블록암호
(DES, PRESENT)

https://youtu.be/-4K-OZyVAxc

IT융합공학부 송경주

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

암호화란?

의미를 알 수 없는 형식(암호문)으로 정보를 변환하는 것. 암호문의 형태로 정보를 기억 장치에 저장하거나 통신 회선을 통해 전송함으로써 정보를 보호할 수 있다.

[방식]

1. 대칭키 암호
2. 블록암호 ex) DES
3. 스트림 암호
4. 비대칭키 암호

…

암호의 기본 개념 2가지

1. Confusion (혼돈)
   메시지 원문의 내용을 짐작하기 어렵게 만듦.

2. Diffusion (확산)
   암호화 알고리즘의 패턴 추론을 어렵게 만듦.
   Ex)비트치환

→2가지 성질을 모두 만족하는 암호가 좋은 암호라고 할 수 있다.

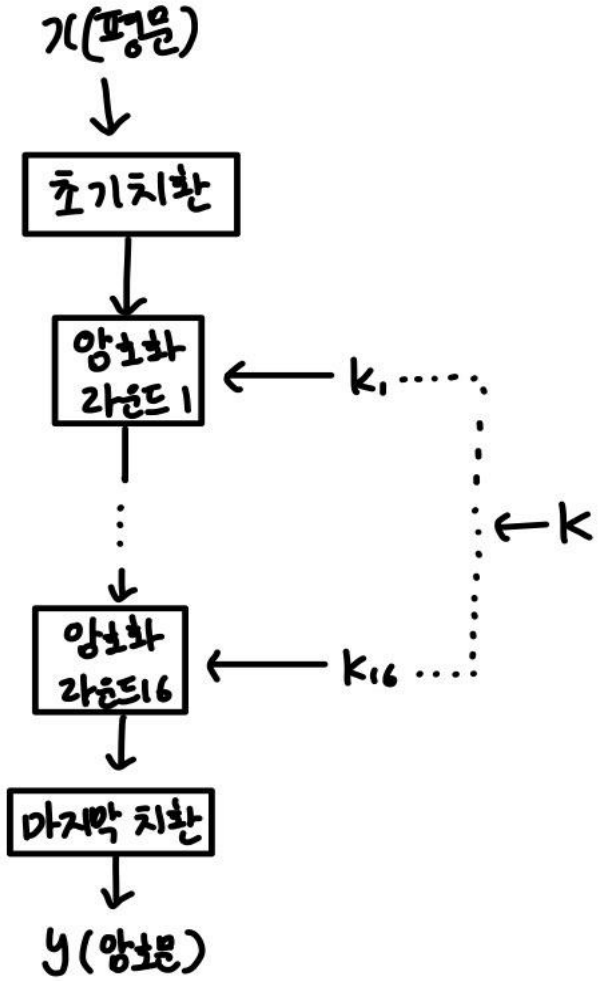
DES

64비트의 평문을 46비트의 암호문으로 만드는 블록 암호 시스템

<동작원리>

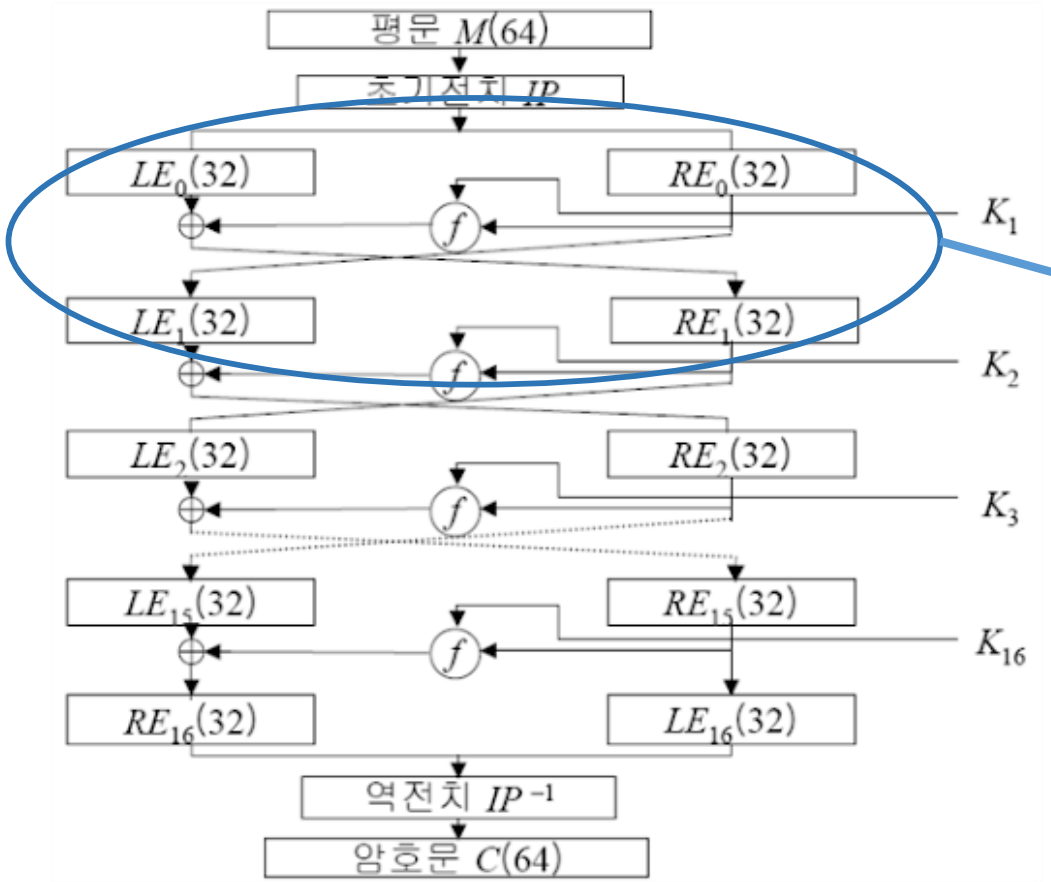
1. 대칭암호, 반복적인 알고리즘 (평문 : 64 bit, 암호문 : 64bit, 키: 56bit)
2. 16라운드 수행 (모든 라운드는 동일한 연산을 수행)
3. 각 라운드에서 서로 다른 서브키 사용 (서브키는 주키에서 유도됨)

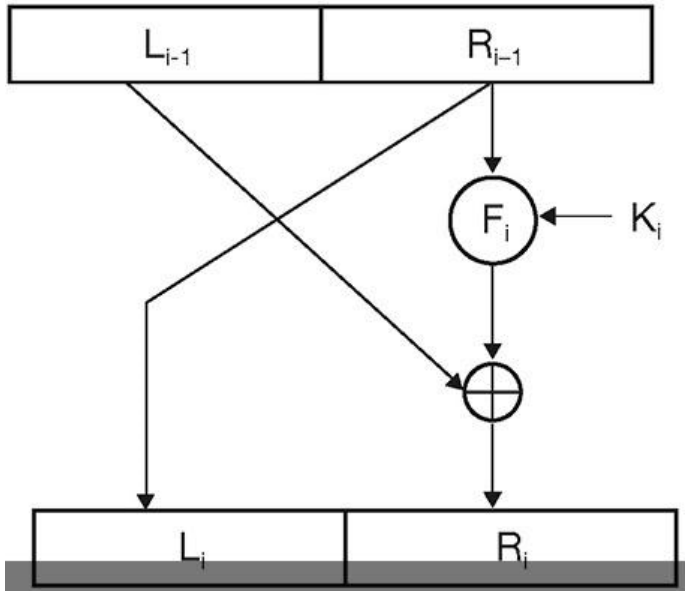
DES 반복구조



- 64 비트 평문을 초기 치환하고 동일한 연산을 수행하는 16라운드를 수행.
- 마지막 치환 후 암호문 도출
- 키 K는 초기 키로부터 라운드 키 16개를 얻음

DES의 Feistel 구조

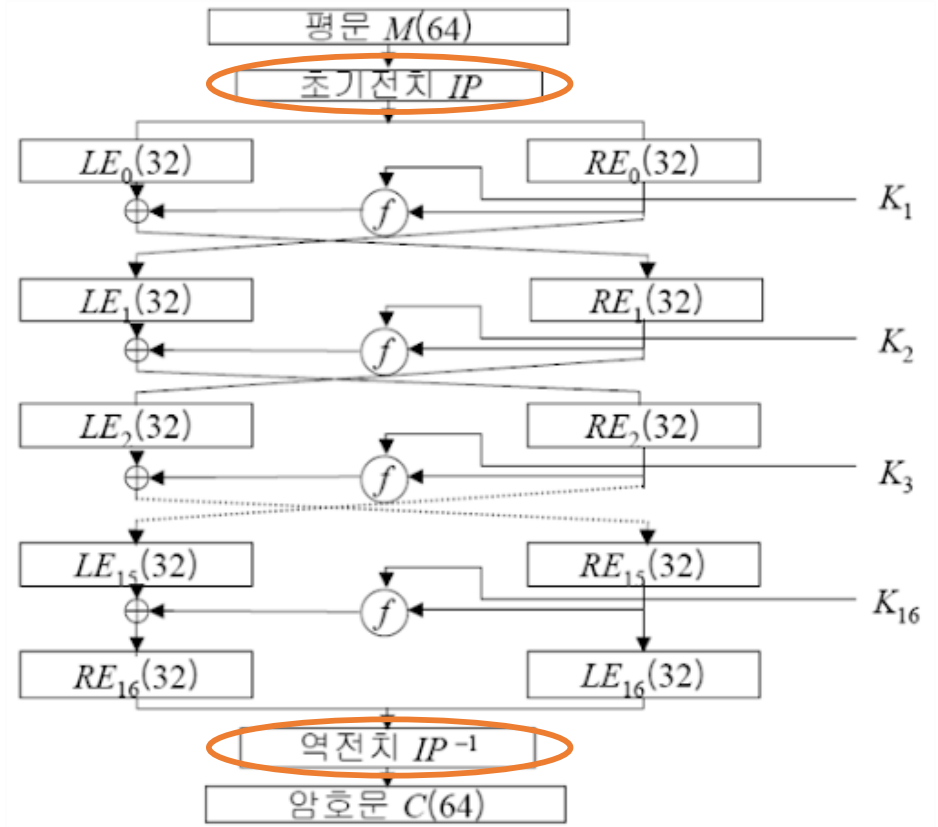




$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, k)$

DES 내부구조

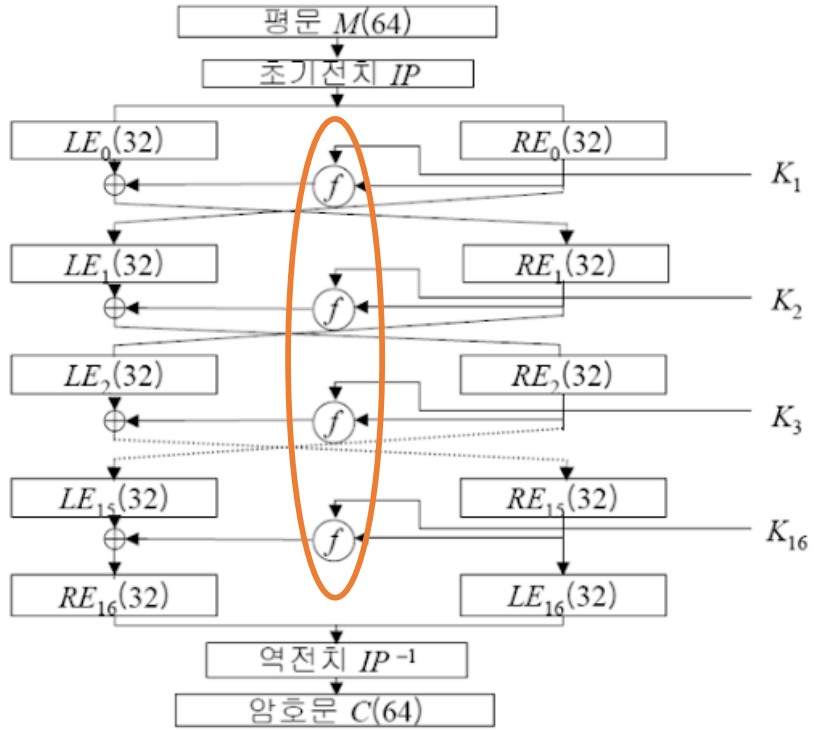


<초기 치환, 마지막 치환>

안전성 증진에는 전혀 도움 X

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

DES 내부구조

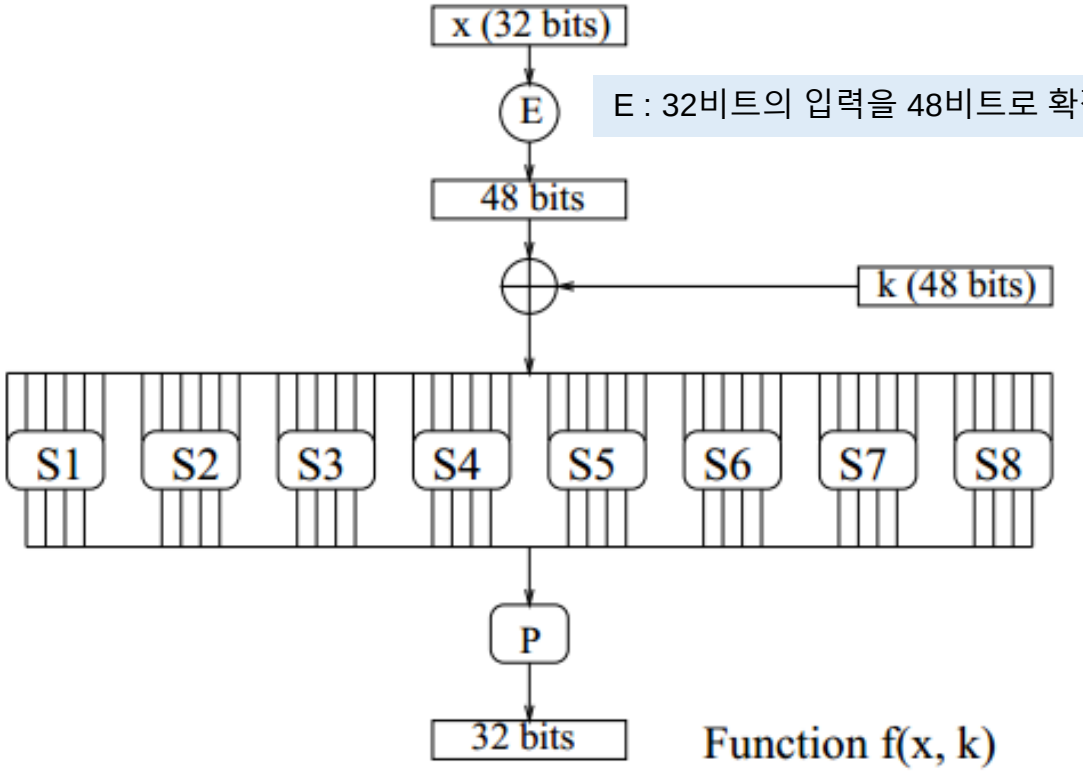


<f-함수>

DES 안전성에 중요한 역할

(S-Box가 여기에 포함됨)

DES내부구조_f 함수



E : 32비트의 입력을 48비트로 확장 시킴.

S-Box : 6비트 입력을 4비트 결과로 대응 시킴.

S-Box

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S-Box

암호학적 강도 측면에서 DES의 핵심.

DES알고리즘에서 유일한 비선형 요소→Confusion제공

설계 기준에 따라 설계된 DES에서 가장 중요한 요소.

<설계 기준>

1. 각 S-Box는 6비트의 입력과 4비트의 결과를 갖는다.
2. 어떠한 단일 결과 비트도 입력 비트들의 선형결합에 너무 유사해서는 안된다.
3. 입력의 가장 낮은 비트와 가장 높은 비트가 고정되고 4개의 중간비트가 변하면 4비트의 가능한 결과 값은 반드시 정확하게 한번씩만 나타나야 한다.
4. S-Box에 대한 두 입력이 정확하게 한 비트만 다르면 결과는 반드시 최소 두개의 비트가 달라야 한다.
5. S-Box에 대한 두 입력의 중간의 두 개의 비트가 다르면 결과는 반드시 최소 두개의 비트가 달라야 한다.
6. S-box에 대한 두 입력의 첫번째 두 비트가 다르고 마지막 두 비트가 동일하면 두 개의 결과는 반드시 달라야 한다.
7. 두 입력이 0이 아닌 6비트가 서로 다르면 그러한 차이를 보여주는 입력의 32개 쌍 중 8 이하만이 동일한 결과의 차이를 생성한다.
8. 8개의 S-Box의 32비트 결과에서 충돌(결과차이가 없음)은 3개의 인접한 s-box에서만 발생한다.

S-Box

<S-Box 보는 방법>

각 6비트 입력의 최상위 비트(MSB)와 최하위 비트(LSB)는 표의 행을 의미하고
내부의 4비트는 열을 의미한다.

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

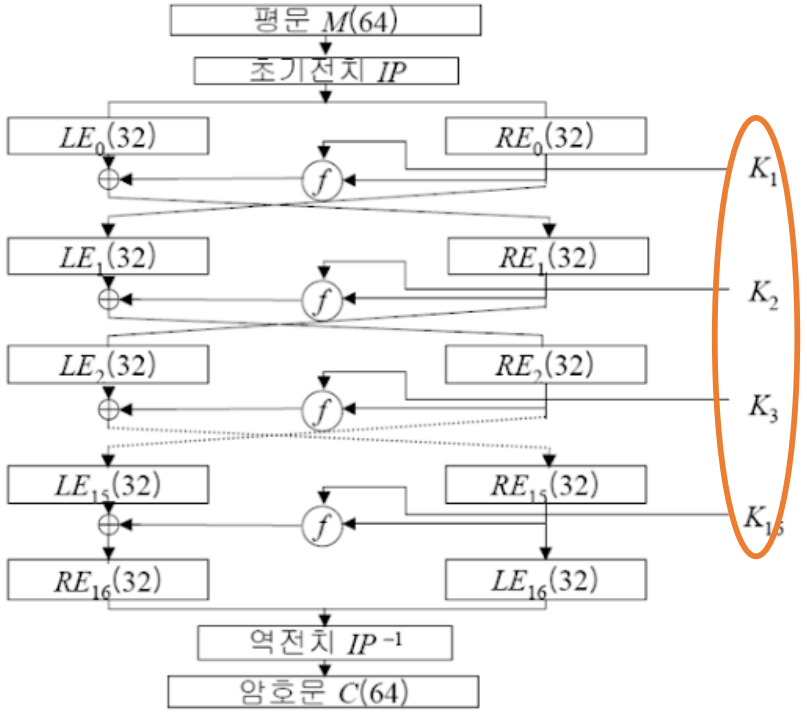
ex) S-Box 입력 $b = (100101)_2$

→ 행 : $11_2 = 3$

열 : $0010_2 = 2$

→ S-Box의 3행 2열과 매칭된다

DES 내부구조



<키 스케줄>

기존 56비트 키로부터 16라운드 키 유도.

각 키는 48비트로 구성됨.

각 키의 8번째 비트는 이전 7개의 비트에 대한 홀수 패러디 비트로 사용됨.

경량암호란?

[경량암호]

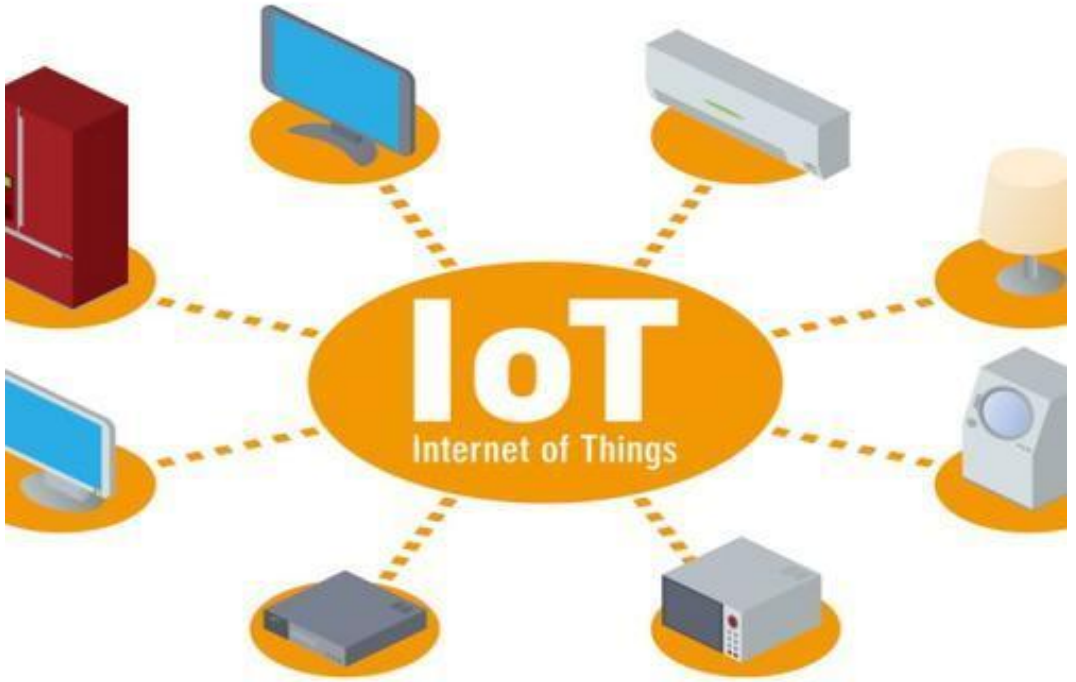
사물인터넷의 소형화 특성에 따라 제한된 하드웨어 면적과 전력 소비량, 메모리 크기에 맞춰 개발되고 있는 암호기술.

Feistel(ARX) 구조

종류: LEA, HIGHT, SIMON/SPECK, DES 등등

SPN 구조

종류: PRESENT, CLEFIA, SEA, AES 등등

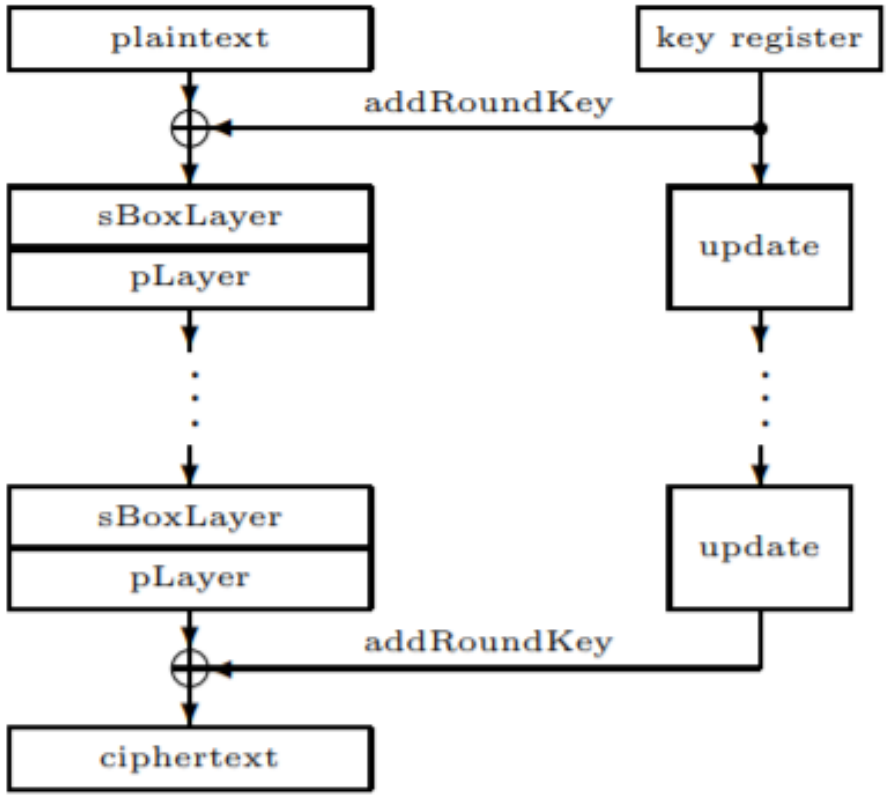


PRESENT

전력과 비용이 극히 제한되는 RFID태그나 소형 기기를 위해 설계된 암호기술.

<특징>
1) SP(대치-치환) 네트워크
2) 31개의 라운드로 구성
3) 블록길이: 64비트, 키: 80비트와 128비트의 두개의 키 쌍이 지원됨.

PRESENT 내부구조



AddRoundKey : 각 라운드 시작에 라운드 키 k는 현재 STATE에 XOR됨.

S-Boxlayer : PRESENT는 단일 4비트에서 4비트로의 S-box를 이용함.

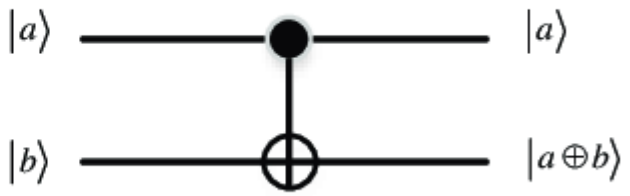
(8비트의 S-Box보다 소형 구현이 가능하므로 하드웨어 효율성 향상)

PLayer : 비트 치환.

키 스케줄 : 사용자가 제공하는 키(80비트)를 이용하여 64비트의 라운드 키 추출.

CNOT-gate

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



<CNOT-gate>

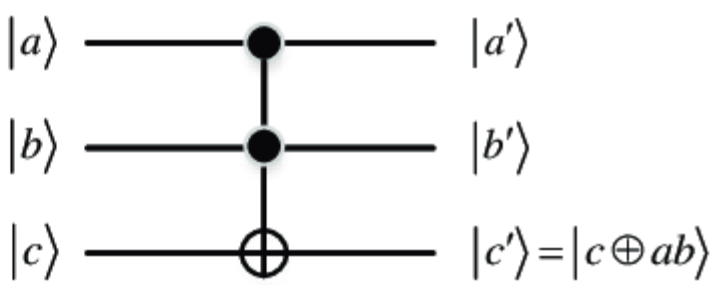
a , b 2개의 입력을 받으며 a의 값이 b의 값에 영향을 준다.

a가 1일 경우에만 b를 토글 시킨다.

```
CNOT | (x[a], x[b])
```

Toffoli-gate

Inputs			Ouputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



<Toffoli-gate>

a , b, c 3개의 입력을 받으며 a, b의 값이 c의 값에 영향을 준다.

a, b 모두 1일 경우에만 c를 토글 시킨다.

```
Toffoli | (x[a], x[b], x[c])
```

PRESENT S-Box

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

```
F[0] = X[0];
F[1] = X[2];
F[2] = X[3];
F[3] = X[1];

F[2] = RNOT1(F[2]);
F[3] = CCNOT2(F[2], F[1], F[3]);
F[0] = CCNOT2(F[3], F[1], F[0]);
F[2] = CNOT1(F[2], F[0]);
F[1] = CCCNOT2(F[0], F[2], F[3], F[1]);
F[3] = RNOT1(F[3]);
F[1] = CCNOT2(F[3], F[0], F[1]);
F[0] = CNOT1(F[0], F[3]);
F[3] = CNOT1(F[3], F[2]);
F[2] = CCCNOT2(F[0], F[1], F[3], F[2]);
F[0] = CCNOT2(F[2], F[1], F[0]);

X[0] = F[2];
X[1] = F[0];
X[2] = F[1];
X[3] = F[3];
```

S-Box 양자 회로로 구현

```
def p_sbox(eng, x):
    X | (x[3])
    Toffoli | (x[3], x[2], x[1])
    Toffoli | (x[1], x[2], x[0])
    CNOT | (x[0], x[3])
    with Control(eng, x[0]):
        Toffoli | (x[3], x[1],x[2])
    X | (x[1])
    Toffoli | (x[1], x[0], x[2])
    CNOT | (x[3], x[0])
    CNOT | (x[3], x[1])
    with Control(eng, x[0]):
        Toffoli | (x[2], x[1], x[3])
    Toffoli | (x[3], x[2], x[0])
```

PRESENT S-Box

```
def p_sbox(eng, x):
    X | (x[3])
    Toffoli | (x[3], x[2], x[1])
    Toffoli | (x[1], x[2], x[0])
    CNOT | (x[0], x[3])
    with Control(eng, x[0]):
        Toffoli | (x[3], x[1],x[2])
    X | (x[1])
    Toffoli | (x[1], x[0], x[2])
    CNOT | (x[3], x[0])
    CNOT | (x[3], x[1])
    with Control(eng, x[0]):
        Toffoli | (x[2], x[1], x[3])
    Toffoli | (x[3], x[2], x[0])

def test(eng):
    x = eng.allocate_qureg(4)
    X | x[0]
    X | x[1]
    X | x[2]
    X | x[3]

    p_sbox(eng, x)
    Swap | (x[0], x[3])
    Swap | (x[3], x[1])
    All(Measure) | x
    return int(x[0]), int(x[1]), int(x[2]), int(x[3])
    Resource = ClassicalSimulator()
    eng = MainEngine(Resource)
    print(test(eng))
    eng.flush()
```

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

입력 값을 넣고 PRESENT S-box 값에 맞는 값이 나오는지 확인함으로써 코드의 유효성을 확인할 수 있었음.

Q & A