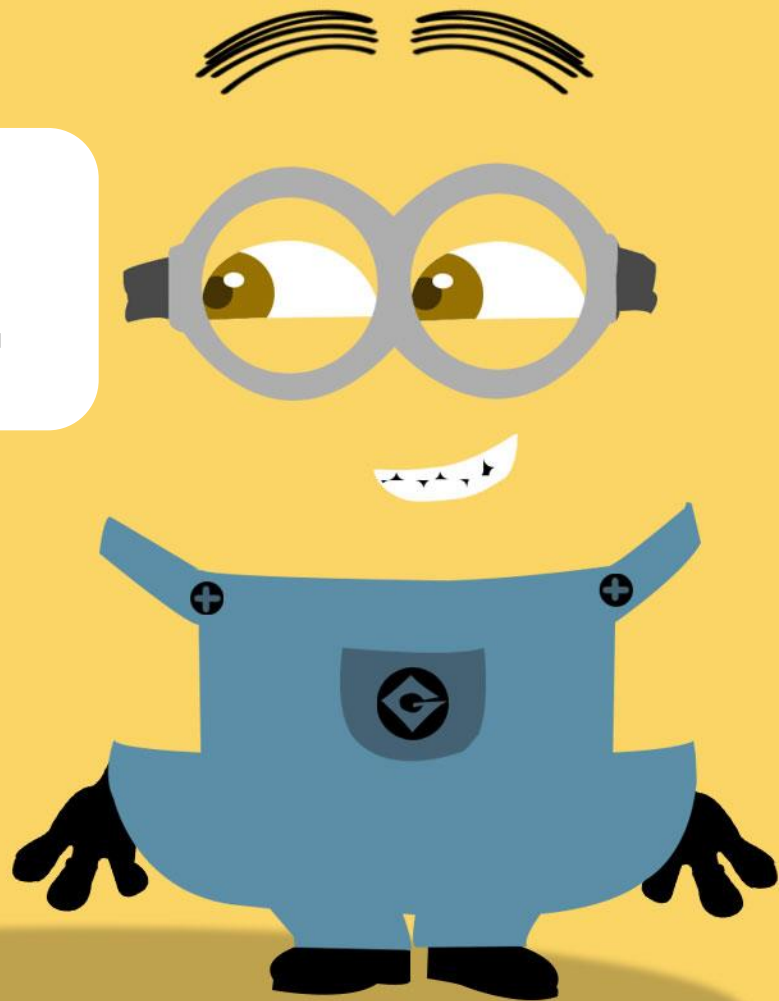




경량 블록 암호





목적

Korea

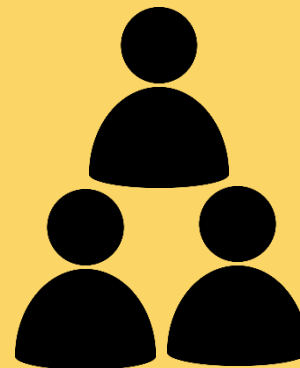


다수의 국산
경량블록암호 존재함

?



경량블록암호와 관
련된 자료가 분산되
어 있음



중·고등학생들을
위한 경량블록암호
관련 자료가 없음

중,고등학생들을 위한 국산 경량블록암호 학습 자료의
필요성 느낌

1. 들어가기
2. 경량 블록 암호
3. CHAM
4. LEA
5. HIGHT
6. CHAM, LEA, HIGHT 성능 비교





들어가기..

<https://bit.ly/2ooC3ow>





블록 암호란?

암호화 하기 전, 평범한 문장

암호화 과정 반대=복호화 과정

평문을 정해진 블록 단위로 암호화 하는 대칭키 암호 시스템

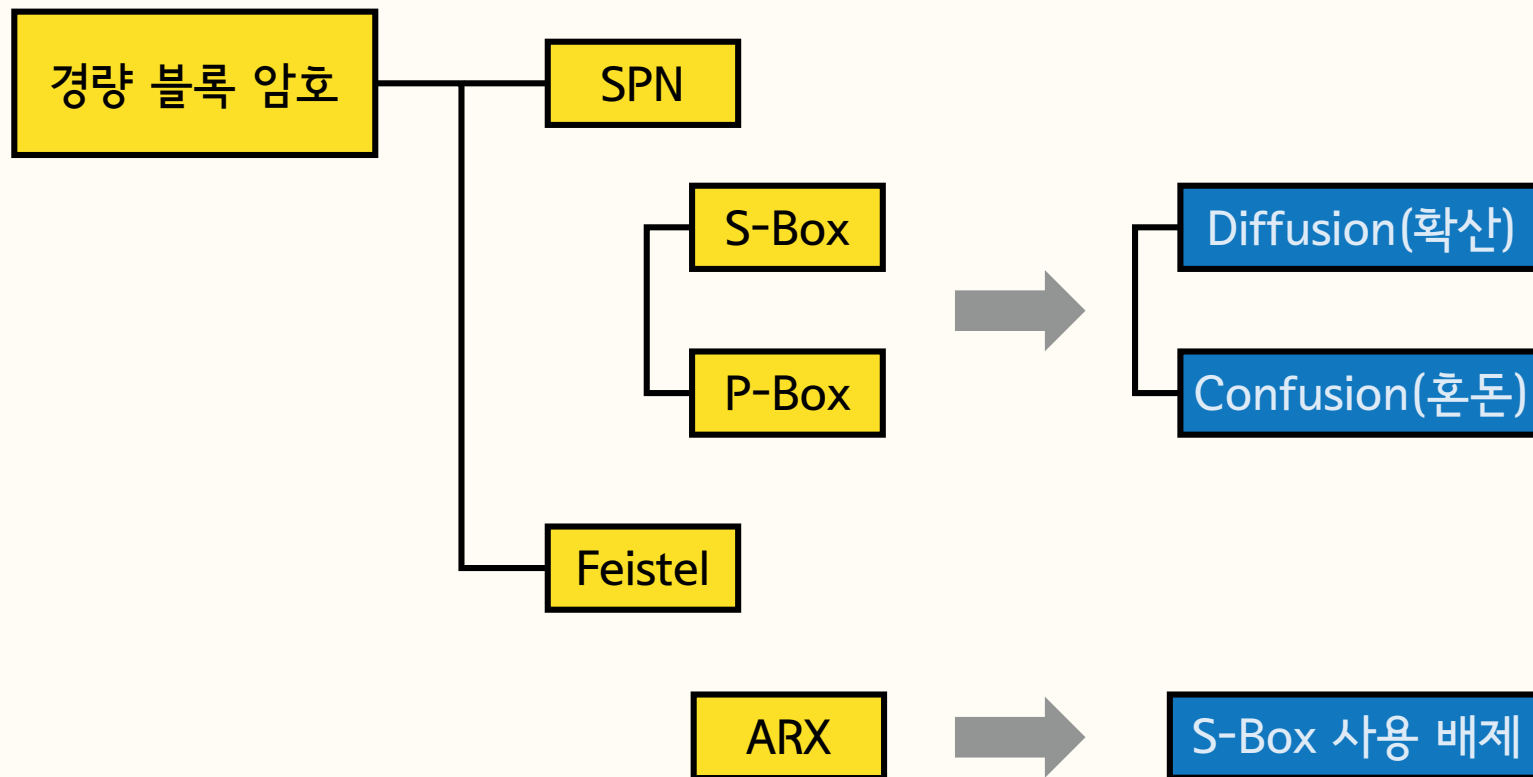
구분	블록 암호
장점	높은 확산, 기밀성, 해시함수 등 다양
단점	암호 속도가 느림, 에러 전파현상 있음
단위	블록(block)
사례	DES,AES,IDEA,SEED,RC5
대상	일반 데이터 전송, 스토리지 저장

(표 1) 블록 암호 설명





경량 블록 암호란?





SPN 구조

S-Box

Substitution boxes (치환 상자)

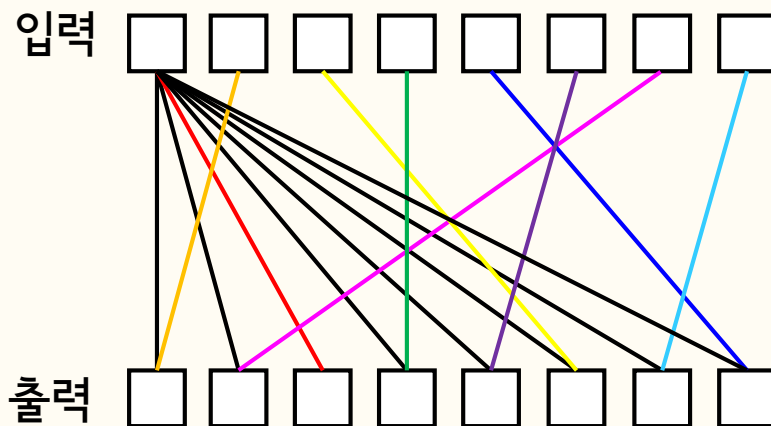
S_i		중간 비트			
		0 (00)	1 (01)	2 (10)	3 (11)
외측 비트	0 (00)	1	0	3	2
	1 (01)	3	2	1	0
	2 (10)	0	2	1	3
	3 (11)	3	1	3	2

중간비트 : 열
ex) 001 → 3
외측비트 : 행

P-Box

Permutation boxes (순열 상자)

nP_r





SPN 구조

Diffusion(확산)

- 암호문-평문 사이의 관계를 숨겨줌
- 평문이 단 한글자라도 바뀌면, 암호문 비트가 모두 바뀜

Confusion(혼돈)

- 암호문-키 사이의 관계를 숨겨줌
- 키의 비트가 바뀌면, 암호문 비트가 모두 바뀜





Feistel 암호

Feistel 암호란?

(평문→암호문)

- 동일한 치환을 반복하면서 암호화되는
반복블록암호
- 주로 작은 IoT 기기에 사용됨
- ARX를 기반으로 함





Feistel 암호 (=ARX Based)

ARX

modular **A**ddition

- 더하기(addition)한 후 2^{bit} 로 나눈 나머지
- $x \boxplus y = (x \% 2^{32}) + (y \% 2^{32})$ (x, y 는 32비트열)

bitwise **R**otation

- 비트열을 규칙적으로 옮겨준다.
- $ROR_i(x)$: 비트열을 오른쪽으로 옮겨준다.
- $ROL_i(x)$: 비트열을 왼쪽으로 옮겨준다.

bitwise **X**OR

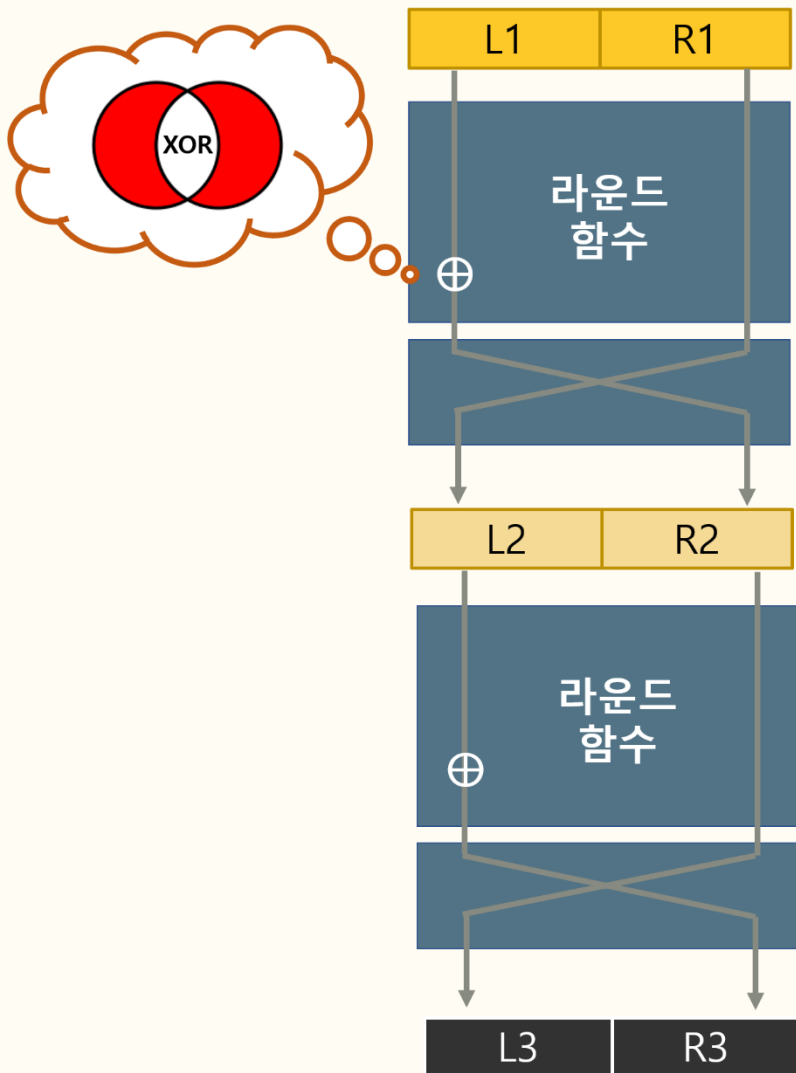
- $x \oplus y = (x \cup y) - (x \cap y)$

파이스텔



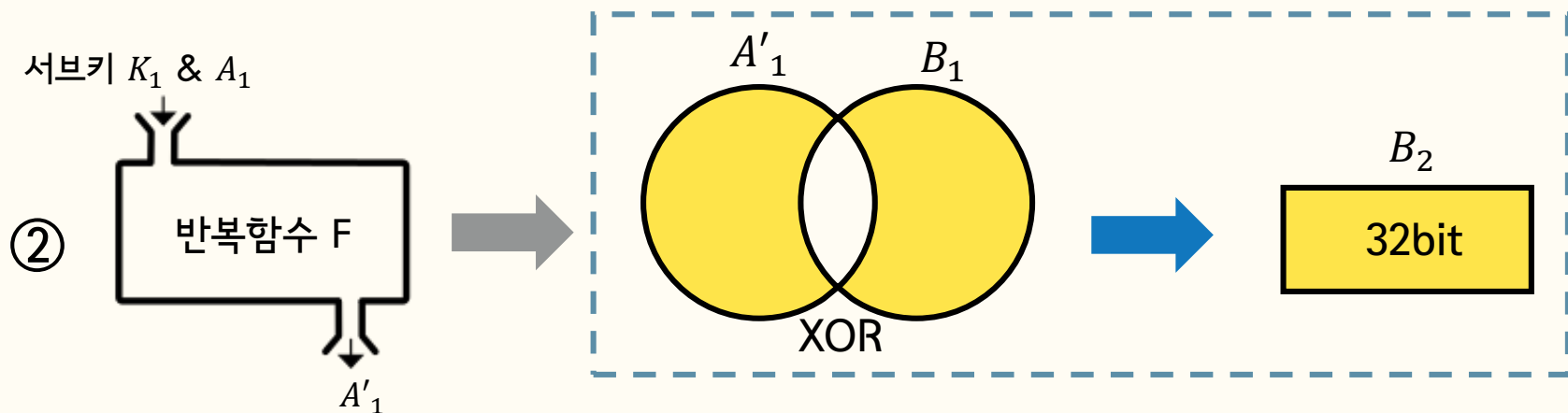
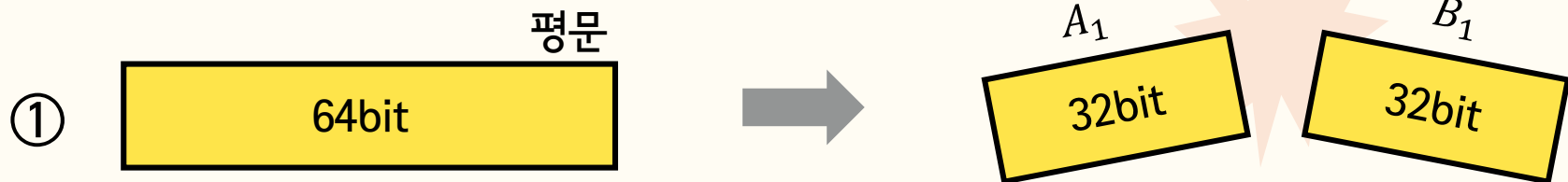


Feistel 암호



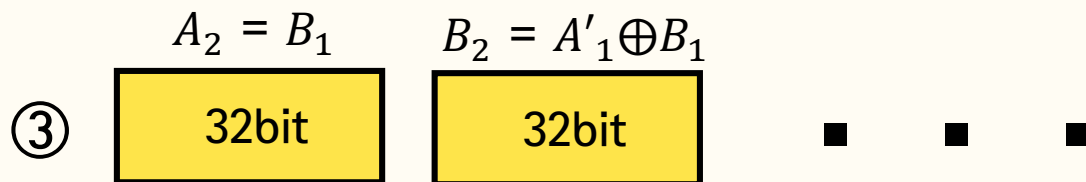


Feistel 암호

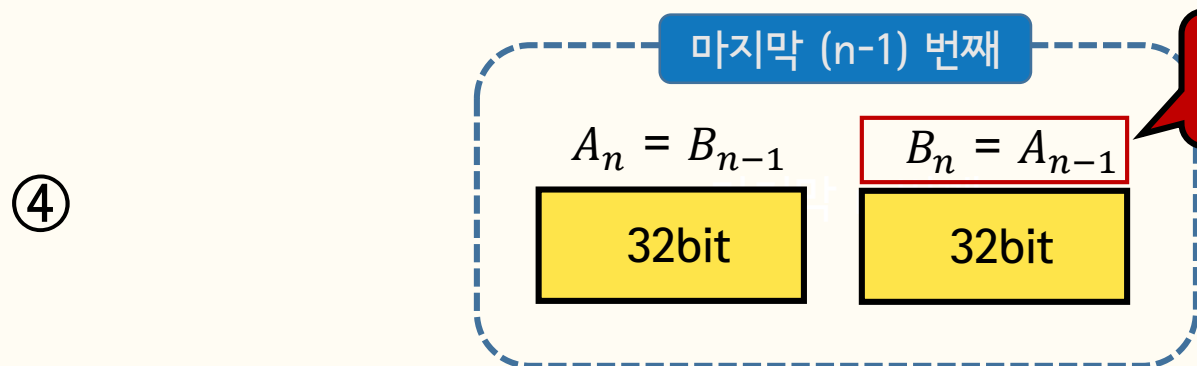




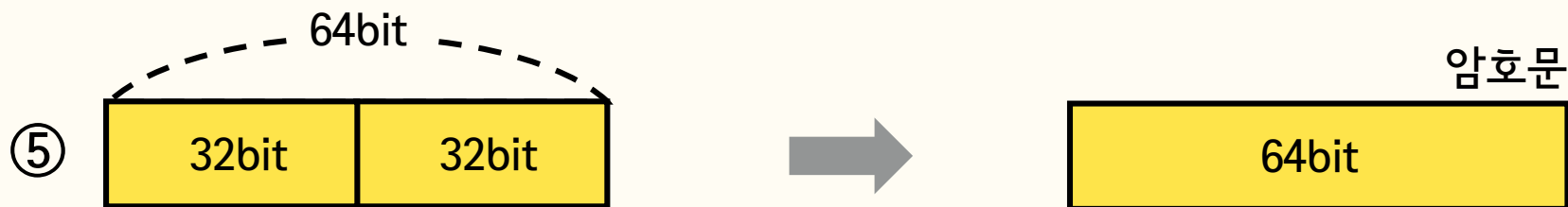
Feistel 암호



앞 과정(①~③) 반복



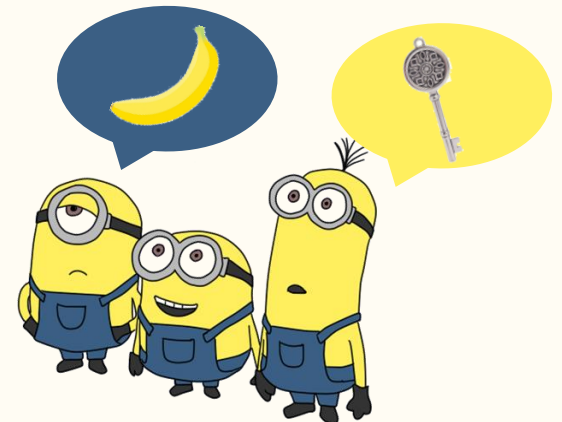
XOR 연산 하지 않음





CHAM

Stateless 기반 round key를 사용하는 경량 암호 알고리즘





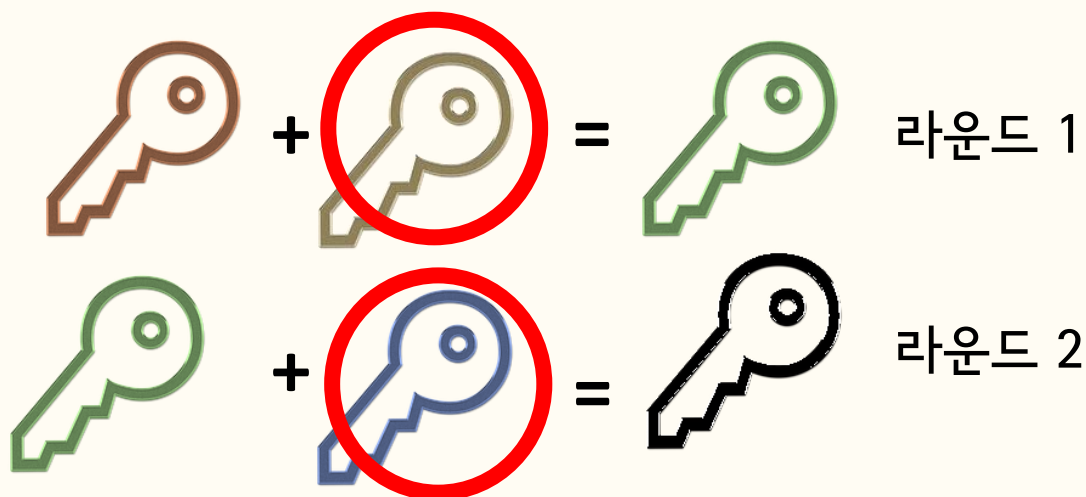
CHAM

👁 Stateless 기법

키의 상태를 저장하지 않는 기법

👁 라운드 키 (Round key)

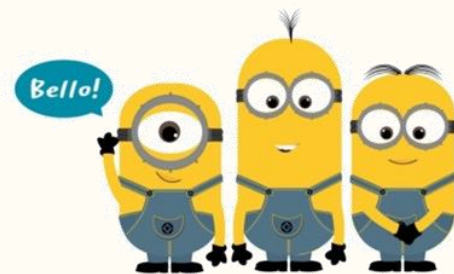
암호화 과정 중 키 덧셈 과정은 키의 변형된 형태를 더하는 과정
-> 각 라운드마다 더하는 키가 다름
이를 **라운드 키**라고 함





① 키 상태를 업데이트 하지 않아도 되는 매우 간단한 키 스케줄

- 하드웨어에서 암호 구현 시
플립플롭의 수 줄이는 것을 도와줌





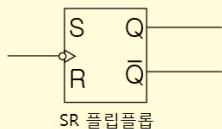
키 스케줄(stateless-on-the-fly 구현)

- 지정된 비밀 키에서 원형 키를 계산하는 절차
- 다른 블록 암호(ex. AES 등)는 현 키상태에서 각각의 라운드 키를 계산하고 키 상태를 업데이트하는 함수로 구성

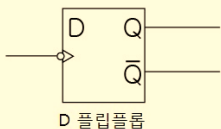


플립플롭(Flip-Flop)

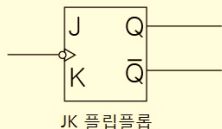
- 1 비트의 정보를 보관, 유지할 수 있는 회로
- 이전 상태를 계속 유지하여 저장



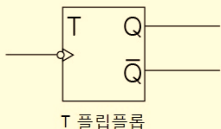
SR 플립플롭



D 플립플롭



JK 플립플롭



T 플립플롭





② 회전 수보다 라운드 키의 수가 매우 적음

- 라운드 키를 저장하는 메모리 수를 줄여줌





③ 부호 매김은 두 개의 타입을 사용 (1비트, 8비트)

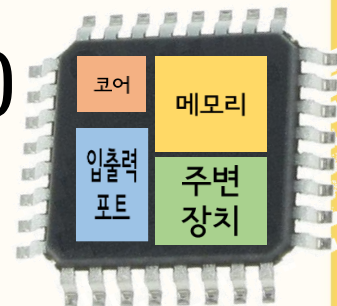
- 8비트 AVR 마이크로컨트롤러의 연산 숫자를 줄여줌





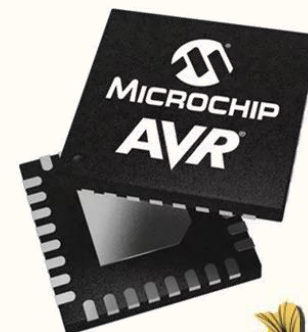
마이크로컨트롤러 (MCU : Micro Controller Unit)

- 지능화와 소형화를 위하여 마이크로 프로세서에 메모리와 각종 주변장치들을 함께 집적하여 넣은 칩



AVR(Automatic Voltage Regulator) MCU

- 8비트 제어용 마이크로프로세서





④ 슬라이드 공격과 회전식 공격을 막기 위해 랜덤 값 대신에 **라운드 인덱스**를 이용

- 라운드 인덱스를 통해 n번째 라운드에서는 같은 비밀키를 사용하지 못하도록 조정





슬라이드 공격

일반적인 생각(“취약한 암호라도 라운드 수를 늘려 차이를 막을 수 있다.”)을
다루기 위해 고안된 공격의 한 형태

회전식 공격

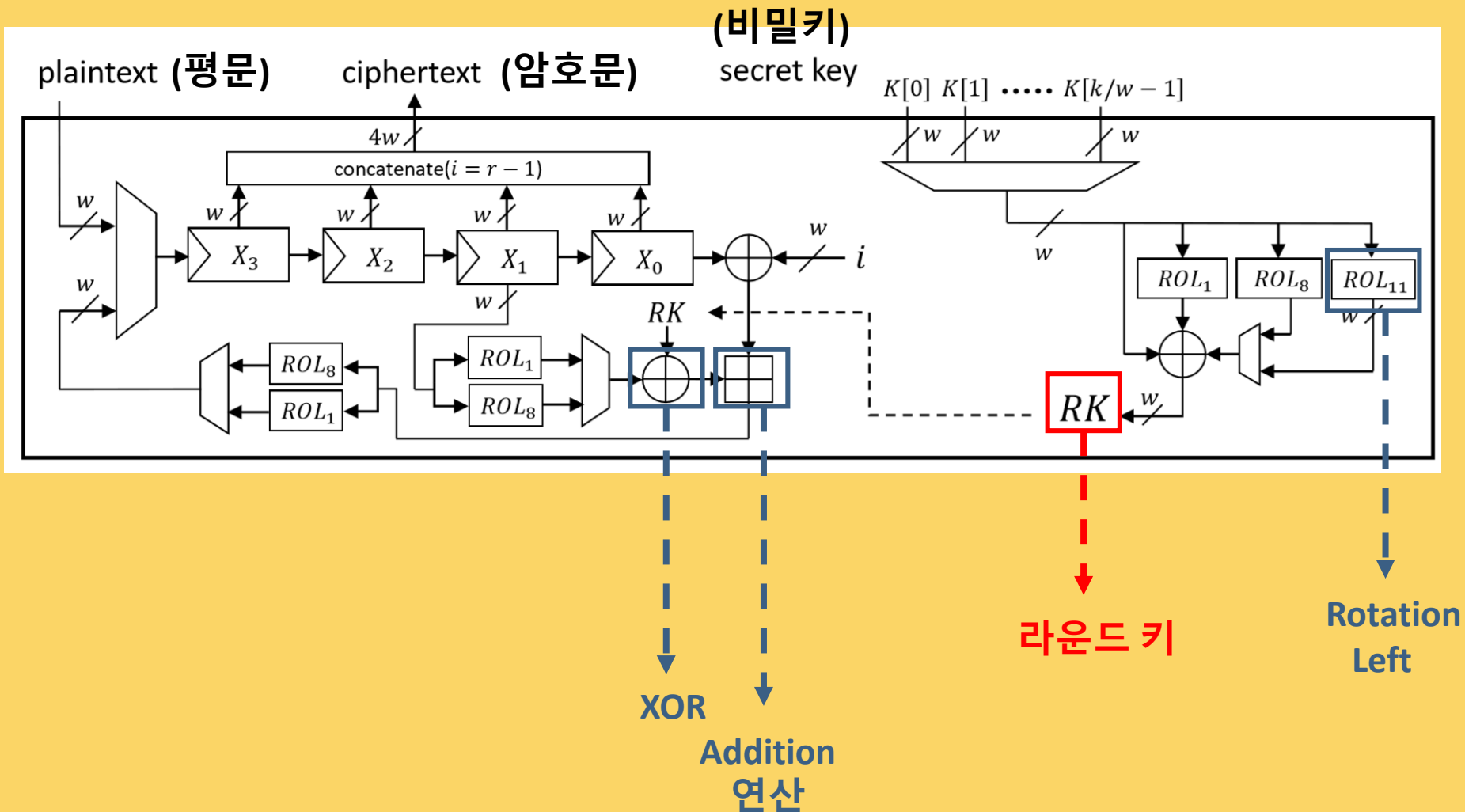
Addition, Rotation, XOR

ARX 세가지 작업에 의존하는 알고리즘에 대한
일반적인 공격





Round 기반 하드웨어 구조







CHAM

종류

종류	블록 크기	키 크기	라운드 수	워드 비트 길이	키 크기/ 비트 길이
CHAM-64/128	64	128	80	16	8
CHAM-128/128	128	128	80	32	4
CHAM-128/256	128	256	96	32	8

(표 3) CHAM 종류

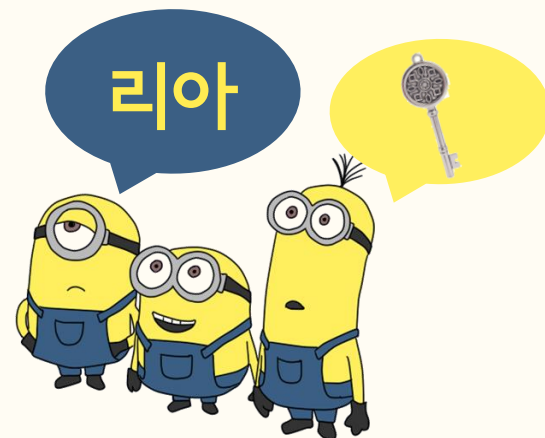
단위 : 비트






LEA

고속환경&경량 환경에서
사용하기 위해 개발된
128비트 **블록 암호 알고리즘**





LEA

 Lightweight Encryption Algorithm
경량 암호 알고리즘

 2013년 '국가 보안 기술 연구소'에서 개발함

 GFN(Generalized Feistel Network) 구조





LEA

특징 ①



고속환경



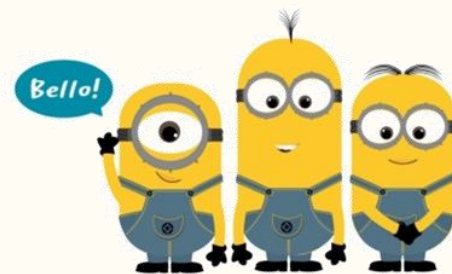
빅데이터, 클라우드



경량환경



모바일 기기





여러가지 기능을 제공하는 공통 실행환경
(ex. Windows, Chrome, Internet Explorer)

32-bit / 64-bit **플랫폼**에
최적화되어 있음

	LEA-128	LEA-192	LEA-256
평문/암호문 블록 길이	128 bit		
비밀키 길이	128 bit	192 bit	256 bit
라운드함수 반복횟수	24	28	32

(표 2) LEA 종류

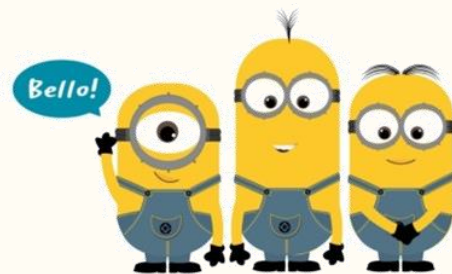


암호화를 할 때, 오직 **ARX 연산**만 사용함

👤 코드 길이가 짧음

👤 S/W 암호화 속도가 빠름

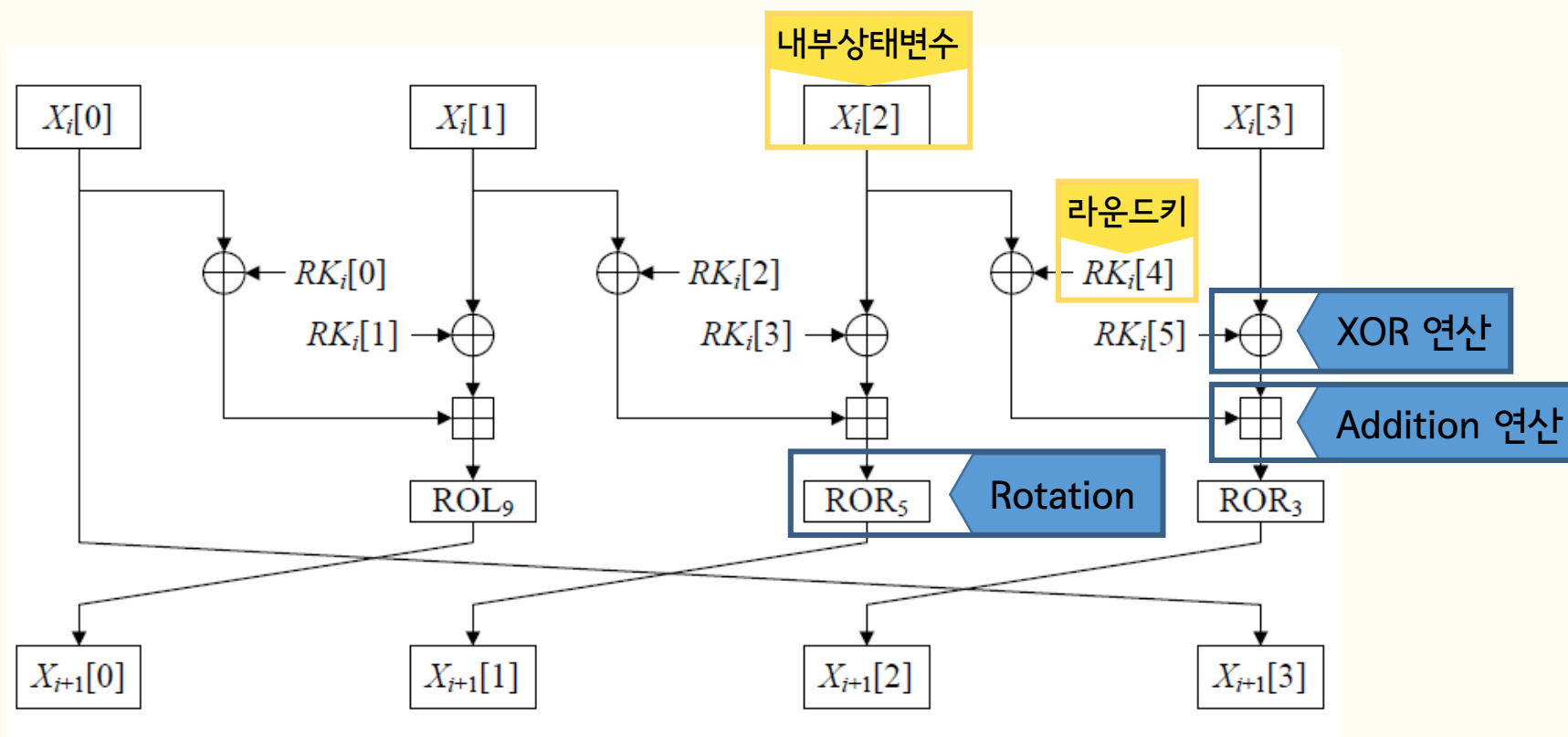
👤 다양한 공격(부메랑 공격, 차분공격 등)에 저항이 강함





LEA

라운드 함수



(그림 7) 암호화 과정 : 1 번째 라운드 함수



Feistel

라운드 함수의 마지막($n-1$ 번째) 부분만 다름

➔ 마지막($n-1$ 번째)은 **XOR연산**을 하지 **않음**

LEA

라운드 함수의 마지막 부분($n-1$ 번째)이 다르지 **않음**

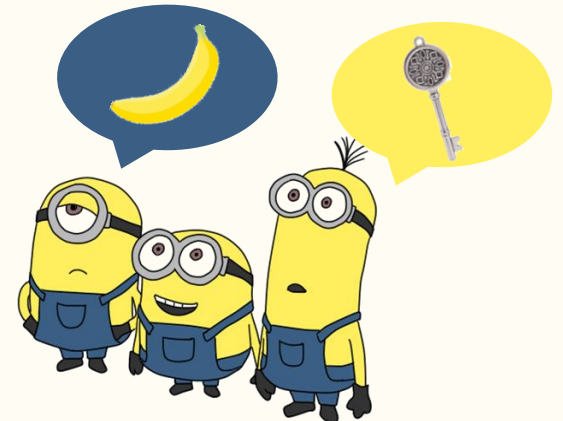
➔ 마지막($n-1$ 번째)도 똑같이 **XOR연산**을 **함**





HIGHT

저전력 & 경량화 블록 암호 알고리즘





HIGHT



첫 번째



HIGH security and light weig**HT**



64비트 블록 길이의 메시지 -> 128비트 키길로 암호화



2005년 KISA, 구 국가보안연구소 및 고려대 공동 개발

√ 2006년 12월 정보통신단체 표준 제정

√ 2010년 12월 ISO/IEC 국제 블록 암호 알고리즘 표준 제정



일반화된 **Feistel** 변형 구조





① Initialization (키 생성)



두 번째



암호화

평문

↓ ↓ ↓ ↓ ↓ ↓

초기변환

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

최종변환

↓ ↓ ↓ ↓ ↓ ↓

암호문

키스케줄

MK

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

WK_3 WK_2 WK_1 WK_0

SK_3 SK_2 SK_1 SK_0

⋮

SK_{127} SK_{126} SK_{125} SK_{124}

WK_7 WK_6 WK_5 WK_4

복호화

암호문

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

HIGHT의 라운드키는
화이트닝키와 LFSR을
사용하여 생성한
서브키들로 이루어짐

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

초기변환⁻¹

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

평문

키스케줄

MK

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

WK_3 | WK_2 | WK_1 | WK_0

SK_3 | SK_2 | SK_1 | SK_0

⋮

SK_{127} | SK_{126} | SK_{125} | SK_{124}

WK_7 | WK_6 | WK_5 | WK_4

화이트닝키(WK)

반복 구조의 블록암호에서

안전성을 높이기 위해

알고리즘의 초기변환 또는 최종변환에

적용되는 라운드키

$$WK_i = \begin{cases} MK_{i+12} & , x \leq i \leq 3 \\ MK_{i-4} & , 4 \leq i \leq 7 \end{cases}$$

→ 화이트닝키는 마스터키를 사용하여 생성



키스케줄

MK

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

WK_3 WK_2 WK_1 WK_0

SK_3 SK_2 SK_1 SK_0

⋮

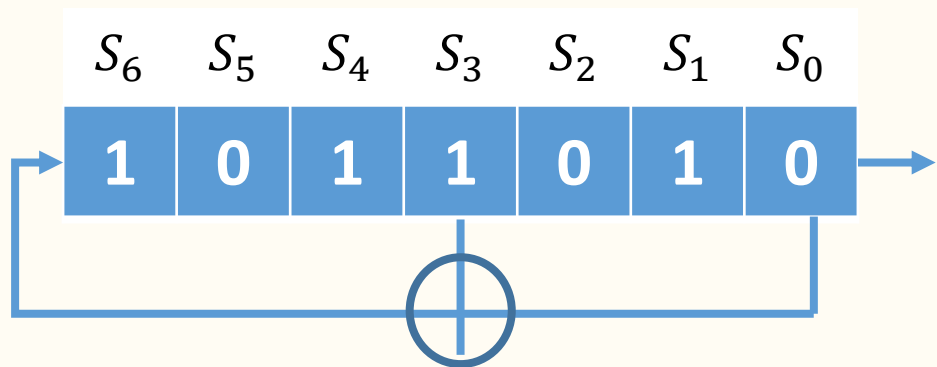
SK_{127} SK_{126} SK_{125} SK_{124}

WK_7 WK_6 WK_5 WK_4

서브키(SK)

라운드 함수에서 사용되는 키

→ 서브키는 LFSR을 사용한 알고리즘을 통하여 생성



Linear Feedback Shift Register

레지스터에 입력되는 값이 이전 상태 값들의 선형 함수로 계산되는 구조



② Encryption (암호화)



두 번째



암호화

평문

↓ ↓ ↓ ↓ ↓ ↓

초기변환

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

최종변환

↓ ↓ ↓ ↓ ↓ ↓

암호문

키스케줄

MK

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

WK_3 WK_2 WK_1 WK_0

SK_3 SK_2 SK_1 SK_0

⋮

SK_{127} SK_{126} SK_{125} SK_{124}

WK_7 WK_6 WK_5 WK_4

복호화

암호문

↓ ↓ ↓ ↓ ↓ ↓

최종변환⁻¹

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

초기변환⁻¹

↓ ↓ ↓ ↓ ↓ ↓

평문

암호화

평문

↓ ↓ ↓ ↓ ↓ ↓

초기변환

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

최종변환

↓ ↓ ↓ ↓ ↓ ↓

암호문

$$X_{0,i} = P_i, \quad i = 1, 3, 5, 7$$

$$X_{0,0} = P_0 \oplus WK_0$$

$$X_{0,2} = P_2 \oplus WK_1$$

$$X_{0,4} = P_4 \oplus WK_2$$

$$X_{0,6} = P_6 \oplus WK_3$$

암호화 초기변환

$$X_{i,j} = X_{i-1,j-1}, \quad j = 1, 3, 5, 7$$

$$X_{i,0} = X_{i-1,7} \oplus (F_0(X_{i-1,6}) \oplus SK_{4i-1})$$

$$X_{i,2} = X_{i-1,1} \oplus (F_0(X_{i-1,0}) \oplus SK_{4i-4})$$

$$X_{i,4} = X_{i-1,3} \oplus (F_0(X_{i-1,2}) \oplus SK_{4i-3})$$

$$X_{i,6} = X_{i-1,5} \oplus (F_0(X_{i-1,4}) \oplus SK_{4i-2})$$

i 번째 라운드 함수

$$X_{32,i} = X_{31,i}, \quad i = 0, 2, 4, 6$$

$$X_{32,1} = X_{31,1} \oplus (F_1(X_{31,0}) \oplus SK_{124})$$

$$X_{32,3} = X_{31,3} \oplus (F_0(X_{31,2}) \oplus SK_{125})$$

$$X_{32,5} = X_{31,5} \oplus (F_1(X_{31,4}) \oplus SK_{126})$$

$$X_{32,7} = X_{31,7} \oplus (F_0(X_{31,6}) \oplus SK_{127})$$

32번째 라운드 함수

$$C_i = X_{32,i}, \quad i = 1, 3, 5, 7$$

$$C_0 = X_{32,0} \oplus WK_4$$

$$C_2 = X_{32,2} \oplus WK_5$$

$$C_4 = X_{32,4} \oplus WK_6$$

$$C_6 = X_{32,6} \oplus WK_7$$

암호화 최종변환



③ Decryption (복호화)



두 번째



암호화

키스케줄

복호화

평문

↓ ↓ ↓ ↓ ↓ ↓

초기변환

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

최종변환

↓ ↓ ↓ ↓ ↓ ↓

암호문

MK

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

암호화 과정의 역순

덧셈 → 뺄셈

순환이동 → 반대

⋮

SK_{127}

SK_{126}

SK_{125}

SK_{124}

WK_7

WK_6

WK_5

WK_4

암호문

↓ ↓ ↓ ↓ ↓ ↓

최종변환⁻¹

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_1$

↓ ↓ ↓ ↓ ↓ ↓

⋮ ⋮ ⋮ ⋮ ⋮ ⋮

↓ ↓ ↓ ↓ ↓ ↓

$ROUND_{32}$

↓ ↓ ↓ ↓ ↓ ↓

초기변환⁻¹

↓ ↓ ↓ ↓ ↓ ↓

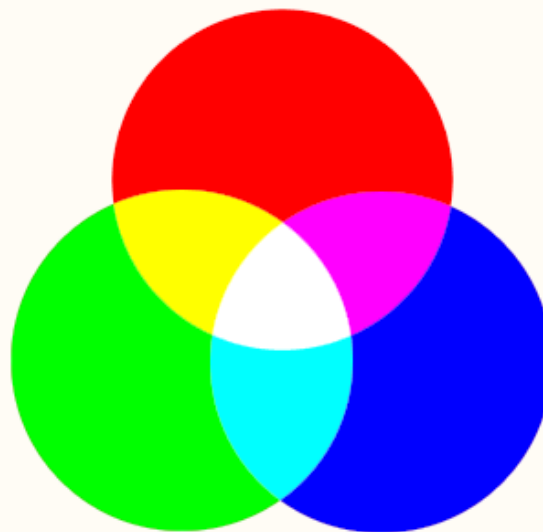
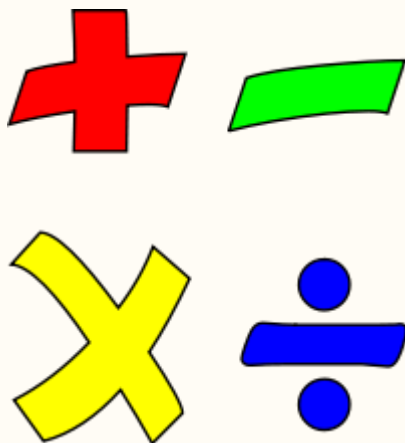
평문



특징①



- 제한적 자원을 갖는 환경에서 구현될 수 있도록
8비트 단위의 기본적인 산술 연산 (XOR, 덧셈, 순환이동)
→ SEED, AES등 보다 간단한 알고리즘 구조로 설계
(안전성, 효율성)





특징②



- 저전력, 경량화를 요구하는 컴퓨팅 환경에서 기밀성 제공
USN안의 센서 or RFID 태그 등 **유비쿼터스 컴퓨팅 장치에 적합**
→ 휴대형 기기 및 모바일 환경에 적합
→ 모바일 기기의 제한된 성능 및 용량, 배터리 등의 문제를 해결





비교

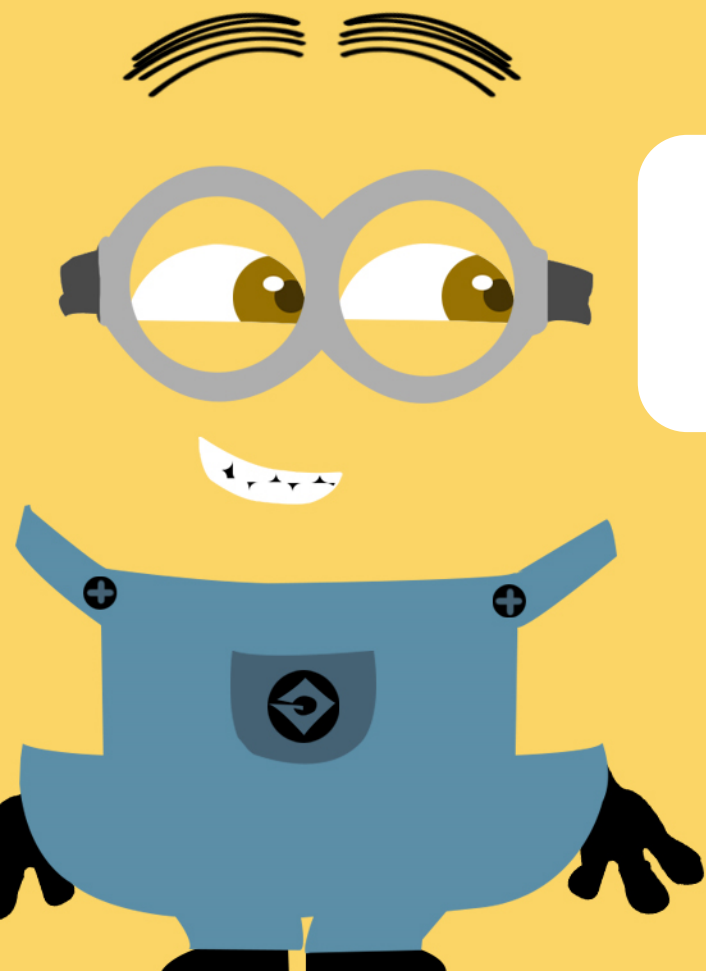
종류

n k	종류	bit-serial		round-based		tech
		공간	처리량	공간	처리량	
64 128	HIGHT	-	-	3,048	188	250n
	CHAM	665	5	826	80	IBM130
		859	5	1,110	80	UMC180
		727	5	985	80	UMC90
128 128	LEA	2,302	4.2	3,826	76.2	UMC130
	CHAM	1,057	5	1,499	160	IBM130
		1,296	5	1,899	160	UMC180
		1,084	5	1,691	160	UMC90



n = 블록 사이즈
k = 키 사이즈

(표 4) HIGHT, LEA, CHAM 성능 비교



Thank U