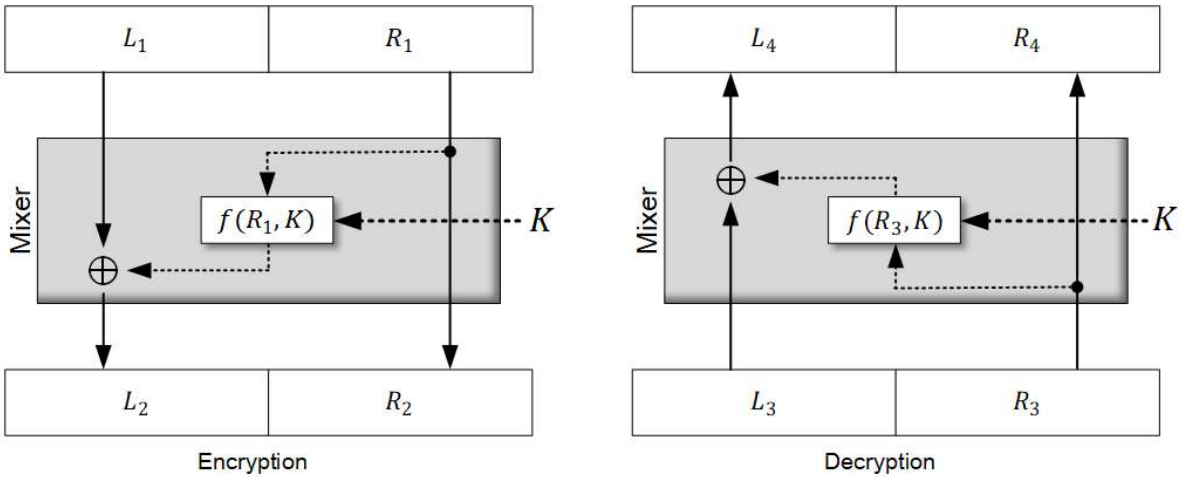
평문과 암호문의 길이가 4비트 이고 키의 길이가 3비트라 하자.
함수는 키의 첫 번째와 세 번째 비트를 입력 받는다.
두 비트는 10진수로 인식하고, 제곱한 뒤, 그 결과 값을 4비트 2진수로 출력한다.
만약 평문이 0111이고 키가 101일 때 암호화와 복호화의 결과 값을 보이시오.

키의 1,3 번째 비트를 통해 2진수로 11, 10진수로 3의 값을 얻고 제곱한 값은 9임
9는 4비트 2진수로 1001임
- 암호화 : $C = P \oplus f(K) = 0111 \oplus 1001 = 1110$
- 복호화 : $P = C \oplus f(K) = 1110 \oplus 1001 = 0111$

$\therefore f(K)$는 역함수가 존재하지 않지만, 혼합기는 자기 자신을 역함수로 가짐

현대 대칭-키 암호 소개

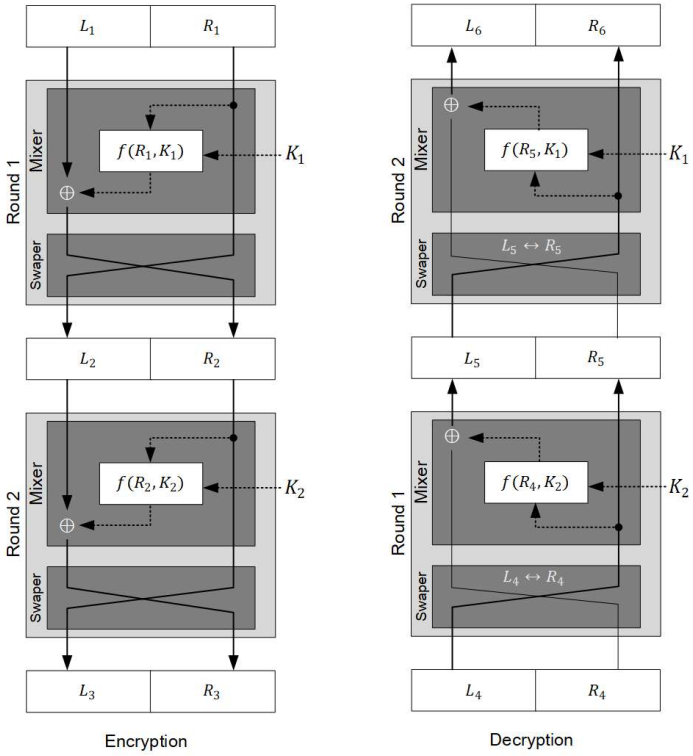
- 현대 블록 암호
  - 합성 암호(Product Cipher) (12/14)
    - Feistel 암호
      - 향상(Improvement)
        - 입력 값으로 키뿐만 아니라 평문의 일부분도 함께 사용함
        - 키를 사용하는 요소와 사용하지 않는 요소를 복합적으로 사용함
        - 평문과 암호문을 두개의 동일한 길이의 왼쪽(L)과 오른쪽(R)로 분리함
          - R은 함수에 입력, L은 함수의 출력값과 배타적 논리합 연산을 수행함



- $R_4 = R_3 = R_2 = R_1$
  $L_4 = L_3 \oplus f(R_3, K) = L_2 \oplus f(R_2, K) = L_1 \oplus f(R_1, K) \oplus f(R_1, K) = L_1$

현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (13/14)
    - Feistel 암호
      - 최종 구조
        - 평문의 오른쪽 절반은 변화 없이 암호문에 노출되어 취약함
        - 개선 방향
          - 라운드 수의 증가로 확산을 강화할 수 있음
          - 각 라운드에 스와퍼(swapper)를 추가하여 특정 부분의 고정 노출을 방지할 수 있음
        - 두 라운드를 갖는 feistel 암호의 최종 구조
          - Round 1
            $L_2 = R_1,\ R_2 = L_1 \oplus f(R_1, K_1)$
          - Swap
            $L_3 = R_2\ ,\qquad R_3 = L_2$
          - Round 2
            $L_4 = R_3,\ \ R_4 = L3 \oplus f(R_3, K_2)$



현대 대칭-키 암호 소개

- 현대 블록 암호
  - 합성 암호(Product Cipher) (14/14)
    - non-Feistel 암호
      - 역함수가 존재하는 요소만을 사용함
        - e.g., 동일한 입/출력 개수를 가지는 S-박스, 단순 P-박스
      - 각 요소가 역이 존재하므로 각 라운드가 역이 존재함
      - 복호 알고리즘은 암호 알고리즘의 역순으로 수행됨

현대 대칭-키 암호 소개

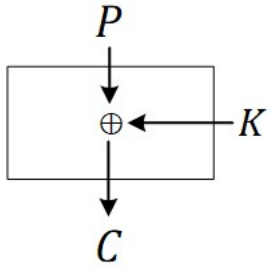
- 블록 암호에 대한 공격 (1/12)
  - 차분 분석(Differential Cryptanalysis)
    - 서로 다른 두 평문 쌍의 입력 차이(ΔP)와 출력 차이(ΔC)의 관계를 분석하여 키 정보를 추정하는 방법
    - 선택 평문 공격에서 사용되는 분석 기법
  - 선형 분석(Linear Cryptanalysis)
    - 평문, 암호문, 키 사이의 관계를 근사하는 선형식을 찾아, 선형식이 만족되는 확률적 편차를 기반으로 키 정보를 추정하는 방법
    - 기지 평문 공격에서 사용되는 분석 기법

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (2/12)
  - 차분 분석(Differential Cryptanalysis)
    - 배타적 논리합(XOR) 연산
      - 두 비트간의 차이를 나타내는 연산
      - 두 비트가 같으면 0, 다르면 1을 출력
    - 평문 차분 (Plaintext Difference)
      - 두 개의 입력 평문 간의 차이를 XOR을 통해 나타내는 이진 벡터
      - $\Delta P = P_1 \oplus P_2$
    - 암호문 차분 (Ciphertext Difference)
      - 두 개의 출력 암호문 간의 차이를 XOR을 통해 나타내는 이진 벡터
      - $\Delta C = C_1 \oplus C_2$

현대 대칭-키 암호 소개

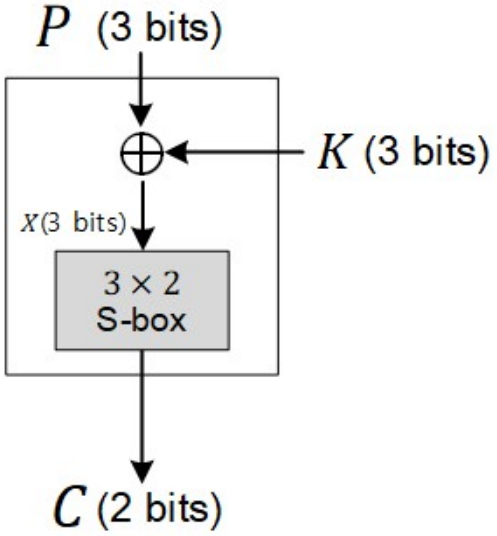
- 블록 암호에 대한 공격 (3/12)
  - 차분 분석(Differential Cryptanalysis) – 과정
    1. 알고리즘 분석 (1/3)
       - 선택 평문 공격 이용 전, 평문에 대한 특정 정보를 수집하기 위한 과정
       - 특정 암호는 암호 키를 모르더라도 알고리즘 내에 평문의 차분과 암호문의 차분 사이의 관계를 찾을 수 있는 구조적 취약점이 존재함
       - e.g., 단순 암호 구조를 통한 차분 분석 예시



- $C_1 = P_1 \oplus K\ ,\quad C_2 = P_2 \oplus K \rightarrow C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$
- 암호구조가 단순한 경우 입력 차분과 출력 차분 간에 직접적이고 예측 가능한 관계를 발견할 수 있음

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (4/12)
  - 차분 분석(Differential Cryptanalysis) - 과정
    1. 알고리즘 분석 (2/3)
       - e.g., S-박스 구조에서의 차분 분석 예시



X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

S-box table

- S-박스가 추가되어 평문 차분과 암호문 차분 사이의 명백한 관계를 찾기 어려움
- $X_1 \oplus X_2 = P_1 \oplus P_2$를 만족하므로 확률적 관계를 알아낼 수 있음
- 이 관계를 기반으로 S-박스의 입출력 표와 차분 분포표를 작성할 수 있음

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (5/12)
  - 차분 분석(Differential Cryptanalysis) - 과정
    1. 알고리즘 분석 (3/3)
       - e.g., S-박스 구조에서의 차분 분석 예시

$C_1 \oplus C_2$

$P_1 \oplus P_2$	00	01	10	11
000	8			
001	2	2		4
010	2	2	4	
011		4	2	2
100	2	2	4	
101		4	2	2
110	4		2	2
111			2	6

$C_1 \oplus C_2$

$P_1 \oplus P_2$	00	01	10	11
000	1	0	0	0
001	0.25	0.50	0	0.50
010	0.25	0.25	0.50	0
011	0	0.50	0.25	0.25
100	0.25	0.25	0.50	0
101	0	0.50	0.25	0.25
110	0.50	0	0.25	0.25
111	0	0	0.25	0.75

- 길이가 3비트 이므로 입력 차분은 총 8($= 2^3$)가지 경우가 존재함
- 위의 예시는 확률이 균일한 분포로 존재하지 않음

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (6/12)
  - 차분 분석(Differential Cryptanalysis) - 과정

    2. 선택 평문 공격의 시작
       - 이전의 분석한 내용은 암호의 구조가 변하지 않는 한 지속적으로 사용할 수 있음
       - 이를 기반으로 공격을 위해 필요한 평문을 선택함
         - 차분 확률 분포표에서 가장 높은 확률을 갖는 평문의 차분을 선택해야 함

    3. 키 값의 추측 (1/2)
       - 키 값의 추측을 위해 특정 평문과 암호문 쌍을 찾음
       - 암호문 C로부터 평문 P가 생성되도록 함

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (7/12)
  - 차분 분석(Differential Cryptanalysis) - 과정
    3. 키 값의 추측 (2/2)
       - 예제 5.16

a. 조건 기반 분석

- $P_1 \oplus P_2 = 001,\ C_1 \oplus C_2 = 11$인 경우 선택
- 확률이 0.5인 경우로 유의미한 차분

X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

S-box table

b. 평문/암호문 쌍 역산 (선택 평문 공격 기반)

- $P_1 = 010, P_2 = 011$로 설정
- $C_1 = 00 \rightarrow X_1 = 001\ or\ 111$
  $\rightarrow P_1 = 010 \rightarrow K = X_1 \oplus P_1 = 011\ or\ 101$
- $C_2 = 11 \rightarrow X_2 = 000\ or\ 110$
  $\rightarrow P_2 = 011 \rightarrow K = X_2 \oplus P_2 = 011\ or\ 101$
- $\therefore K \in \{011, 101\}$

$C_1 \oplus C_2$

$P_1 \oplus P_2$	00	01	10	11
000	1	0	0	0
001	0.25	0.50	0	0.50
010	0.25	0.25	0.50	0
011	0	0.50	0.25	0.25
100	0.25	0.25	0.50	0
101	0	0.50	0.25	0.25
110	0.50	0	0.25	0.25
111	0	0	0.25	0.75

c. 후보군 축소 및 비트 유추

- 두 후보군이 공유하는 오른쪽 최상위 비트가 1임을 확인
- 다양한 조건으로 위 과정을 반복하여 더 많은 키 비트 확인 가능

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (8/12)
  - 차분 분석(Differential Cryptanalysis)
    - 차분 분석을 이용한 공격 절차
      1. 각 라운드가 동일하므로 각각의 S-박스에 대한 차분 분포표 구성
      2. 각 라운드가 독립이라는 가정 하에, 대응하는 확률을 곱해서 전체 암호에 대한 분포표 구성
      3. 2단계에서 구성한 분포표에 기반하여 공격에 필요한 평문 목록 구성
      4. 암호문을 선택하고 그에 대응하는 평문을 획득, 키의 특정 비트 정보 추출을 위해 결과 분석
      5. 더 많은 키 비트 정보를 얻기 위해 4단계 반복
      6. 충분한 키 비트의 정보를 획득 후, 전체 키를 알기 위한 전수조사 공격 시도

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (9/12)
  - 선형 분석(Linear Cryptanalysis)
    - 선형(Linear)
      - 입력•출력 비트, 키 비트 간의 관계가 XOR로 표현될 수 있는 성질
    - 선형분석의 한계
      - S-box는 본래 비선형 구조를 가지므로 근사식이 정확하지 않음
      - 유의미한 통계적 분석을 위한 충분한 양의 암복호 쌍이 필요함
      - 확률 편차(ε)가 작으면 효과가 미미함
        - $\varepsilon \geq 0.05$ 부터 실용적인 선형 분석으로 판단할 수 있음

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (10/12)
  - 선형 분석(Linear Cryptanalysis) - 과정
    1. 선형 근사식(Linear Approximation)
       - 비선형 알고리즘을 구성하는 연산들에 대해 입력,출력,키 비트 사이의 선형적인 관계가 특정 확률로 근사적으로 성립하는 식
       - S-박스는 특정 선형 함수에 의해 확률적으로 근사적으로 될 수 있음
       - 일반적으로 n-비트 평문과 암호문, m-비트의 키가 주어지면 아래의 수식을 갖춤
         $(k_0 \oplus k_1 \oplus \cdots \oplus k_x) = (p_0 \oplus p_1 \oplus \cdots \oplus p_y) \oplus (c_0 \oplus c_1 \oplus \cdots \oplus c_z)$
         - 단, $1 \le x \le m, 1 \le y \le n, 1 \le z \le n$ 을 만족함
       - 각 수식은 확률 $\frac{1}{2} + \varepsilon$ 을 만족함 (ε 는 편차)
         - ε가 클수록 효과적임

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (11/12)
  - 선형 분석(Linear Cryptanalysis) – 과정

    2. 데이터 수집
       - 선형 근사식 적용을 위해 다수의 (평문, 암호문) 쌍을 수집함
       - 일반적으로 수천~수십만 개의 쌍이 필요함
         - 선형 근사식이 확률적으로 성립하므로, 유의미한 식별을 위해 충분한 양의 표본이 필요함

    3. 근사식 검증
       - 수집한 쌍들에 대해 선형 근사식을 적용함
       - 좌변과 우변이 일치하는 빈도(경우의 수)를 계산함
       - 성립한 횟수 $T \rightarrow$ 경험적 확률 $\hat{p} = T/N$

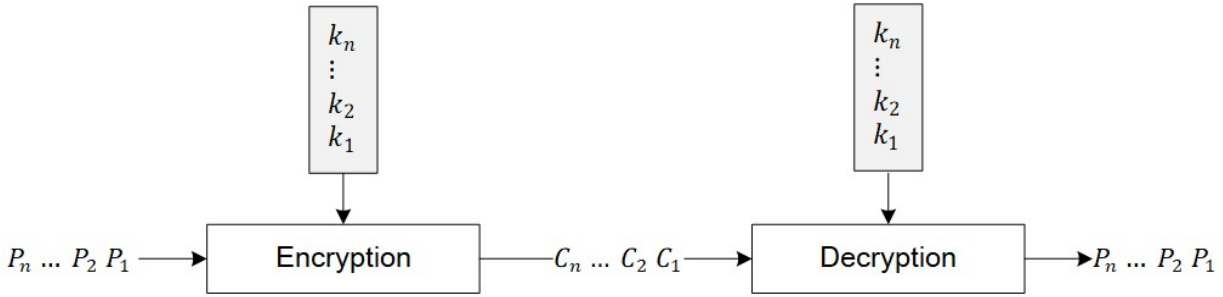
    4. 편향(ε) 검증
       - 위에서 계산한 $\hat{p}$가 충분히 0.5를 넘는지 판단함
       - $\varepsilon = \hat{p} - 0.5$

현대 대칭-키 암호 소개

- 블록 암호에 대한 공격 (12/12)
  - 선형 분석(Linear Cryptanalysis)
    - 선형 분석을 이용한 공격 절차
      1. 각 라운드의 S-box에 대한 선형 근사식 구성
      2. 키 비트 일부가 포함된 근사식 선택
      3. (평문, 암호문) 쌍들을 적용하여 좌변=우변인 경우의 빈도 계산
      4. 빈도가 가장 크게 편향된 경우의 키 선택
      5. 필요한 경우 다른 근사식으로 반복 수행하여 키 후보 축소
      6. 가능한 키 후보에 대해 전수조사 등으로 전체키 결정

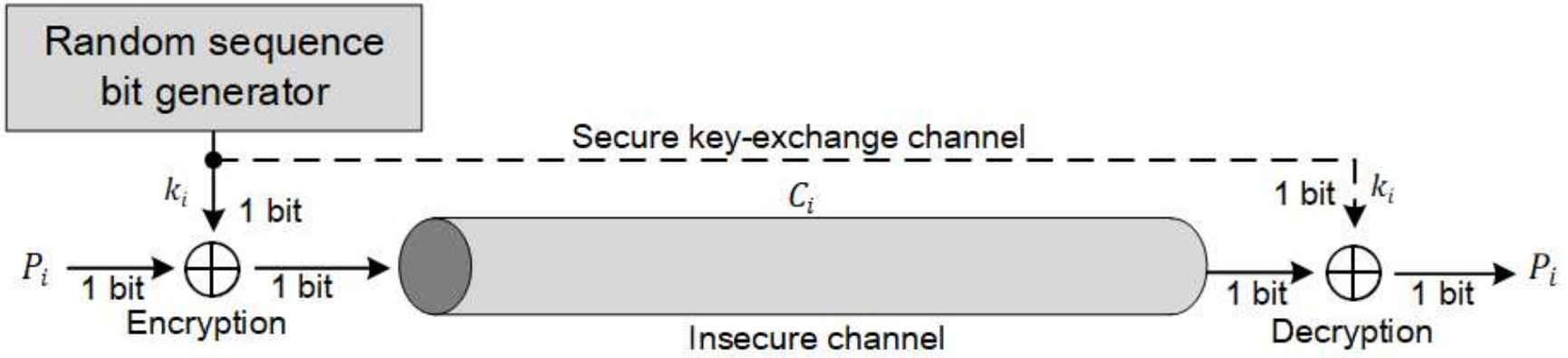
현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 정의
    - 평문 스트림의 r-비트 워드를 키 스트림의 r비트 워드를 사용하여 r-비트 암호문 워드로 암호화하는 방식
  - 종류
    - 동기식 스트림 암호
      - 키 + 초기값에 기반한 내부 상태로 키 스트림을 생성함
      - 오류 발생 시 해당 비트만 영향을 미침
    - 비동기식 스트림 암호
      - 이전 암호문 비트에 기반하여 동적으로 키스트림을 생성함
      - 오류 발생 시 다수 비트에 영향을 미침



현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (1/11)
    - 일회용 패드(One-Time Pad, OTP)
      - 암호화를 수행할 때마다 평문 비트열과 동일한 길이의 무작위 키 스트림을 생성하고 각 비트를 XOR 연산으로 암호화 하는 방식
        - 1비트 상에서 수행되며 체는 $GF(2)$
      - 키 스트림을 전달할 수 있는 안전한 채널이 존재해야 함
      - 공격자가 키나 평문을 추측할 수 있는 방법이 없으며, 암호문의 통계적 특성이 존재하지 않음
        - 전수조사로만 공격이 가능 함



현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (2/11)
    - 일회용 패드
      - 예제 5.10

다음 경우 각각에 대하여 일회용 패드 암호의 암호문의 패턴은 무엇인가
a. 평문이 n개의 0으로 구성되는 경우
b. 평문이 n개의 1로 구성되는 경우
c. 평문이 0과 1로 교차되어 구성되는 경우
d. 평문이 랜덤 비트의 스트림인 경우

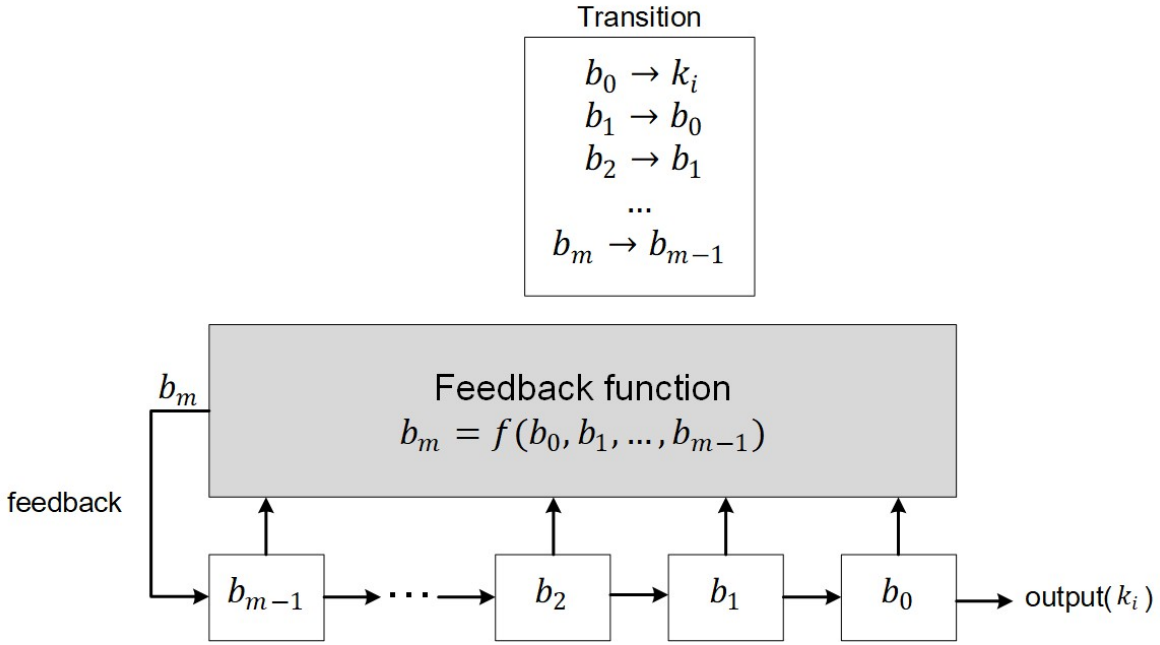
a. $0 \oplus k_i = k_i$ 이므로, 암호문 스트림은 키 스트림과 동일함 (패턴 유지)
b. $1 \oplus k_i$ 의 연산 결과는 k_i의 보수이므로 암호문 스트림은 키 스트림의 보수와 같음 (패턴유지)
c. 1과 0이 번갈아 등장하므로 암호문 스트림의 각 비트는 키 스트림에 대응하는 비트와 같거나 보수임 (패턴유지)
d. P가 랜덤이면 평문 자체에 패턴이 없기 때문에 무작위로 보임 (패턴 없음)

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (3/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 이전 비트 값을 기반으로 다음 비트를 생성하는 구조의 반복 회로
      - 소프트웨어와 하드웨어 환경에서 모두 구현될 수 있음
        - 하드웨어 구현이 상대적으로 용이함
      - 구성요소
        - 귀환(Feedback) 함수
        - 이동(Shift) 레지스터

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (4/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 구성요소 - 귀환함수
        - 레지스터 내 특정 비트의 값을 논리 연산하여 다시 주입하는 함수
        - 셀들의 값을 입력으로 받아 b_m 값의 생성 방식을 정의함
        - $b_m = f(b_0, b_1, \cdots, b_{m-1})$



현대 대칭-키 암호 소개

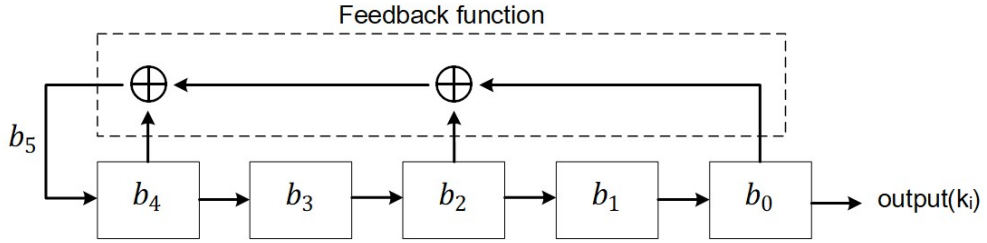
- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (5/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 구성요소 - 이동 레지스터
        - 구조
          - b_0에서 b_{m-1}까지 m개의 셀로 구성된 일렬 배열
          - 각각의 셀은 1비트 값을 저장함
          - 전체 레지스터는 m-비트 길이를 가짐
        - 초기화
          - 시작할 때 레지스터는 임의의 m-비트 값으로 설정됨
          - 초기화하는 초기 값을 시드(seed)라고 하며, 알고리즘의 시작 상태를 결정함
        - 동작 방식
          - 출력비트가 필요할 때마다 각 셀은 자신의 값을 오른쪽 셀에 전달함
          - 오른쪽 최상위 셀 b_0는 출력 값 k_i로 자신의 값을 출력함
          - 왼쪽 최상위 셀 b_{m-1}은 귀환 함수의 출력 값을 가짐

현대 대칭-키 암호 소개

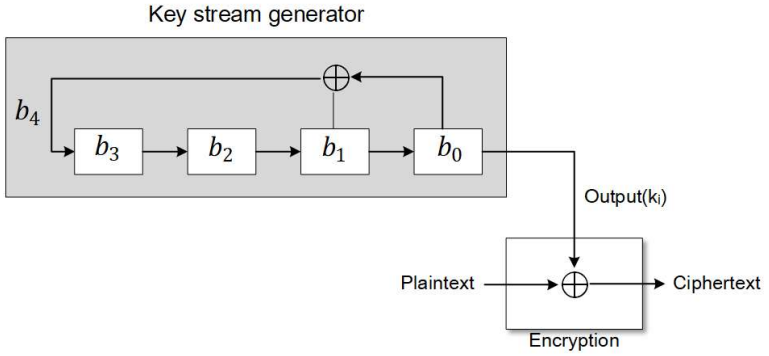
- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (6/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 선형 귀환 이동 레지스터(LFSR, Linear Feedback Shift Register)
        - 귀환 함수가 선형인 FSR의 한 형태
        - LFSR에서 b_m은 $b_0, b_1, \cdots, b_{m-1}$의 선형 함수임
          - $b_m = c_{m-1}b_{m-1} \oplus \cdots \oplus c_2 b_2 \oplus c_1 b_1 \oplus c_0 b_0 \ (c_0 \neq 0)$
        - 체 $GF(2)$ 상에서 C_i의 값은 1이거나 0이지만, C_0는 출력 값이 귀환 되어야 하므로 항상 1임

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (7/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 선형 귀환 이동 레지스터(LFSR, Linear Feedback Shift Register)
        - e.g., $b_5 = b_4 \oplus b_2 \oplus b_0$을 만족하고 5개의 셀을 갖는 선형 귀환 이동 레지스터 설계



        - e.g., $b_4 = b_1 \oplus b_0$을 만족하고 4개의 셀을 갖는 선형 귀환 이동 레지스터 설계 (단, 시드가 $(0001)_2$일 때 20번의 변환에 대한 출력 값 출력



현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (8/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 선형 귀환 이동 레지스터 - 의사난수열
        - 난수처럼 보이지만, 규칙적인 알고리즘에 의해 생성된 수열
        - LFSR은 시드에 따라 결정적으로 동작하며, 키 스트림으로 의사난수열을 생성함
        - 스트림의 주기는 시드와 관련이 있으나 정확히 같지는 않음 (주기 $\leq 2^m - 1, m$은 레지스터 길이)
        - m-비트 시드는 모두 0인 수열에서부터 모두 1인 수열까지 2^m개의 서로 다른 패턴을 만들어 냄
          - 시드가 모두 0이면 생성된 키 스트림은 사용하지 않음
          - 평문 스트림이 모두 0인 경우에도 암호화에서 제외됨

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (9/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 선형 귀환 이동 레지스터 - 의사난수열
        - e.g. 의사난수열 예시
          - $b_m = b_0 \oplus k_i$ 의 선형귀환함수를 가지는 의사난수열의 키 스트림
          - 난수열 처럼 보이나, 지속적으로 키스트림을 생성하면, 100010011010111의 15비트가 반복됨

states	b_4	b_3	b_2	b_1	b_0	k_i
Initial	1	0	0	0	1	
1	0	1	0	0	0	1
2	0	0	1	0	0	0
3	1	0	0	1	0	0
4	1	1	0	0	1	0
5	0	1	1	0	0	1
6	1	0	1	1	0	0
7	0	1	0	1	1	0
8	1	0	1	0	1	1
9	1	1	0	1	0	1
10	1	1	1	0	1	0
11	1	1	1	1	0	1
12	0	1	1	1	1	0
13	0	0	1	1	1	1
14	0	0	0	1	1	1
15	1	0	0	0	1	1
16	0	1	0	0	0	1
17	0	0	1	0	0	0
18	1	0	0	1	0	0
19	1	1	0	0	1	0
20	1	1	1	0	0	1

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (10/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 선형 귀환 이동 레지스터 - 특성 다항식(Characteristic Polynomial)
        - LFSR의 귀환 함수를 수학적으로 표현한 이항 다항식
        - LFSR이 키스트림의 최대 주기를 갖기 위하여 사용함
          - 최대 주기를 만들기 위해서는 특성 다항식이 *원시 다항식이어야함.

        - e.g., LFSR의 특성다항식 정의
          - m-비트 LFSR에서 비트 상태가 $b_0, b_1, \ldots, b_{m-1}$이고, 귀환 함수가 $b_m = c_0 b_0 \oplus c_1 b_1 \oplus \cdots \oplus c_{m-1} b_{m-1}$ 임
          - $c_i \in \{0,1\}$이고, $GF(2)$에서 정의됨
          - $f(x) = x^m + c_{m-1}x^{m-1} + \cdots + c_1 x + c_0$로 특성다항식이 정의됨

*원시 다항식(Primitive Polynomial)
인수분해가 되지 않는 기약 다항식이면서 $x^k \equiv 1 \bmod \mathrm{p(x)}$를 만족하는 가장 작은 k가 $2^{\mathrm{m}} - 1$인 다항식

현대 대칭-키 암호 소개

- 현대 스트림 암호(Morden Stream Cipher)
  - 동기식(Symchronous) 스트림 암호 (11/11)
    - 귀환 이동 레지스터(FSR, Feedback Shift Register)
      - 비선형 귀환 이동 레지스터(NLFSR, Nonlinear Feedback Shift Register)
        - 귀환 함수가 선형이 아닌 FSR의 한 형태
        - b_m이 비선형 함수로 정의되며, 이외에는 LFSR과 동일한 구조를 가짐
      - 결합(Combination)
        - 스트림 암호는 선형과 비선형 구조를 결합하여 사용함
          - 선형구조는 빠르고 단순하여 키 스트림 생성에 적합하지만 예측 가능하다는 단점을 가짐
          - 비선형 구조는 키 스트림에 복잡성과 예측 불가능성을 부여하여 통계적 분석 및 선형/차분 분석에 강한 저항성을 가짐
          - 결합을 통해 속도와 보안성 모두를 만족하는 키 스트림 생성기를 구현할 수 있음
        - e.g., A5/1

현대 대칭-키 암호 소개

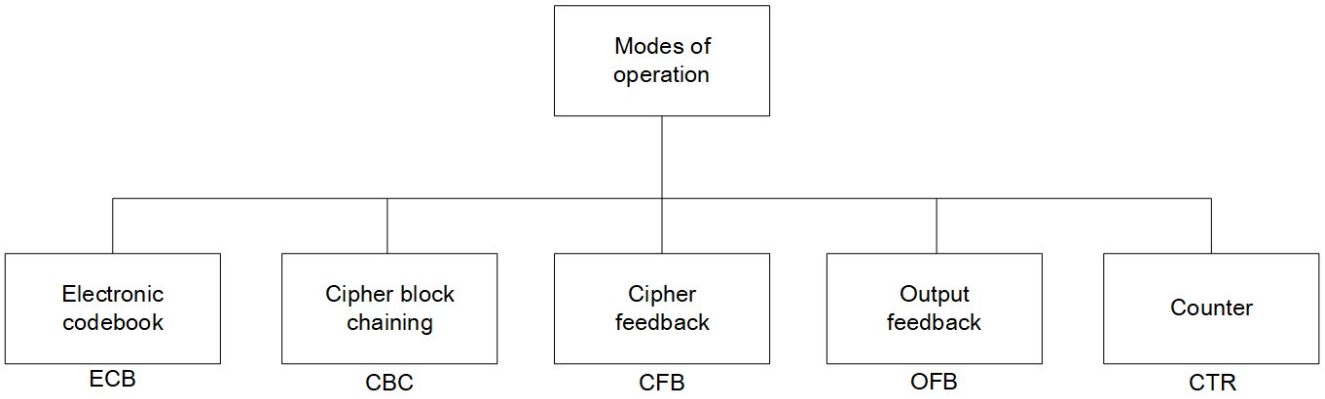
- 현대 스트림 암호(Morden Stream Cipher)
  - 비동기식(Nonsymchronous) 스트림 암호
    - 정의
      - 키 스트림이 평문 혹은 암호문에 의존적으로 생성되는 방식
    - 특징
      - 이전 평문/암호문을 참조하여 다음 키 스트림을 생성함
      - 키 스트림 생성기가 피드백 구조를 가짐
        - e.g., CFB, OFB, CTR 등
      - 일부 모드는 오류 파급이 발생할 수 있음
        - e.g., CFB 등

목 차

- 보충
- 현대 대칭-키 암호 소개
  - 현대 블록 암호 개요
  - 치환 군으로서의 블록 암호
  - 현대 블록 암호의 구성요소
  - 현대 블록 암호
  - 블록 암호에 대한 공격
  - 현대 스트림 암호
- 현대 대칭-키 암호를 이용한 암호화 기법
  - 현대 블록 암호의 사용
  - 스트림 암호의 사용
  - 키 관리와 키 생성

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - 임의의 길이의 텍스트를 암호화 하는 방법
    - 종류
      - ECB(Electronic Codebook)
      - CBC(Cipher Block Chaining)
      - CFB(Cipher Feedback)
      - OFB(Output Feedback)
      - CTR(Counter)



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - 암호문 훔치기(CTS, Ciphertext Stealing)
      - 정의
        - 덧붙이기 없이 평문과 같은 크기로 암호문을 생성하도록 하는 방법
      - 사용 요소
        - P_{N-1}: n-비트 블록
        - P_N : $m(m < n)$-비트 블록
        - $head_m$: 왼쪽 최상위 m비트를 선택하는 함수
        - $tail_{n-m}$: 오른쪽 최상위 $(n-m)$비트를 선택하는 함수
      - 동작 절차
        - $X = E_k(P_{N-1}) \quad \rightarrow \quad C_N = head_m(X)$
          $Y = P_N | tail_{n-m}(X) \quad \rightarrow \quad C_{N-1} = E_K(Y)$

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - 초기 벡터(IV, Initial Vector)
      - 블록 암호에서 첫 번째 블록의 암호화를 위한 초기 입력 값
      - 송신자와 수신자간에 공유되어야 함
      - 반드시 비밀일 필요는 없음
      - IV를 사용하는 권장 방법
        - 송신자가 의사난수를 선택하고, 선택된 난수열을 안전한 채널로 전송함
        - 둘 사이의 비밀 키를 공유하고 있는 경우, 고정된 값을 IV로 공유함

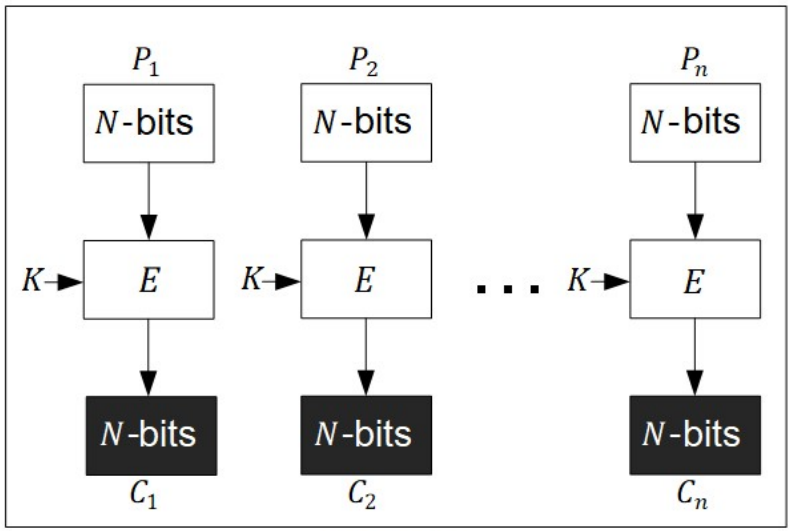
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - ECB(Electronic Codebook) 모드 (1/5)
      - 정의
        - 각 평문 블록을 독립적으로 암호화하는 가장 단순한 블록 암호 모드
      - 특징
        - 각 블록이 독립적으로 암호화됨
          - 블록 간 연관성 없음
          - 암호화와 복호화 모두 병렬 처리 할 수 있음
        - 항상 동일한 키를 사용하므로 동일한 평문 블록은 동일한 암호문 블록을 생성함
          - 패턴 노출 위험성이 존재함
        - 평문을 N개의 n비트 블록으로 분할하므로 평문의 크기가 n의 배수가 아니면 마지막 블록에 덧붙이기(padding)가 필요함
      - 알고리즘
        - ECB 모드의 알고리즘은 부록 #1 참고

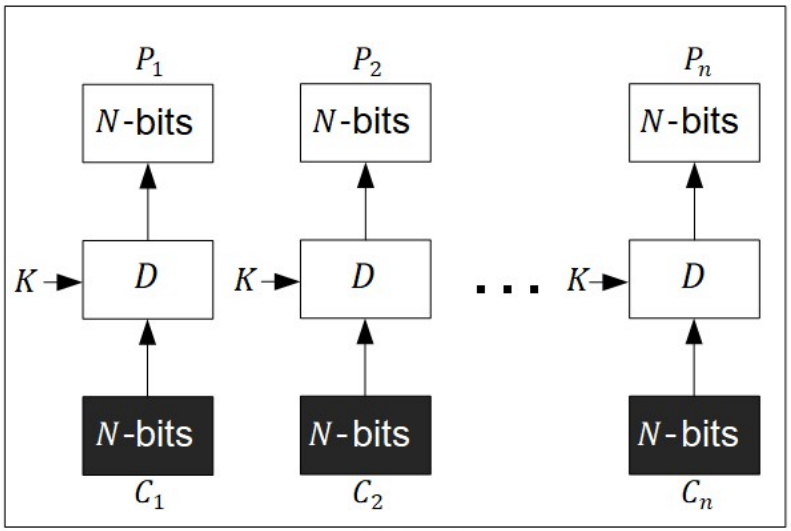
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - ECB(Electronic Codebook) 모드 (2/5)
      - 동작 과정
        - 암호화 : $C_i = E_K(P_i)$
        - 복호화 : $P_i = D_K(C_i)$

E : Encryption D : Decryption
P_i : Plaintet block i C_i : Ciphertext block i
K : Secret key



Encryption



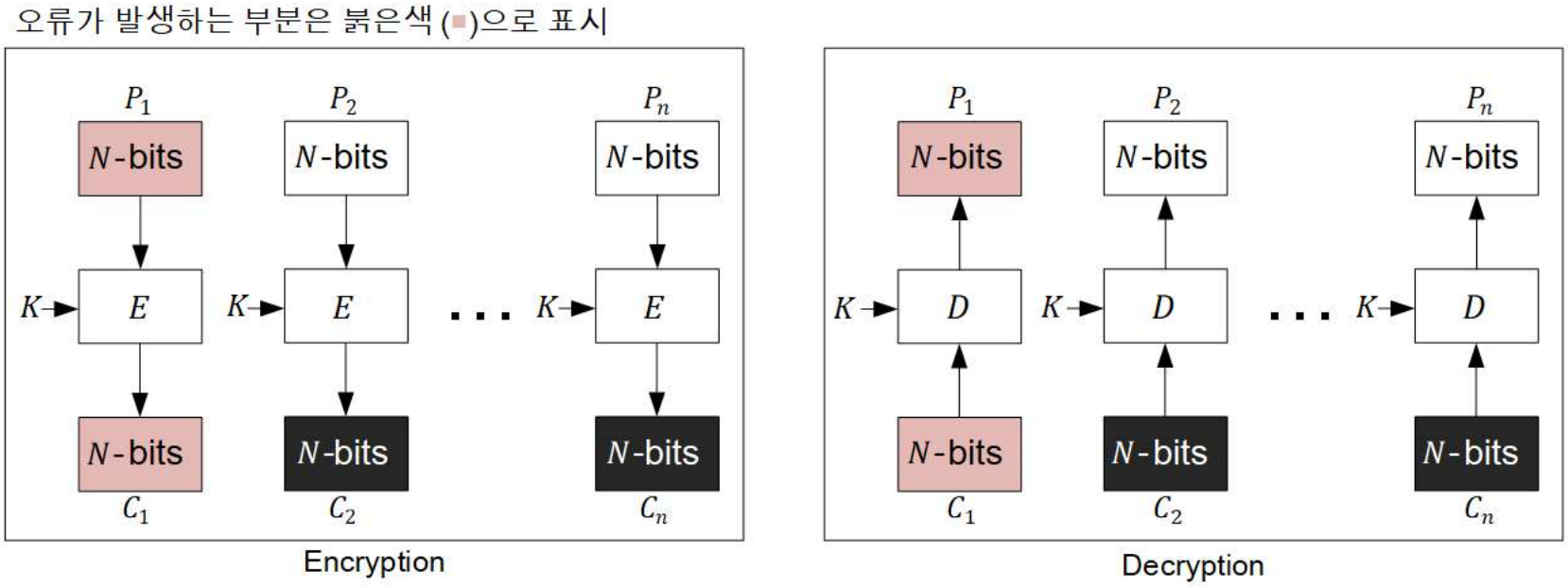
Decryption

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - ECB(Electronic Codebook) 모드 (3/5)
      - CTS
        - $X = E_K(P_{N-1}) \rightarrow C_N = head_m(X)$
          - P_{N-1}: 마지막에서 두 번째 평문 블록 (n-비트 블록)
          - $E_K(P_{N-1})$: 일반적인 ECB 암호화 수행
          - $head_m(X)$: X의 상위 m-비트를 잘라서 마지막 암호문 블록 C_N으로 사용
        - $Y = P_N|tail_{n-m}(X) \rightarrow C_{N-1} = E_K(Y)$
          - P_N : 마지막 평문 블록
          - $tail_{n-m}(\mathrm{X})$: X의 하위 $(n-m)$비트를 잘라 P_N뒤에 이어 붙임
          - Y : 총 n-비트 블록으로 C_{N-1}에 저장

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - ECB(Electronic Codebook) 모드 (4/5)
      - 오류 파급
        - 암호화와 복호화 모두 오류가 발생한 블록에만 영향을 미치고, 다른 블록은 영향을 받지 않음



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - ECB(Electronic Codebook) 모드 (5/5)
      - 장점
        - 각 평문 블록을 독립적으로 동일한 키로 암호화 하므로 처리 속도가 빠름
        - 각 블록이 독립적으로 처리되므로 여러 블록을 빠르게 병렬 처리 할 수 있음
      - 단점
        - 동일한 평문 블록이 동일한 암호문을 생성하므로 반복되는 블록이 있을 경우 통계적 분석이나 패턴 기반 공격에 취약함

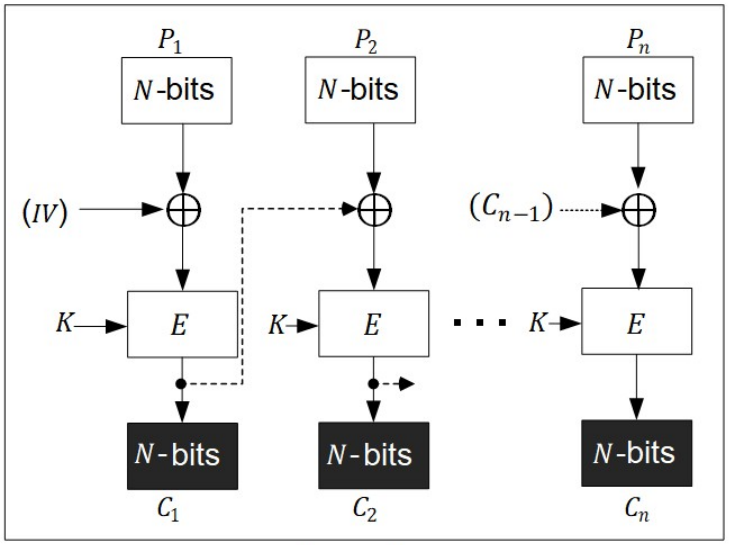
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CBC(Cipher Block Chaining) 모드 (1/5)
      - 정의
        - 각 평문 블록에 이전 암호문 블록을 XOR하여 암호화하는 방식
      - 특징
        - 암호화 과정이 이전 암호문 블록에 의존적임
          - 블록 간 연관성이 있음
          - 암호화는 병렬 처리할 수 없음
        - 복호화 과정은 이전 암호문 블록만 필요함
          - 블록 간 연관성 없음
          - 복호화는 병렬 처리할 수 있음
        - 고정 블록 크기에 맞추기 위해 평문 길이가 부족한 경우 패딩이 필요함
        - 첫 블록의 보안성 확보를 위해 IV 필요함
        - 암호문 블록 C_j의 오류는 평문 블록 P_{j+1}까지만 영향을 주므로 자기복구 가능
      - 알고리즘
        - CBC 모드의 알고리즘은 부록 #1 참고

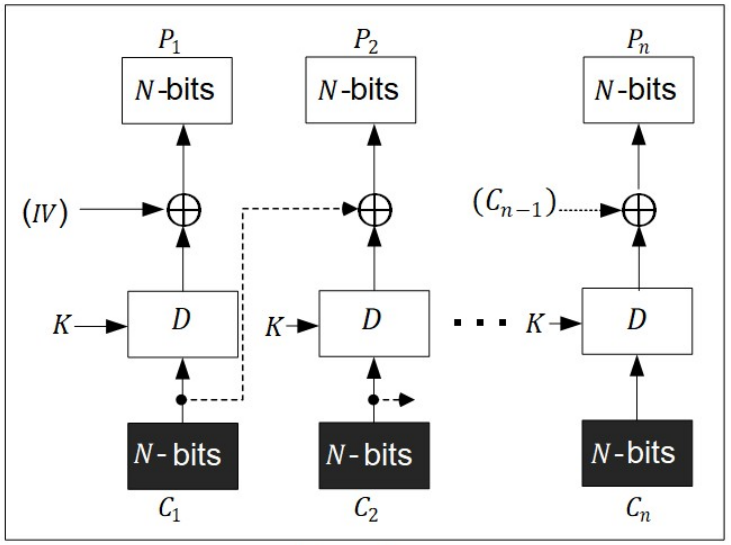
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CBC(Cipher Block Chaining) 모드 (2/5)
      - 동작 과정
        - 암호화 : $C_0 = IV,\ \ C_i = E_K(P_i \oplus C_{i-1})$
        - 복호화 : $P_i = D_K(C_i) \oplus C_{i-1}$

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



Encryption



Decryption

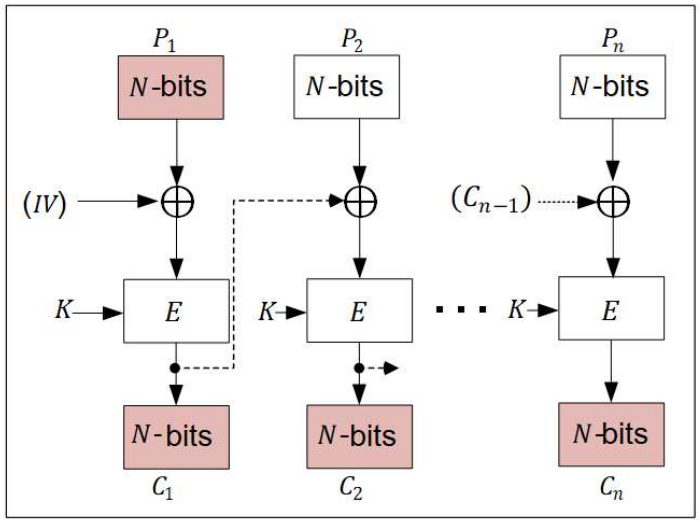
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CBC(Cipher Block Chaining) 모드 (3/5)
      - CTS
        - $U = P_n \oplus C_{n-1} \rightarrow X = E_K(U) \rightarrow C_n = head(X)$
          - P_n : 마지막에서 두 번째 평문 블록
          - C_{n-1} : 이전 암호문 블록
          - U : 평문과 이전 암호문 블록의 XOR 결과
          - $E_K(U)$: 일반적인 CBC 암호화 수행
          - $head(X)$: X의 상위 m-비트를 잘라서 마지막 암호문 C_n으로 사용
        - $V = P_{n+1}|pad(0) \rightarrow Y = X \oplus V \rightarrow C_{n-1} = E_K(Y)$
          - P_{n+1} : 마지막 평문 블록
          - $pad(0)$: 부족한 비트를 0으로 채움
          - V : 총 n-비트로 맞춘 평문 블록
          - $Y = X \oplus V$: X의 출력을 V에 XOR
          - $E_K(Y)$: 암호화 후 결과를 마지막에서 두 번째 암호문 C_{n-1}로 저장

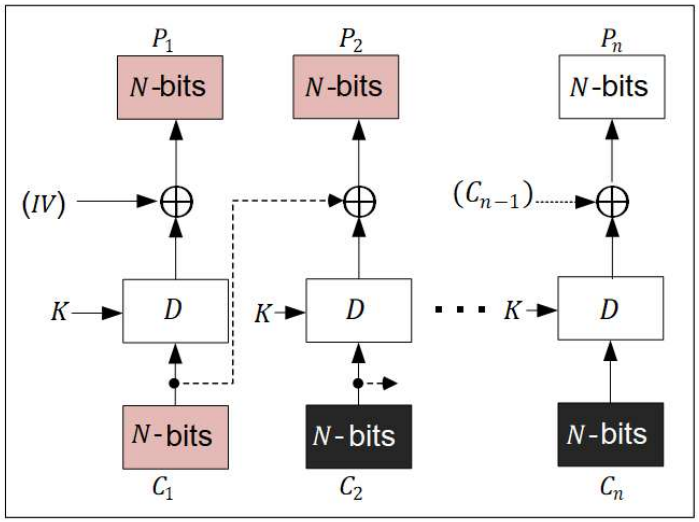
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CBC(Cipher Block Chaining) 모드 (4/5)
      - 오류 파급
        - 암호화에서는 한 평문 블록에 오류가 발생하면, 현재 암호문 블록과 이후 모든 암호문 블록에 오류가 발생함
        - 복호화에서는 한 암호문 블록에 오류가 발생하면, 현재 평문과 다음 평문 블록에만 오류가 발생함

오류가 발생하는 부분은 붉은색 (■)으로 표시



Encryption



Decryption

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CBC(Cipher Block Chaining) 모드 (5/5)
      - 장점
        - 같은 평문 블록이라도 매번 이전 암호문을 사용하므로 암호화 시 서로 다른 암호문 블록을 생성함
          - 패턴 노출이 방지되어 통계적 분석, 패턴 기반 공격에 강함
        - 복호화 과정에서 병렬 처리가 가능하므로 처리 속도가 빠름
      - 단점
        - 암호문 에러 시, 다음 평문 블록에도 영향을 미쳐 복구가 어려움
        - 암호화는 병렬 처리가 불가능하므로 처리 속도가 느림

현대 대칭-키 암호를 이용한 암호화 기법

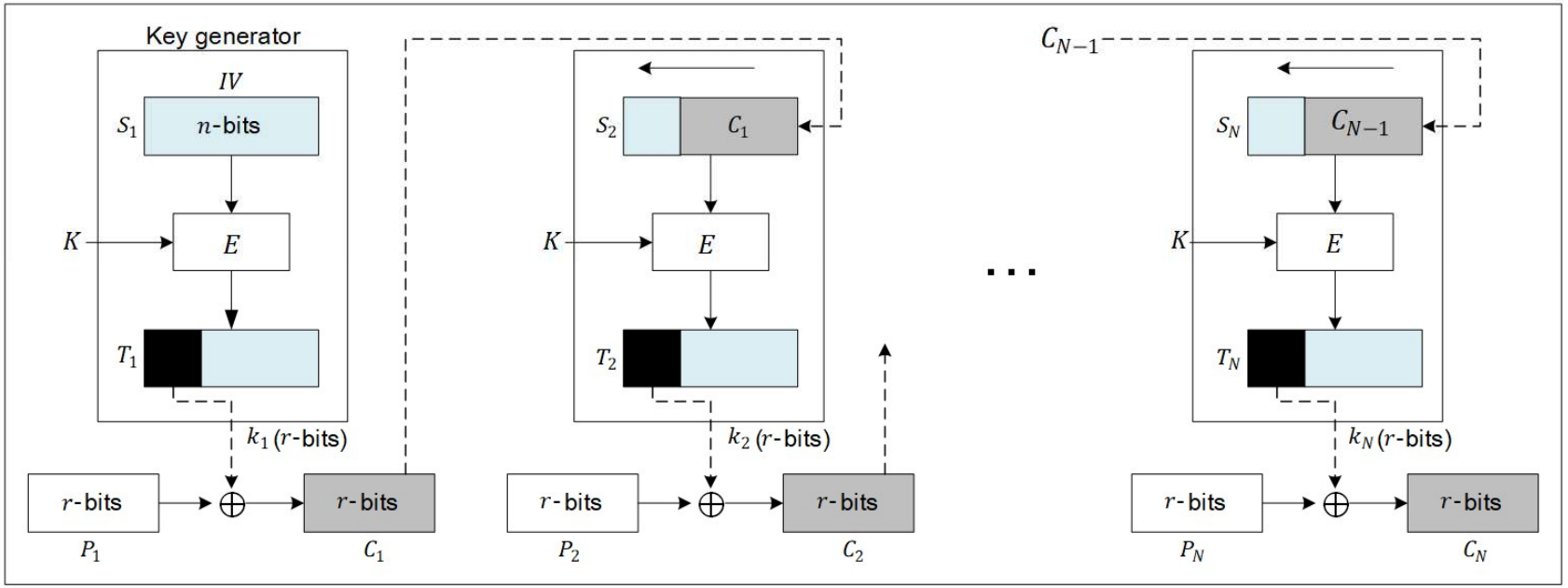
- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (1/7)
      - 정의
        - 이전 암호문을 암호화해 만든 키 스트림을 평문과 XOR하여 암호화하는 방식
      - 특징
        - 암•복호화 시 이전 암호문 블록을 사용함
          - 블록 간 연관성이 존재함
          - 암호화와 복호화 모두 병렬 처리할 수 없음
            - 복호화 시 암호화 함수에 이전 암호문을 사용하므로 CFC와 다르게 병렬 처리할 수 없음
        - 스트림처럼 비트 단위, r-비트 단위 암호화가 가능하므로 패딩이 필요하지않음
        - 첫 키 스트림 생성을 위해 IV 필요함
        - 블록 암호를 스트림 암호로 사용할 수 있음
          - 암호 알고리즘 자체는 블록 암호이지만 입출력 단위를 r-비트로 조절할 수 있음
      - 알고리즘
        - CFB 모드의 알고리즘은 부록 #1 참고

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (2/7)
      - 동작 과정 (1/2)
        - 암호화 : $C_i = P_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{j-1})|C_{i-1}]\}$

ShiftLeft_r : r만큼 왼쪽 쉬프트
$SelectLeft_r$: 상위 r비트 선택

E : Encryption Func C_i : Ciphertext block i S_i : Shift register
P_i : Plaintet block i IV : Initialization vector (C_0) T_i : Temporary register
K : Secret key



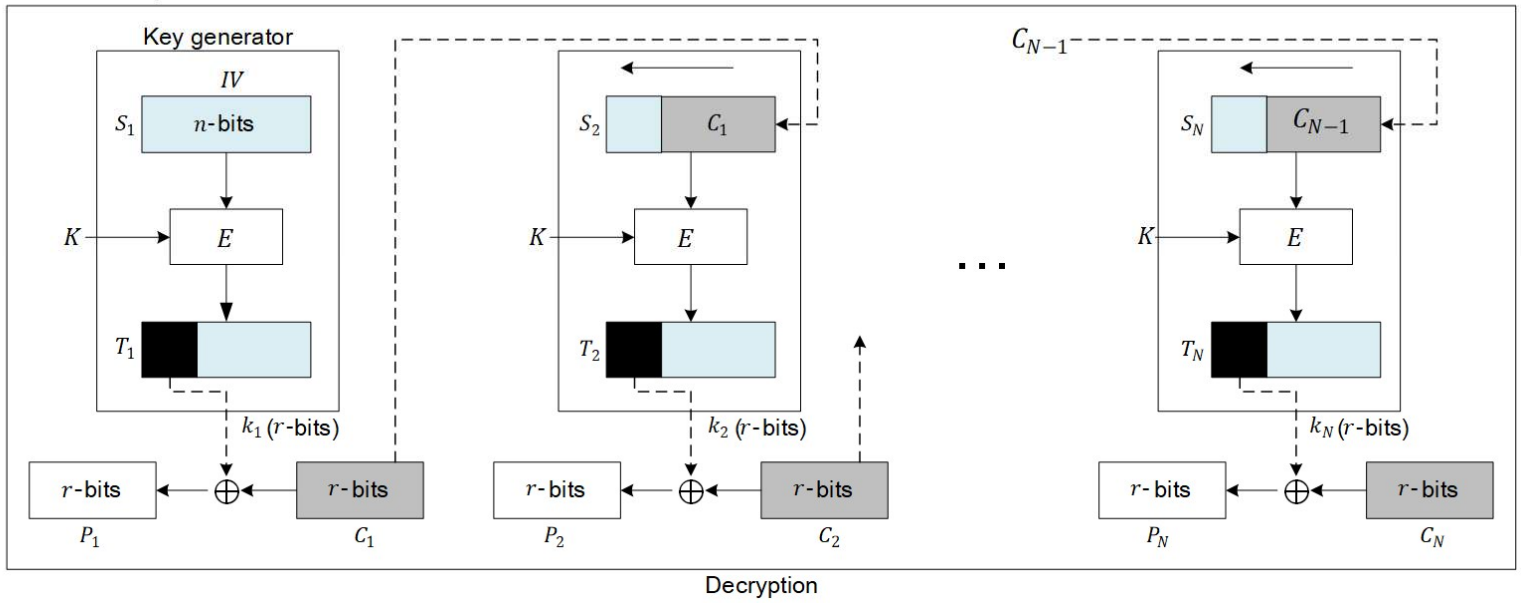
Encryption

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (3/7)
      - 동작 과정 (2/2)
        - 복호화 : $P_i = C_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{j-1})|C_{i-1}]\}$
          - 복호화 시에도 암호화 함수를 사용함

$\mathrm{ShiftLeft}_r$: r만큼 왼쪽 쉬프트
$SelectLeft_r$: 상위 r비트 선택

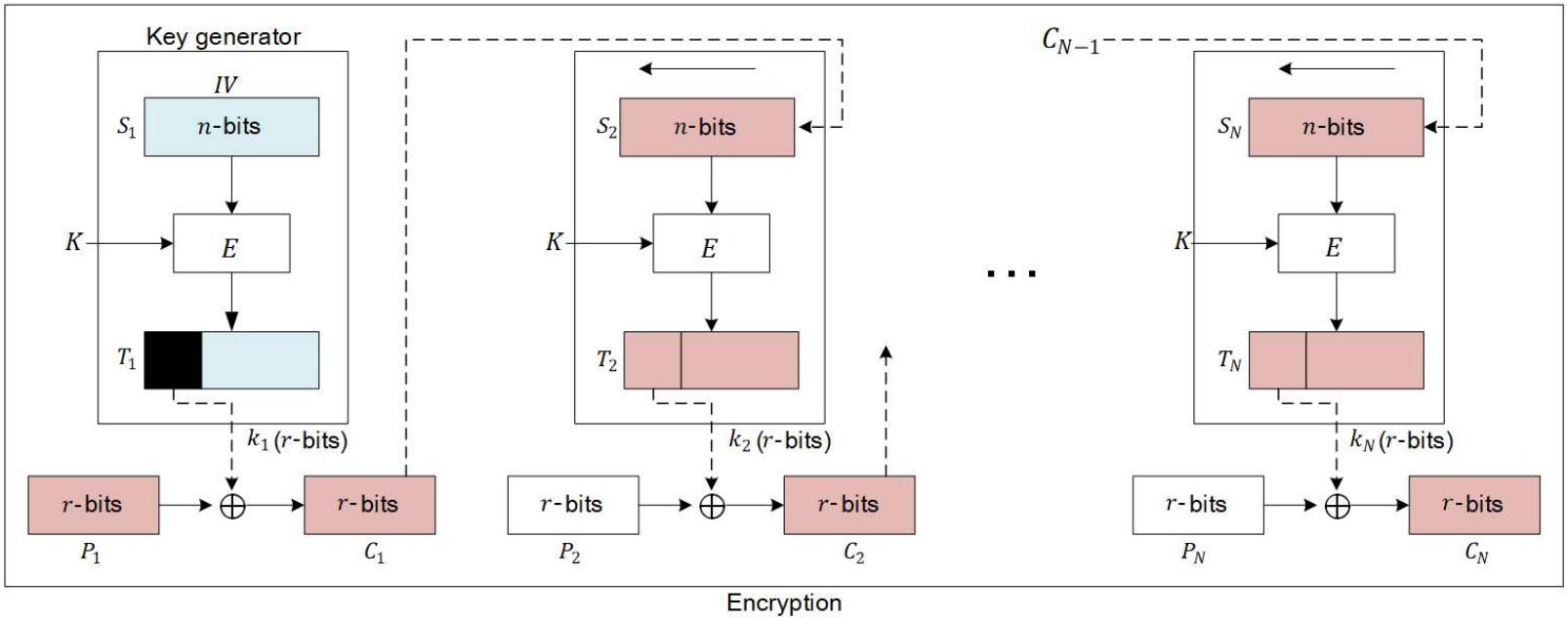
E : Encryption Func C_i : Ciphertext block i S_i : Shift register
P_i : Plaintet block i IV : Initialization vector (C_0) T_i : Temporary register
K : Secret key



현대 대칭-키 암호를 이용한 암호화 기법

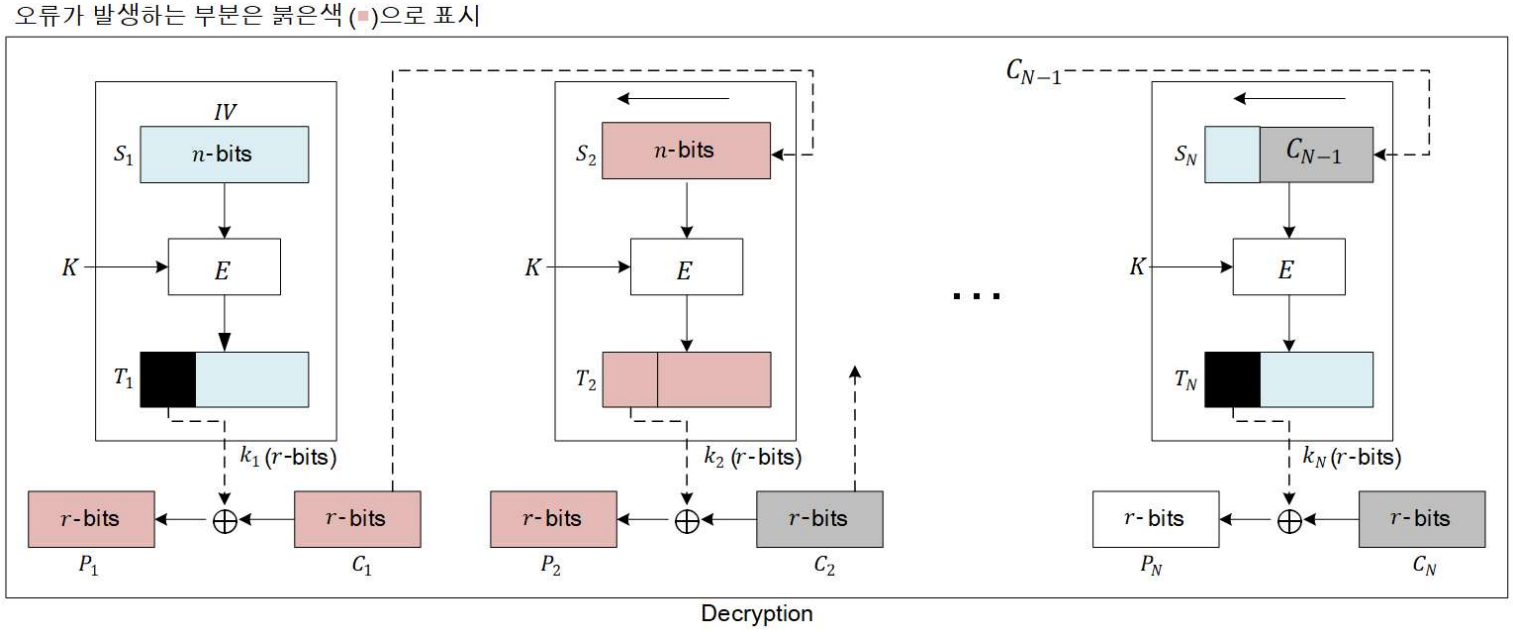
- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (4/7)
      - 오류 파급 (1/2)
        - 암호화
          - 한 평문 블록에 오류가 발생하면, 현재 암호문 블록과 이후 모든 암호문 블록에 오류가 발생함

오류가 발생하는 부분은 붉은색 (■)으로 표시



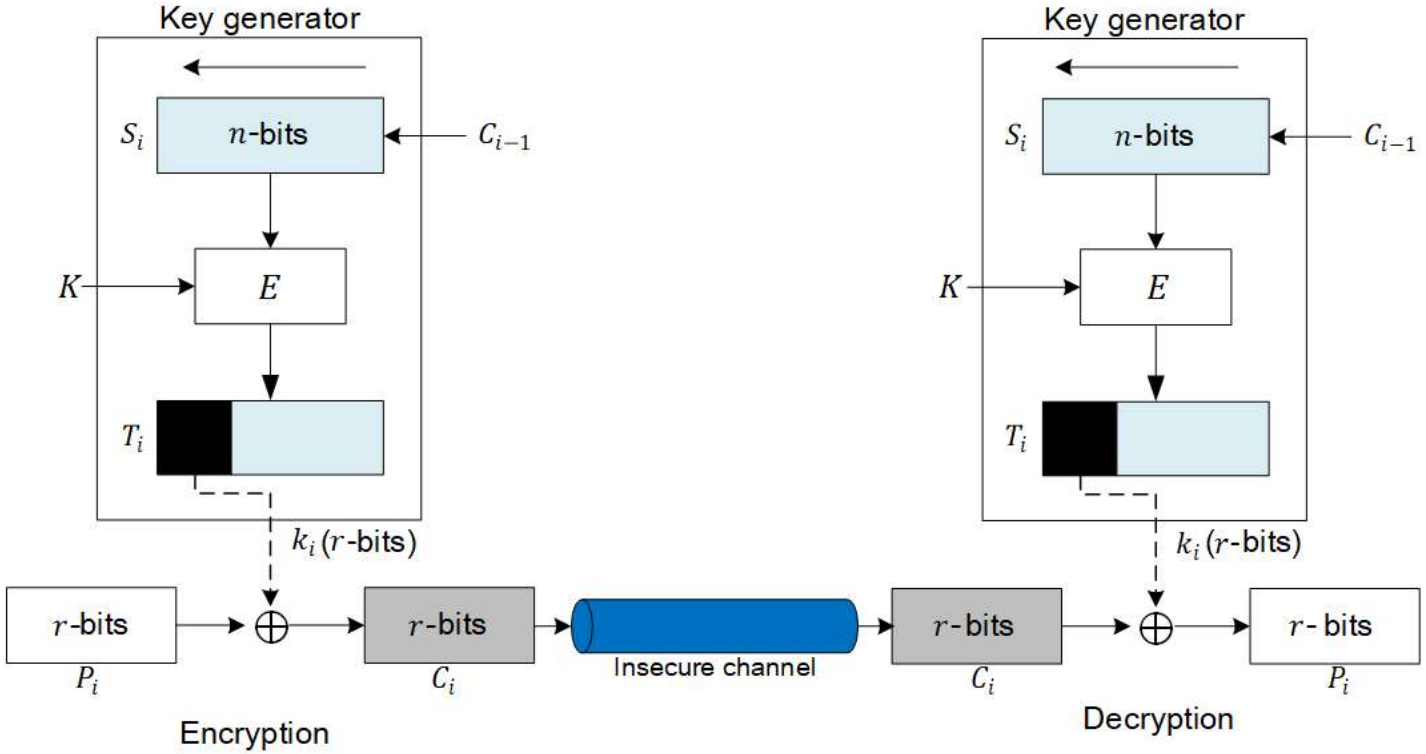
현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (5/7)
      - 오류 파급 (2/2)
        - 복호화
          - 한 암호문 블록에 오류가 발생하면, 현재 평문 블록과 이후 이동 레지스터에 오류가 존재하는 동안의 평문 블록에 오류가 발생함



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (6/7)
      - 스트림 암호로서의 CFB모드
        - 키 스트림이 암호문에 의존하는 비동기식 스트림 암호임



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CFB(Cipher Feedback) 모드 (7/7)
      - 장점
        - 암호화 시 이전 암호문 블록을 사용하므로 평문 구조나 반복이 암호문에 드러나지 않아 안전함
        - 복호화 과정에서 병렬 처리가 가능하므로 처리 속도가 빠름
      - 단점
        - 암호문 에러 시, 다음 평문 블록에도 영향을 미쳐 복구가 어려움
        - 암호화는 병렬 처리가 불가능하므로 처리 속도가 느림

현대 대칭-키 암호를 이용한 암호화 기법

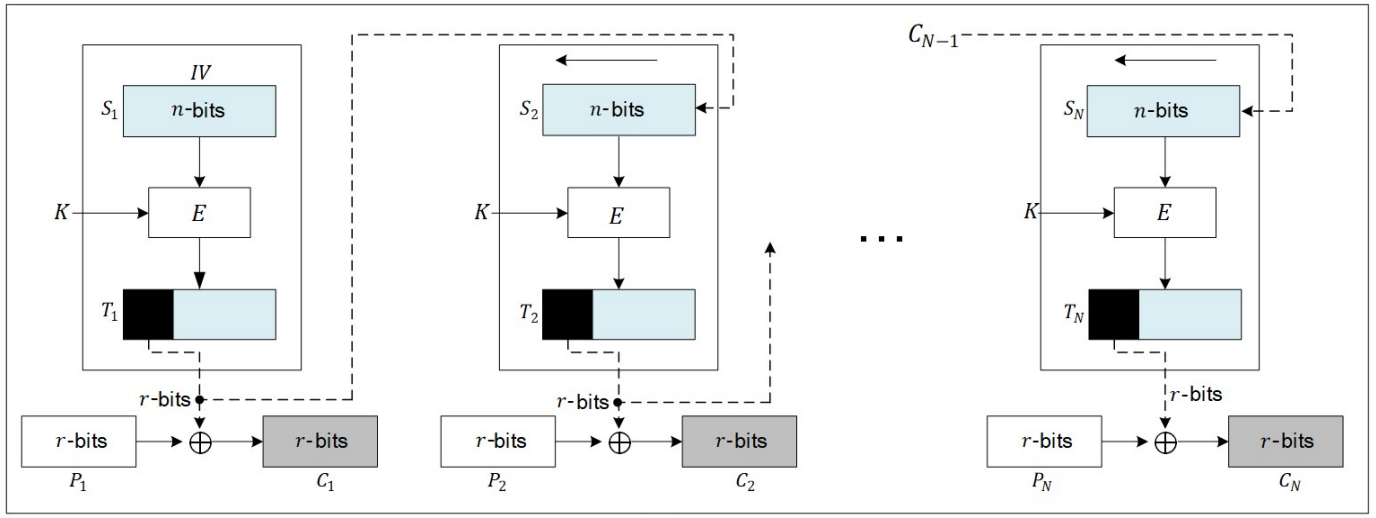
- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (1/7)
      - 정의
        - 이전 암호 출력값을 암호화하여 키 스트림을 만들고 평문과 XOR하여 암호화하는 방식
      - 특징
        - 키 스트림 생성이 암호문과 무관함
          - 블록 간 독립성이 존재함
          - 병렬 처리할 수 있음
        - 스트림처럼 연속적인 데이터 처리가 가능하므로 패딩이 필요하지 않음
        - 최초 피드백 입력값 생성을 위해 IV가 필요함
        - 블록 암호를 스트림 암호로 사용할 수 있음
          - 암호 알고리즘 자체는 블록 암호이나, 출력 키스트림을 사전에 생성하고 이를 r-비트 단위로 연속처리할 수 있음
        - CFB에서 오류 파급을 개선함
          - 암호문에 오류가 생겨도 해당 평문 비트에만 영향을 미침

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (2/7)
      - 알고리즘
        - ECB 모드의 알고리즘은 부록 #1 참고
      - 동작 과정 (1/2)
        - 암호화 : $C_i = P_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{j-1})|SelectLeft_r(T_{i-1})]\}$

$\mathrm{ShiftLeft}_r$: r만큼 왼쪽 쉬프트
$SelectLeft_r$: 상위 r비트 선택

E : Encryption func C_i : Ciphertext block i S_i : Shift register
P_i : Plaintet block i IV : Initialization vector (C_0) T_i : Temporary register
K : Secret key



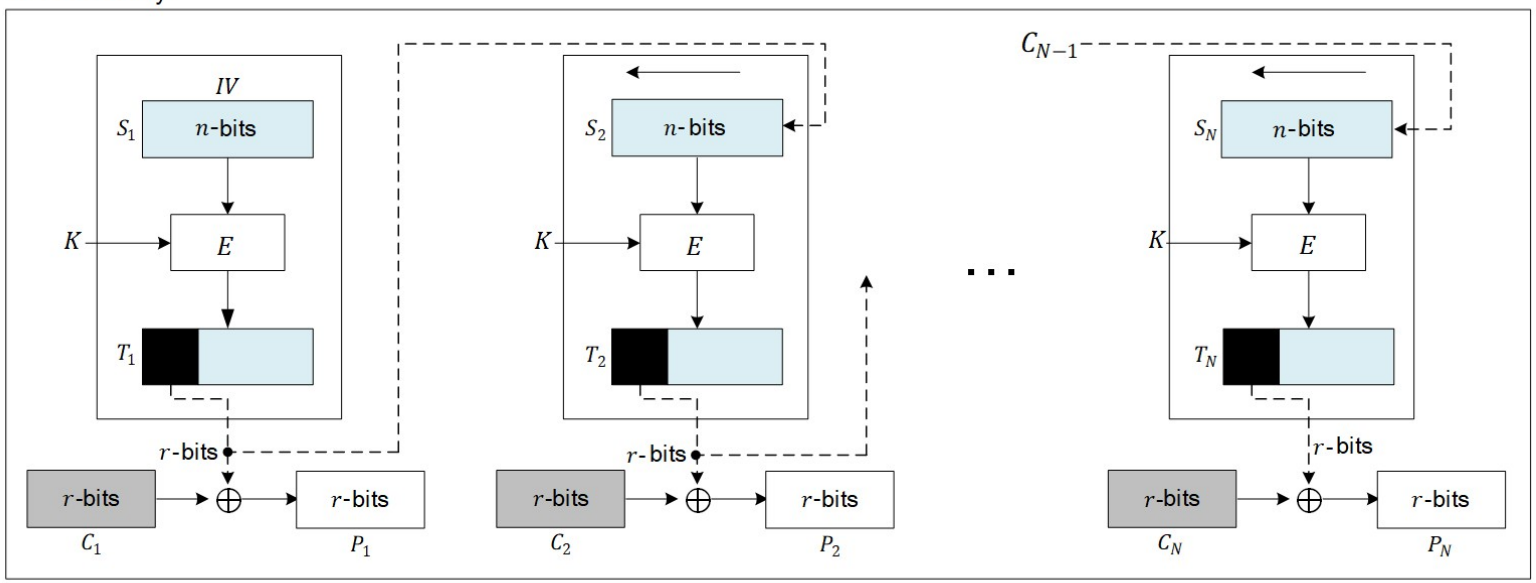
Encryption

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (3/7)
      - 동작 과정 (2/2)
        - 복호화 : $P_i = C_i \oplus SelectLeft_r\{E_K[ShiftLeft_r(S_{j-1})|SelectLeft_r(T_{i-1})]\}$
          - 복호화 시에도 암호화 함수를 사용함

ShiftLeft_r : r만큼 왼쪽 쉬프트
$SelectLeft_r$: 상위 r비트 선택

E : Encryption func C_i : Ciphertext block i S_i : Shift register
P_i : Plaintet block i IV : Initialization vector (C_0) T_i : Temporary register
K : Secret key

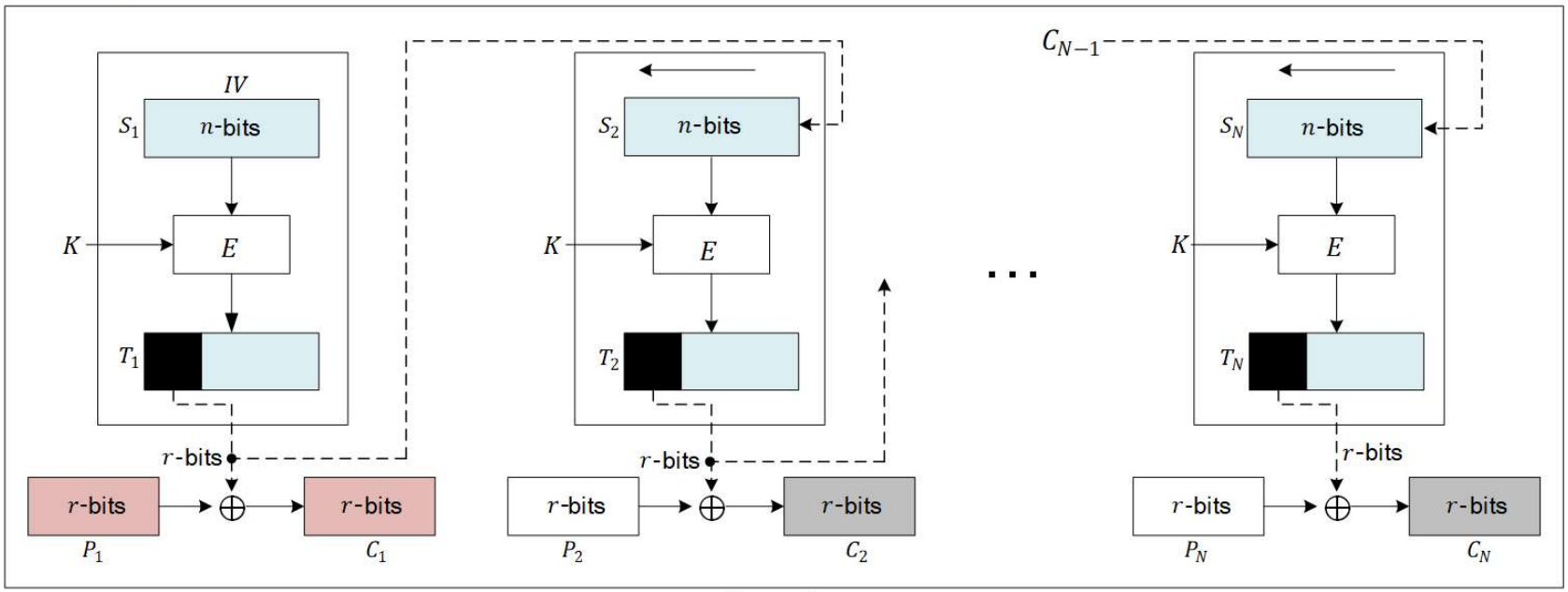


Decryption

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (4/7)
      - 오류 파급 (1/2)
        - 암호화
          - 한 평문 블록에 오류가 발생하면, 현재 암호문 블록에만 오류가 발생함

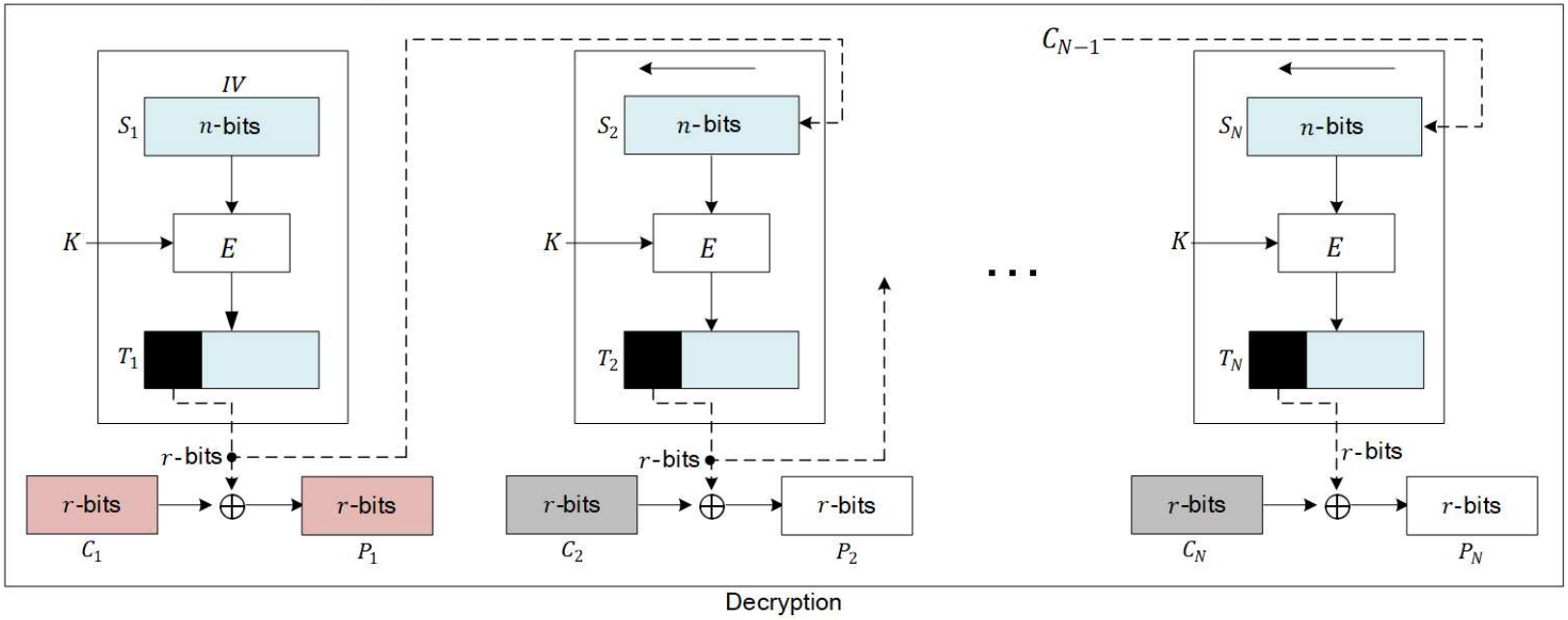
오류가 발생하는 부분은 붉은색 (■)으로 표시



현대 대칭-키 암호를 이용한 암호화 기법

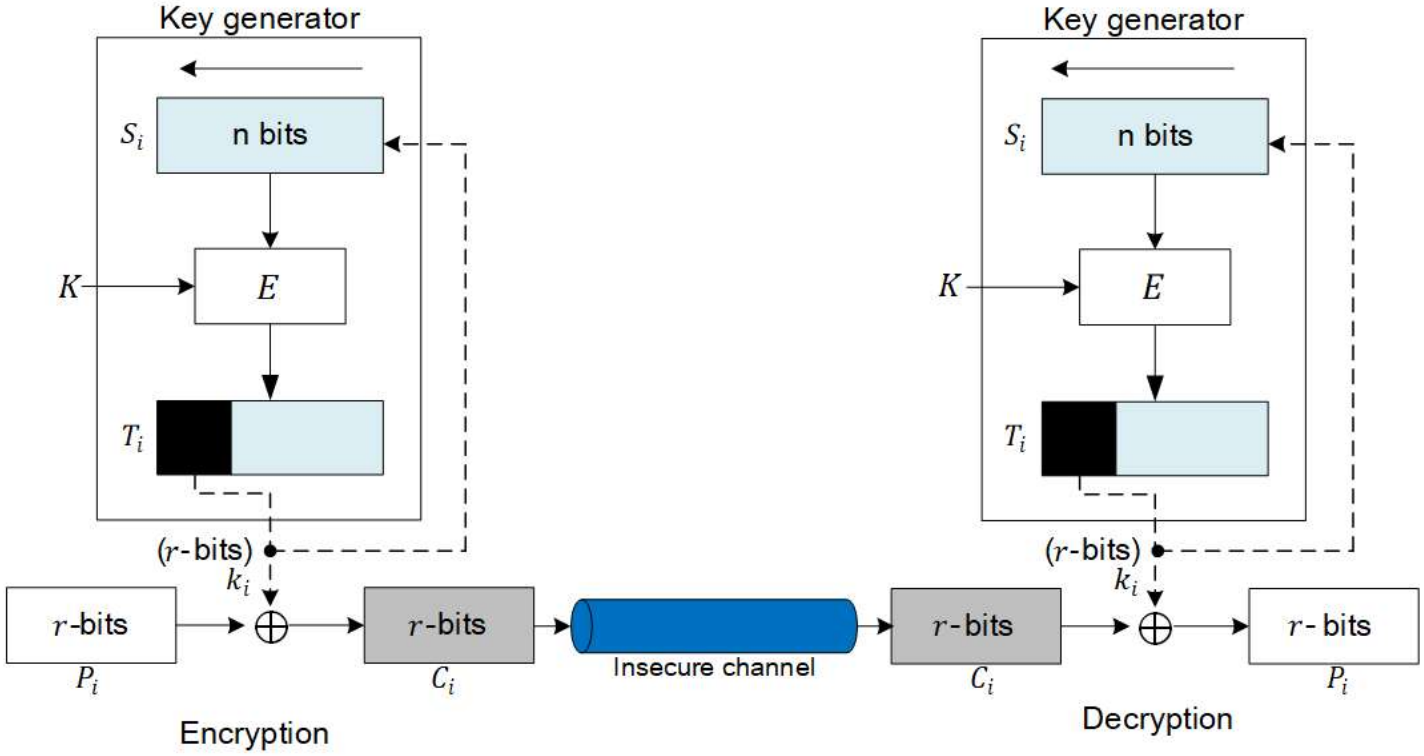
- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (5/7)
      - 오류 파급 (2/2)
        - 복호화
          - 한 암호문 블록에 오류가 발생하면, 현재 평문 블록에만 오류가 발생함

오류가 발생하는 부분은 붉은색 (■)으로 표시



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (6/7)
      - 스트림 암호로서의 OFB모드
        - 키 스트림이 평문이나 암호문과 독립이므로 동기식 스트림 암호임



현대 대칭-키 암호를 이용한 암호화 기법

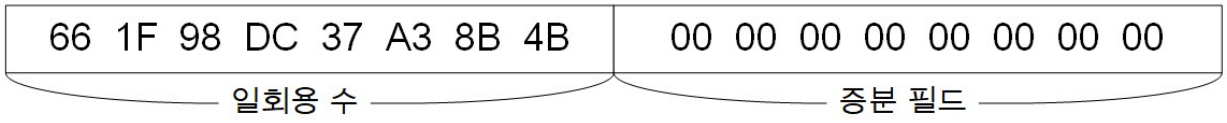
- 현대 블록 암호의 사용
  - 운영 모드
    - OFB(Output Feedback) 모드 (7/7)
      - 장점
        - 내부적으로 생성된 키스트림을 사용하므로 평문 패턴 노출이 없어 안전함
        - 암호문 에러 시에도 다른 평문 블록에 영향을 주지 않음
      - 단점
        - 암호화와 복호화 모두 병렬 처리가 불가능 하므로 처리 속도가 느림
        - 암호문의 특정 비트를 조작하면 해당 위치의 평문 비트가 정확하게 대응되어 변경됨
          - 공격자가 암호문 비트를 조작하여 복호화된 평문을 원하는 대로 변경할 수 있음
          - 메시지 위조, 평문 유추 등의 위협이 발생할 수 있음

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (1/8)
      - 정의
        - 카운터 값을 암호화해 만든 키 스트림을 평문과 XOR
      - 특징
        - 각 블록마다 고유한 카운터 값을 사용함
          - 블록 간 독립성이 존재함
          - 암호화와 복호화 모두 병렬 처리할 수 있음
        - n-비트 키스트림을 생성해 필요한 만큼만 잘라 사용할 수 있으므로 패딩이 필요하지 않음
        - 초기 카운터 값 설정을 위해 IV가 필요함
        - 블록 암호를 스트림 암호로 사용할 수 있음
          - 암호 알고리즘 자체는 블록 암호이나, n-비트 키스트림을 생성해 필요한 만큼만 잘라서 사용함
        - 카운터를 사용함

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (2/8)
      - 알고리즘
        - CTR 모드의 알고리즘은 부록 #1 참고
      - 카운터(Counter)
        - 각 블록마다 고유하게 증가하는 비 중복 수열 값
        - 일반적으로 다음의 구조를 가짐



66 1F 98 DC 37 A3 8B 4B 00 00 00 00 00 00 00 00 (초기값)

66 1F 98 DC 37 A3 8B 4B 00 00 00 00 00 00 00 01 (증분 필드+1)

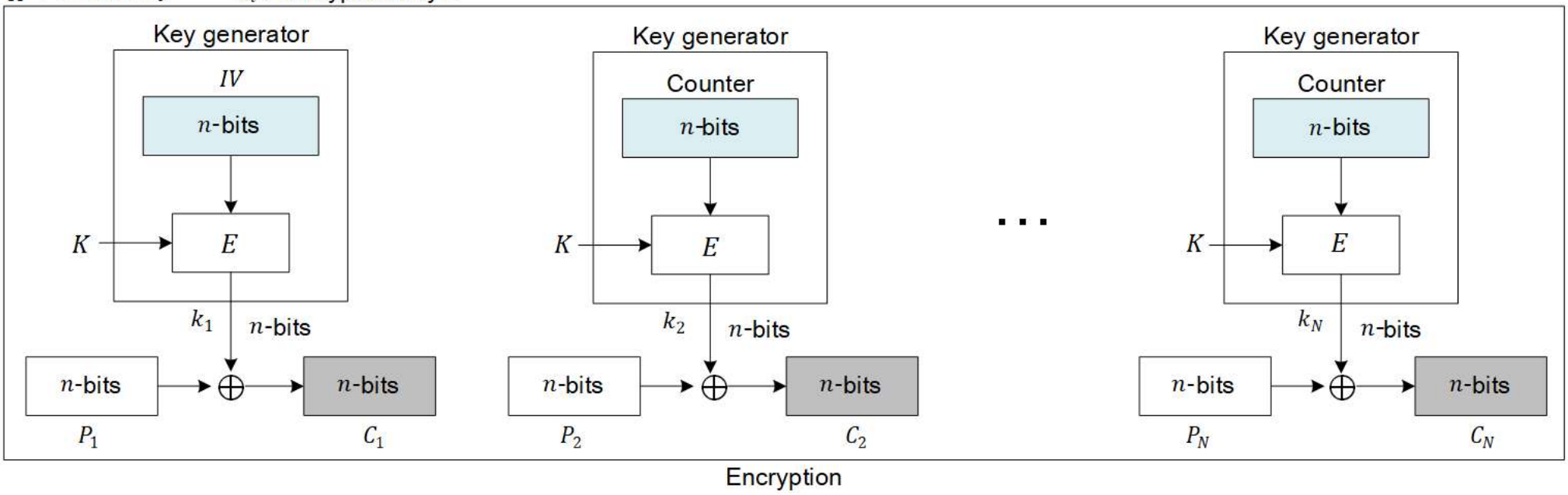
66 1F 98 DC 37 A3 8B 4B 00 00 00 00 00 00 00 02 (증분 필드+2)

.
.
.

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (3/8)
      - 동작 과정 (1/2)
        - 암호화 : $C_i = P_i \oplus E_{ki}(Counter)$

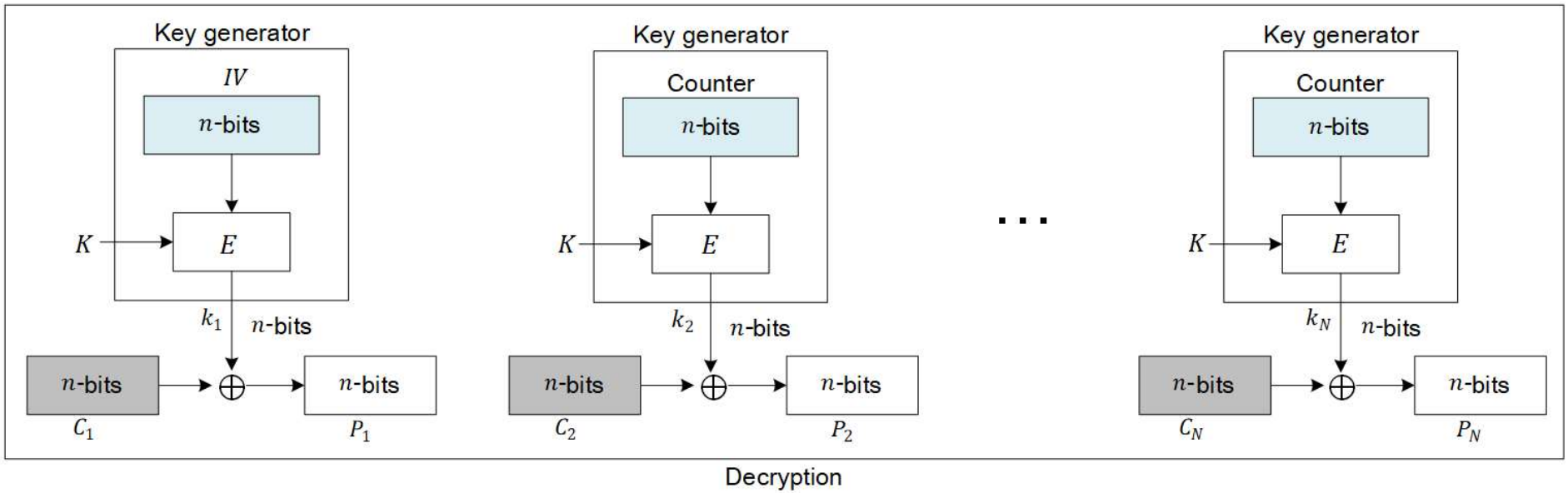
E : Encryption IV : Initialization vector
P_i : Plaintet block i C_i : Ciphertext block i
K : Secret key k_i : Encryption key i



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (4/8)
      - 동작 과정 (2/2)
        - 복호화 : $P_i = C_i \oplus E_{ki}(Counter)$
          - 복호화 시에도 암호화 함수를 사용함

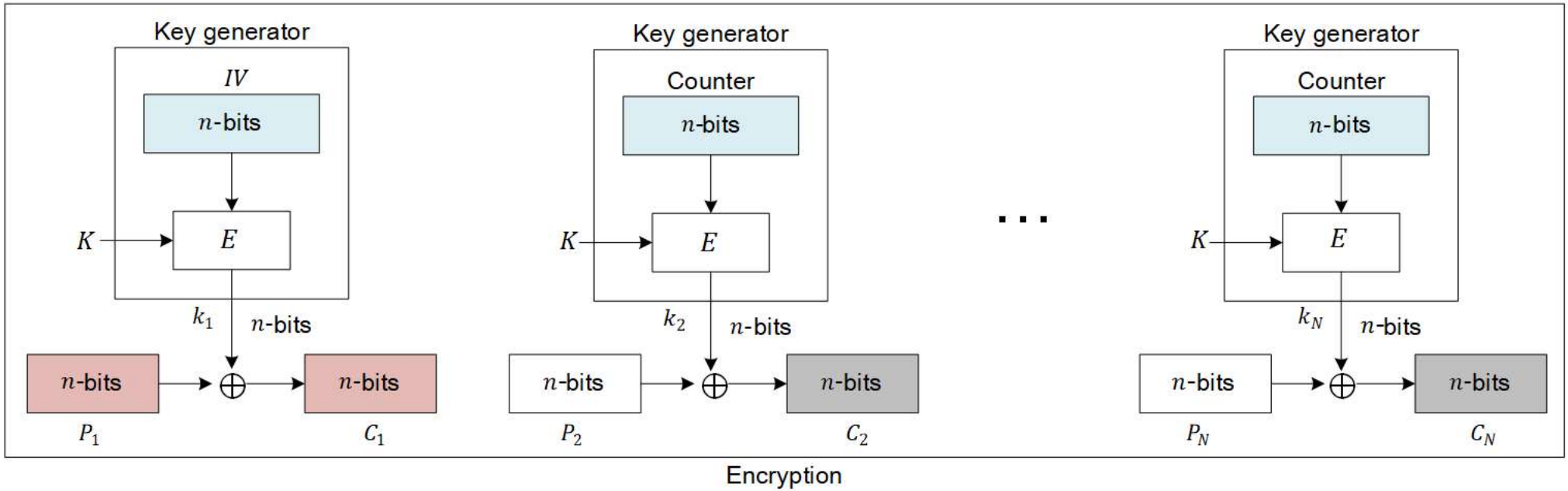
E : Encryption IV : Initialization vector
P_i : Plaintet block i C_i : Ciphertext block i
K : Secret key k_i : Encryption key i



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (5/8)
      - 오류 파급 (1/2)
        - 암호화
          - 한 평문 블록에 오류가 발생하면, 현재 암호문 블록에만 오류가 발생함

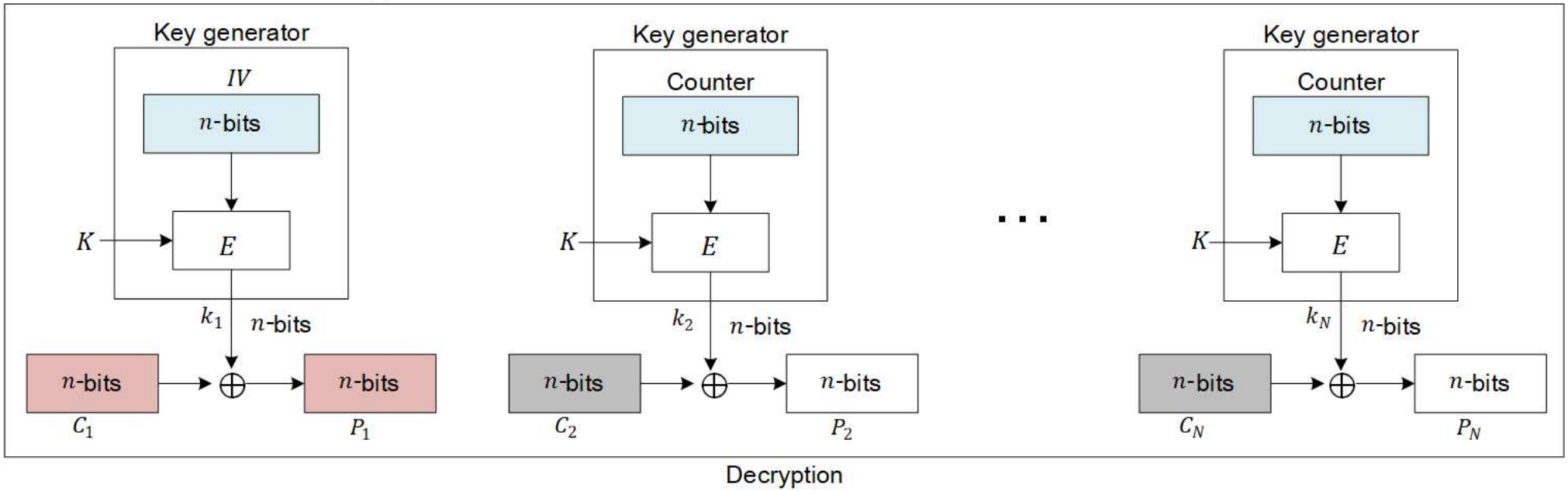
오류가 발생하는 부분은 붉은색(■)으로 표시



현대 대칭-키 암호를 이용한 암호화 기법

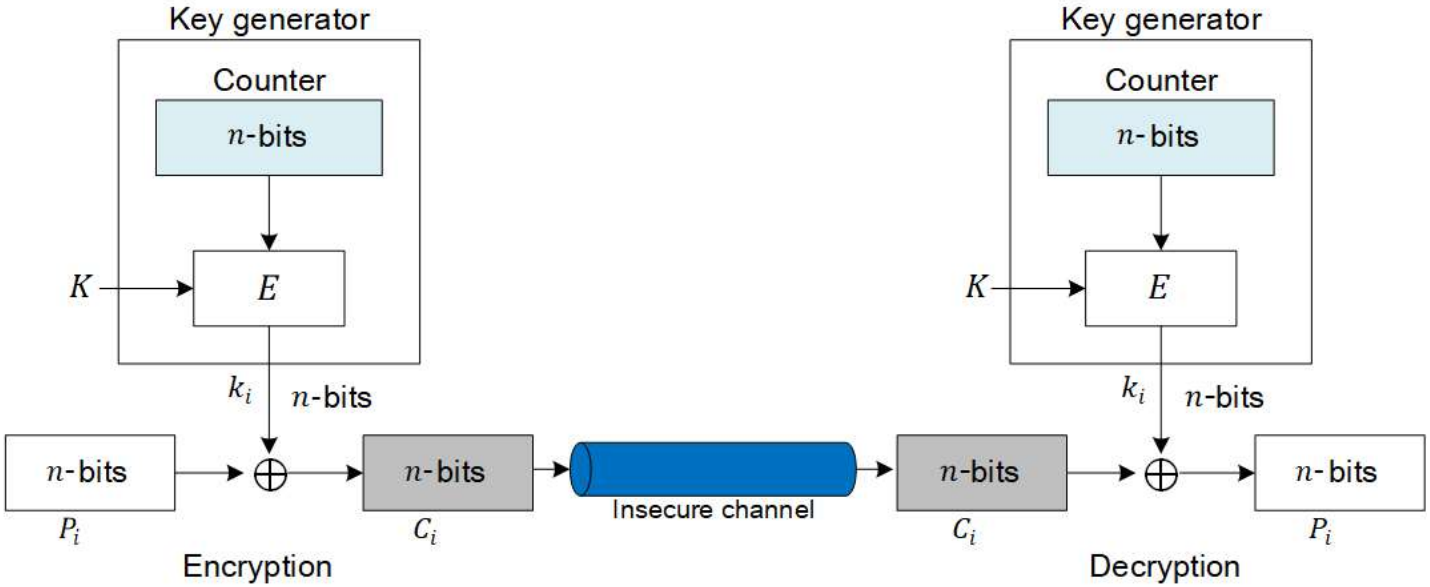
- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (6/8)
      - 오류 파급 (2/2)
        - 복호화
          - 한 암호문 블록에 오류가 발생하면, 현재 평문 블록에만 오류가 발생함

오류가 발생하는 부분은 붉은색 (■)으로 표시



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (7/8)
      - 스트림 암호로서의 CTR모드
        - 키 스트림이 평문이나 암호문과 독립이므로 동기식 스트림 암호임



현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드
    - CTR(Counter) 모드 (8/8)
      - 장점
        - 암호화 시 블록마다 고유한 카운터 값을 사용하므로 서로 다른 암호문 블록을 생성함
          - 패턴 노출이 방지되어 통계적 분석, 반복 기반 공격에 강함
        - 암호문 에러 시에도 다른 블록에 영향을 주지 않음
        - 암호화와 복호화 모두 병렬 처리가 가능하므로 처리 속도가 빠름
      - 단점
        - 암호문의 특정 비트를 조작하면 해당 위치의 평문 비트가 정확하게 대응되어 변경됨
          - 공격자가 암호문 비트를 조작하여 복호화된 평문을 원하는 대로 변경할 수 있음
          - 메시지 위조, 평문 유추 등의 위협이 발생할 수 있음

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드간 비교

운영모드	패딩 필요	IV사용	블록간 관계	병렬 처리	스트림 암호로 변경	동기식 / 비동기식	데이터 단위 크기
ECB	O	X	독립적	암호화, 복호화	X	-	n
CBC	O	O	암호문 의존	복호화	X	-	n
CFB	X	O	암호문 의존	복호화	O	비동기식 스트림 암호	$r \leq n$
OFB	X	O	이전 키스트림 의존	불가	O	동기식 스트림 암호	$r \leq n$
CTR	X	O	독립적	암호화, 복호화	O	동기식 스트림암호	n

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드간 비교
    - 데이터 저장 관점에서 적절한 운영모드
      - CTR 모드
        - 정확한 크기로 데이터를 저장할 수 있음
          - 패딩이 필요하지 않으므로 효율적이고 정확한 크기를 사용함
        - 원하는 블록에 접근할 수 있음
          - 각 블록이 고유한 카운터를 이용해 독립적으로 처리됨
        - 처리 속도가 빠름
          - 패딩이 필요하지 않음
          - 스트림 암호로 사용할 수 있음
          - 암호화와 복호화 모두 병렬 처리가 가능함

현대 대칭-키 암호를 이용한 암호화 기법

- 현대 블록 암호의 사용
  - 운영 모드간 비교
    - 데이터 전송 관점에서 적절한 운영모드
      - OFB 모드
        - 전송을 위한 실시간 처리에 적합함
          - 스트림 암호로 사용할 수 있음
          - 패딩이 필요하지 않음
        - 오류 전파 없음
          - 암호문에 오류가 생겨도 해당 블록에만 영향을 미침

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - 작은 단위의 데이터를 암호화 하기 위해 사용함
    - 블록 암호보다 빠른 속도를 가지며 실시간으로 처리할 수 있음
    - 종류
      - RC4
      - A5/1

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (1/7)
      - 개요
        - 바이트 단위로 작동하도록 설계된 스트림 암호
        - 다양한 크기의 키(1~256바이트)를 사용함
        - WEP(Wired Equivalent Privacy), WPA(Wi-Fi Protected Access)에서 사용됨
      - 알고리즘
        - 부록 #2 참고

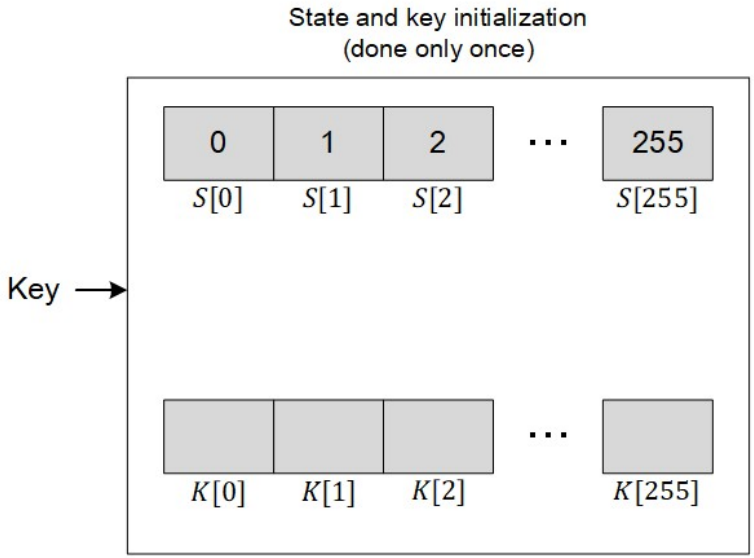
현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (2/7)
      - 바이트 단위 스트림
        - 평문 1바이트(8비트)와 키 1바이트가 XOR 되어 암호문 1바이트를 생성함
        - 키 스트림을 생성하는 키는 1~256바이트 사이의 어떤 값
      - 상태(state)
        - 256바이트로 구성된 상태의 한 바이트는 암호화 키로 사용하기 위해 랜덤하게 선택 됨
        - 원소의 인덱스는 0~255 사이의 값임
        - 원소의 항목은 0~255 사이의 1바이트 정수 값을 가짐
        - $S[0]$ $S[1]$ $S[2]$ ... $S[255]$

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (3/7) – 과정
      1. 상태 배열 S 초기화
         - $S[i]$는 0~255까지 순차적으로 초기화됨
         - $K[i]$는 256바이트가 채워질 때까지 Key를 반복적으로 저장함

```
for (i=0 to 255)
{
    S[i] <- i
    K[i] <- Key [i mod Key Length]
}
```

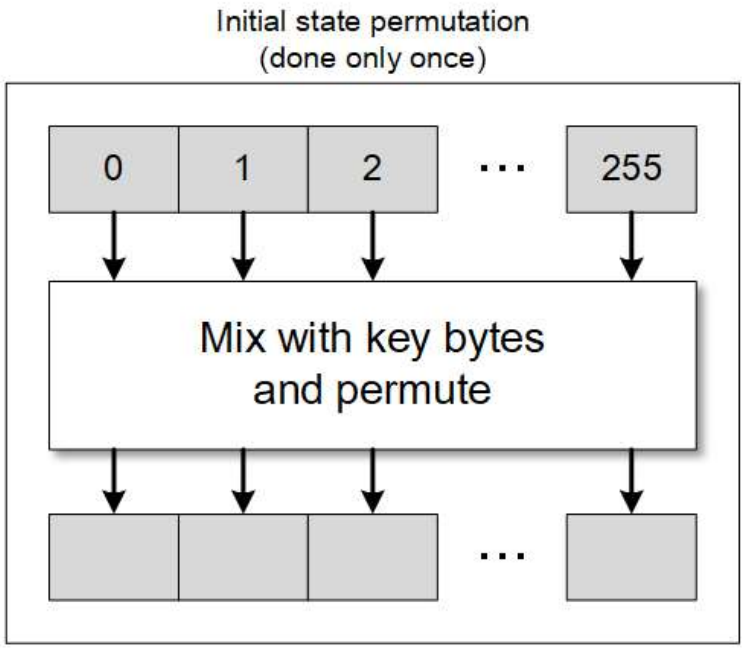


현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (4/7) – 과정

2. KSA(Key Scheduling Algorithm)를 통한 S의 초기 치환
   - K에 따라 $S[i]$를 S의 다른 바이트와 교환함
   - S는 완벽히 섞임

```
j <- 0
for(i=0 to 255)
{
    j<-(j+S[i]+K[i]) mod 256
    swap(S[i], S[j])
}
```



현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (5/7) – 과정

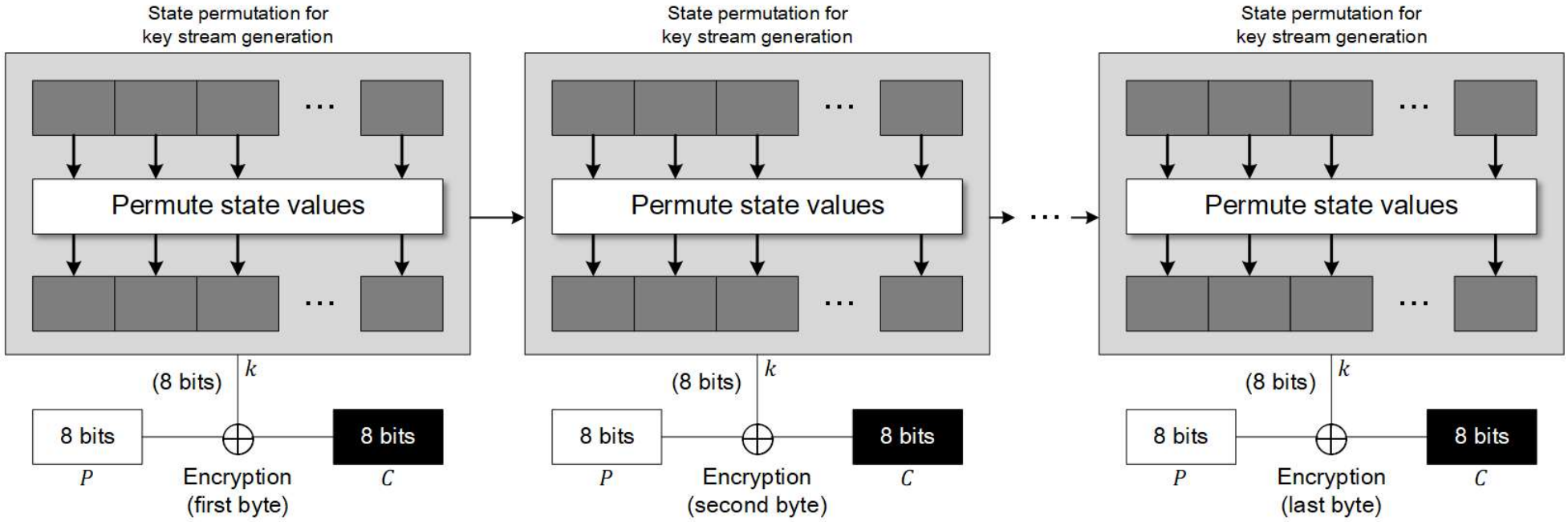
3. 키 스트림 생성
   - i값에 따른 j값을 계산함
   - $S[i]$와 $S[j]$를 교환하여 1개의 키 스트림 바이트를 생성함

```
i <- (i+1) mod 256
j <- (j + S[i] mod 256
swap (S[i], S[j])
k <- S[(S[i]+S[j]) mod 256]
```

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (6/7) – 과정

4. 암호화
   - 키 스트림 k와 평문 1바이트를 XOR함
   - 이를 반복하여 암호문을 획득할 수 있음

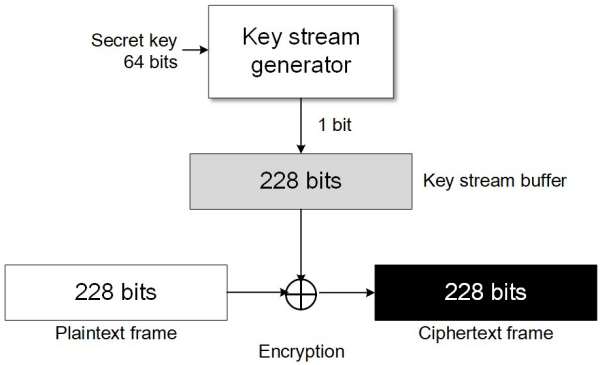


현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - RC4 (7/7)
      - RC4의 한계
        - KSA의 초기 편향
          - S-배열 초기화를 위한 KSA에서 충분하지 않은 랜덤성을 가짐
          - 키 스트림의 초기 바이트가 예측 가능하므로 통계적 분석에 취약함
        - 키 길이에 따른 안전성 문제
          - 이론상 128비트 이상의 키에서는 비교적 안전함
          - 실전에서는 KSA의 취약점 때문에 길이만으로 안전성 보장이 불가능함
          - 40/56/64 비트 등의 짧은 키는 전수조사 공격에 취약함
        - 차분 분석법에 대한 방어 한계
          - 여러 암호문 간의 통계 분석을 통해 키 바이트를 추측할 수 있음

현대 대칭-키 암호를 이용한 암호화 기법

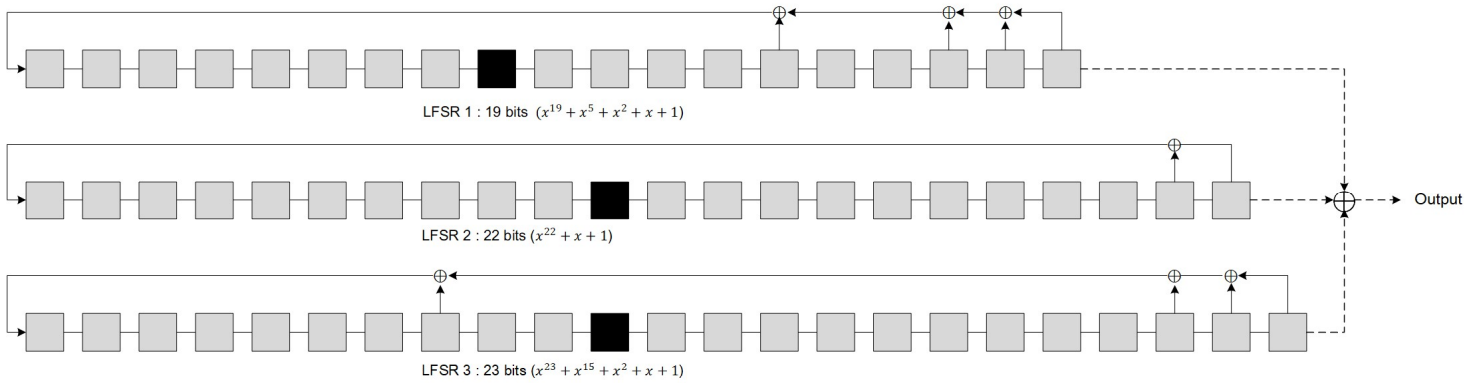
- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - A5/1 (1/5)
      - 개요
        - LFSR을 이용해서 키 스트림을 생성하는 스트림 암호
        - 64비트 키를 사용함
        - 휴대전화 통신을 위한 GSM(Global System for Mobile Communication) 에서 사용됨
          - LFSR가 선형 구조이므로 출력 비트를 통한 레지스터의 현재 값 유추할 수 있음
          - 불충분한 복잡도와 64비트의 작은 키 길이로 전수조사 공격에 취약함



현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - A5/1 (2/5) – 과정
      1. LFSR 초기화
         - 세 개의 LFSR 셀을 모두 0으로 초기화함
         - *클럭킹 비트를 설정함
         - 키와 LFSR을 XOR을 수행함
         - 프레임 번호와 LFSR을 XOR을 수행함
      2. 시프트 레지스터 동기화
         - 100 사이클 동안 전체 클럭킹을 수행함
         - 키와 프레임 번호를 확산시킴

*클럭킹(Clocking)
각각의 LFSR이 한 개의 셀만큼 이동하는 과정



현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - A5/1 (3/5) – 과정
      3. 키 스트림 생성
         - 클럭킹 비트를 사용하여 majority 비트를 계산함
         - majority 비트와 같은 클럭킹 비트를 가지는 LFSR만 선택적으로 클러킹을 수행함
         - 각 LFSR의 오른쪽 최상위 비트에 따른 키 스트림을 생성함

         - Majority 함수
           - 2개 이상의 비트가 1이면 $Majority(b_1, b_2, b_3)$는 1을 출력함
           - 2개 이상의 비트가 0이면 $Majority(b_1, b_2, b_3)$는 0을 출력함
           - LFSR 이동 전에 매번 값을 가짐
           - 오른쪽 최상위 비트를 0번째 라고 하면, LFSR1[10], LFSR2[11], LFSR3[11] 비트를 클럭킹 비트라고 부름

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - A5/1 (4/5) – 과정

      4. 암호화
         - 키 스트림을 버퍼에 저장함
         - 228 비트 프레임과 XOR을 수행함

현대 대칭-키 암호를 이용한 암호화 기법

- 스트림 암호의 사용
  - 스트림 암호(Stream Cipher)
    - A5/1 (5/5)
      - 취약성
        - 시간-기억 복합 공격(Time-Memory Trade-Off)
          - 사전 계산된 테이블과 평문-암호문 쌍을 이용해 키를 추출함
          - A5/1의 내부 상태 공간이 충분히 크지 않고, LFSR 상태 복원이 빠르다는 특징을 활용함
          - 실제 사례 (2000년)
            - 작은 기지 평문으로부터 1분 안에 키를 찾을 수 있는 실시간 공격을 수행함
        - 상태 추론 공격 (State Guessing Attack)
          - 수 분 안에 짧은 평문 조각으로 키를 역 연산할 수 있음
          - A5/1의 레지스터 갱신 규칙이 불균형하고 예측 가능하다는 특징을 활용함
          - 실제 사례 (2003년)
            - 2분에서 5분 사이의 평문을 이용하여 수분 안에 A5/1을 깰 수 있는 공격을 수행함

현대 대칭-키 암호를 이용한 암호화 기법

- 키 관리와 키 생성
  - 키 관리
    - 대칭-키 암호를 이용하여 안전하게 통신하기 위해 비밀 키를 공유해야 함
    - n명의 구성원의 경우 $n(n-1)/2$개의 비밀키가 필요함
    - 해결책
      - 사용자가 통신하기 원할 때 마다, 사용자 사이의 세션(임시) 키를 만들어야 함
      - 하나 혹은 그 이상의 키 분배 기관이 구성원에 대한 세션 키를 분배하기 위하여 커뮤니티를 설립 해야 함

현대 대칭-키 암호를 이용한 암호화 기법

- 키 관리와 키 생성
  - 키 생성
    - 키의 선택은 비밀요소가 누출 되는 것을 막기 위하여 체계적인 접근에 기반을 두어야 함
    - 난수 생성기의 필요
      - 사용자 간의 세션 키를 생성한다면 공격자가 다음 키를 추측할 수 없도록 랜덤하게 선택 해야 함
      - 키 분배 기관이 키 분배를 원한다면 상호간에 할당된 키를 추측할 수 없도록 랜덤하게 선택 되어야 함

Thanks!

송 민 희(23011716@sejong.ac.kr)

부록#1

- 운영 모드의 알고리즘 (1/4)
  - ECB(Electronic Codebook)

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

```
ECB_Decryption (K, Ciphertext blocks)
{
    for(i = 1 to N)
    {
        P_i <- D_k(C_i)
    }
    return Plaintext blocks
}
```

  - CBC(Cipher Block Chaining)

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

```
CBC_Decryption(K, IV, Ciphertext blocks)
{
    C_0 <- IV
    for(i = 1 to N)
    {
        Temp <- D_k(C_i)
        P_i <- Temp ⊕ C_{i-1}
    }
    return Plaintext blocks
}
```

부록#1

- 운영 모드의 알고리즘 (2/4)
  - CFB(Cipher Feedback)

```
CFB_Encryption(IV, K, r)
{
    i <- 1
    while(more blocks to encrypt)
    {
        input(P_i)
        if(i = 1)
            S <- IV
        else
        {
            Temp <- shiftLeft_r(S)
            S <- concatenate(Temp, C_{i-1})
        }
        T <- E_K(S)
        k_i <- selectLeft_f(T)
        C_i <- P_i ⊕ k_i
        output(C_i)
        i <- i + 1
    }
}
```

```
CFB_Decryption(IV, K, r)
{
    i <- 1
    while(more blocks to decrypt)
    {
        input(C_i)
        if(i = 1)
            S <- IV
        else
        {
            Temp <- shiftLeft_r(S)
            S <- concatenate(Temp, C_{i-1})
        }
        T <- E_K(S)
        k_i <- selectLeft_f(T)
        P_i <- C_i ⊕ k_i
        output(P_i)
        i <- i + 1
    }
}
```

부록#1

- 운영 모드의 알고리즘 (3/4)
  - OFB(Output Feedback)

```
OFB_Encryption(IV, K, r)
{
    i <- 1
    while(more blocks to encrypt)
    {
        input(P_i)
        if(i = 1)
            S <- IV
        else
        {
            Temp <- shiftLeft_r(S)
            S <- concatenate(Temp, k_{i-1})
        }
        T <- E_K(S)
        k_i <- selectLeft_r(T)
        C_i <- P_i ⊕ k_i
        output(C_i)
        i <- i + 1
    }
}
```

```
OFB_Decryption(IV, K, r)
{
    i <- 1
    while(more blocks to decrypt)
    {
        input(C_i)
        if(i = 1)
            S <- IV
        else
        {
            Temp <- shiftLeft_r(S)
            S <- concatenate(Temp, k_{i-1})
        }
        T <- E_K(S)
        k_i <- selectLeft_r(T)
        P_i <- C_i ⊕ k_i
        output(P_i)
        i <- i + 1
    }
}
```

부록#1

- 운영 모드의 알고리즘 (4/4)
  - CTF(Counter)

```
CTR_Encryption(IV, K, Plaintext blocks)
{
    Counter <- IV
    for(i = 1 to N)
    {
        Counter <- (Counter + i - 1) mod 2^N
        k_i <- E_K(Counter)
        C_i <- P_i ⊕ k_i
    }
    return Ciphertext blocks
}
```

```
CTR_Decryption(IV, K, Ciphertext blocks)
{
    Counter <- IV
    for(i = 1 to N)
    {
        Counter <- (Counter + i - 1) mod 2^N
        k_i <- E_K(Counter)
        P_i <- C_i ⊕ k_i
    }
    return Plaintext blocks
}
```

부록#2

- 스트림 암호의 알고리즘
  - RC4

```
RC4_Encryption(K)
{
    // Creation of initial state and key bytes
    for (i = 0 to 255)
    {
        S[i] ← i
        K[i] ← Key[i mod KeyLength]
    }

    // Permuting state bytes based on values of key bytes
    j ← 0
    for (i = 0 to 255)
    {
        j ← (j + S[i] + K[i]) mod 256
        swap(S[i], S[j])
    }

    // Continuously permuting state bytes, generating keys, and encrypting
    i ← 0
    j ← 0
    while (more byte to encrypt)
    {
        i ← (i + 1) mod 256
        j ← (j + S[i]) mod 256
        swap(S[i], S[j])
        k ← S[(S[i] + S[j]) mod 256]

        // Key is ready, encrypt
        input P
        C ← P ⊕ k
        output C
    }
}
```