경량암호
PRESENT

https://youtu.be/Av7KyR0MM0k

IT융합공학부 송경주

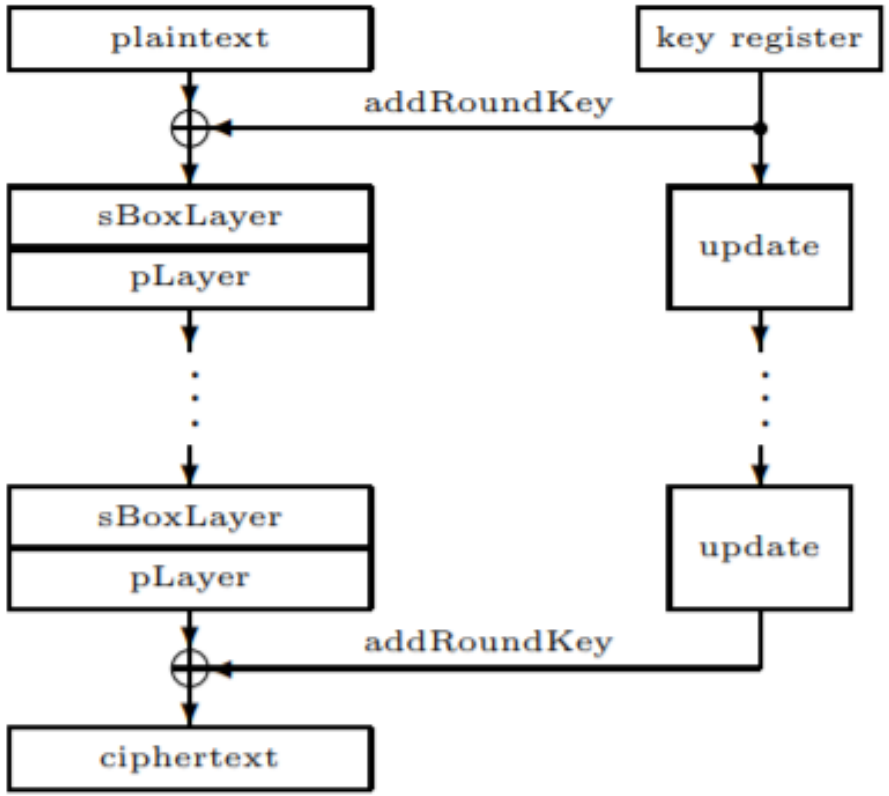
HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

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.

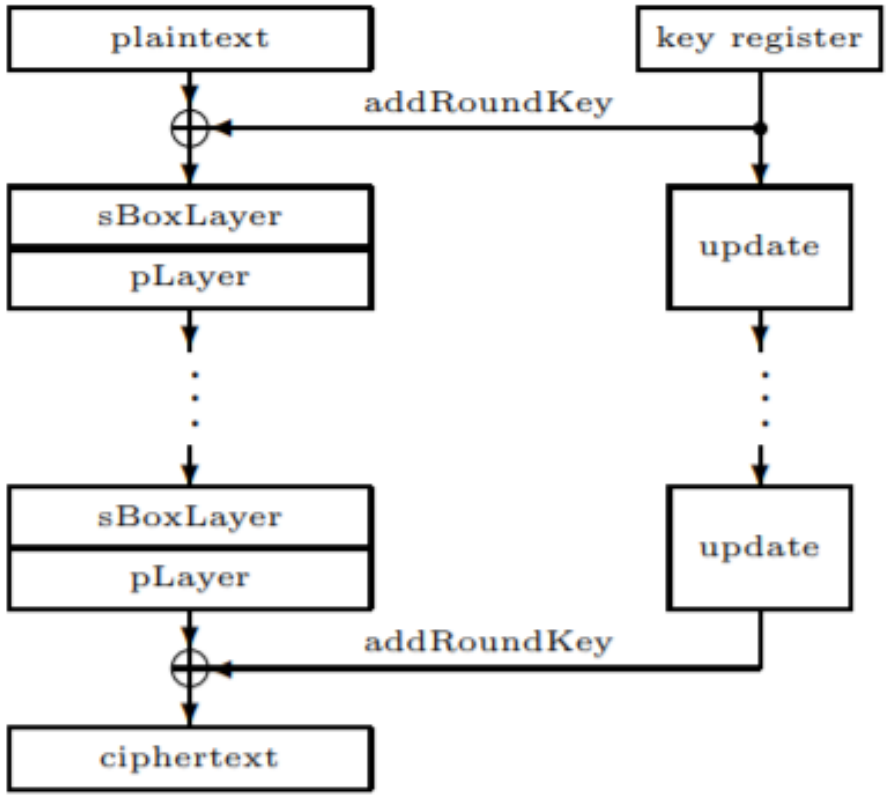
PRESENT_PLayer

PRESENT_Player : 단순 비트 치환

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

```
def Permutation(eng, b):

    Swap|(b[1], b[4])
    Swap|(b[16], b[4])

    Swap|(b[2], b[8])
    Swap|(b[32], b[8])

    Swap|(b[3], b[12])
    Swap|(b[48], b[12])

    Swap|(b[5], b[20])
    Swap|(b[17], b[20])

    Swap|(b[6], b[24])
    Swap|(b[33], b[24])

    Swap|(b[7], b[28])
    Swap|(b[49], b[28])
```

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.

PRESENT_키스케줄

PRESENT 키 스케줄 : 사용자가제공하는 키를 80bit 레지스터 k에 저장하고 64bit 라운드키를 추출한다.

레지스터k → 80개의 비트로 구성됨.

레지스터k에서 64비트의 라운드키 추출.

라운드 키는 현재의 레지스터 k의 제일 왼쪽의 64개의 비트로 구성됨.

PRESENT_키스케줄

-첫번째 서브키(라운드키) k_1은 사용자가 제공한 키 중 64비트를 그대로 복사, 이후 서브키 $k_2,k_3,\cdots,k_{31},k_{32}$ 에 대해 키 레지스터 $k=k_{79},k_{78},\cdots,k_0$ 는 아래와 같이 갱신된다.

1. **$[\ k_{79},k_{78},\cdots,k_1,k_0\ ] = [\ k_{18},k_{17},\cdots,k_{20},k_{19}\ ]$: 키 레지스터는 61 비트 위치만큼 왼쪽으로 순환됨.**

2. **$[\ k_{79},k_{78},k_{77},k_{76}\ ] = S[\ k_{79},k_{78},k_{77},k_{76}\ ]$: 제일 왼쪽의 4개의 비트는 PRESENT S-Box에 입력됨.**

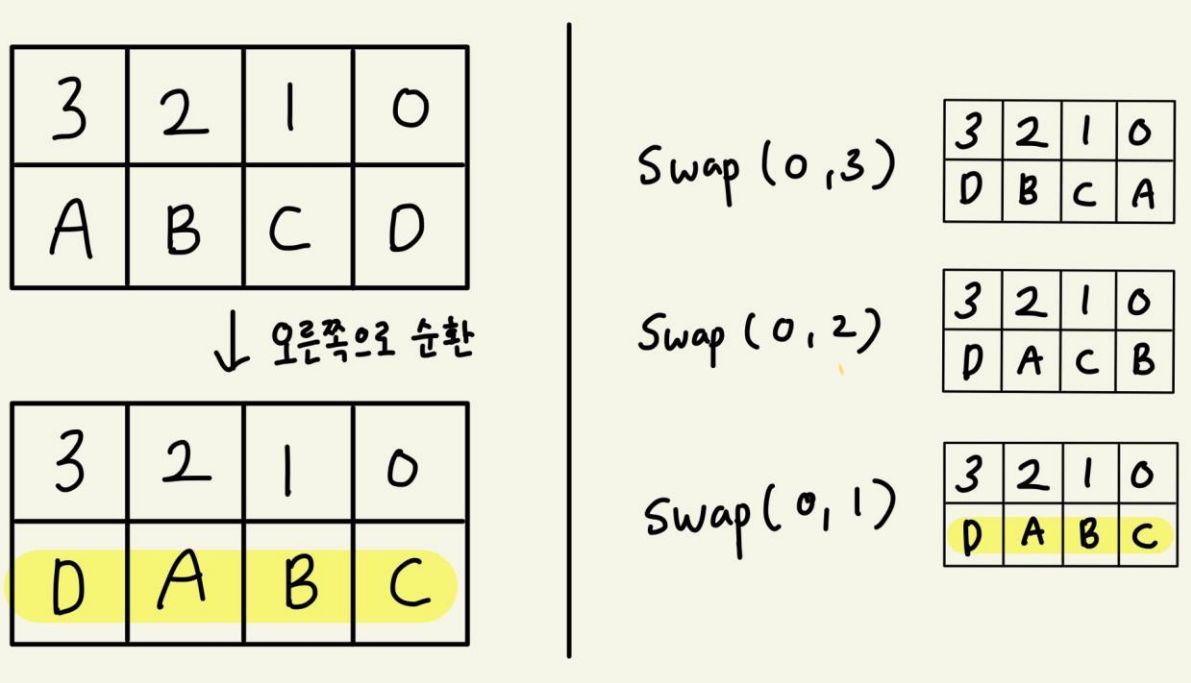
3. **$[\ k_{19},k_{18},k_{17},k_{16},k_{15}\ ] = [\ k_{19},k_{18},k_{17},k_{16},k_{15}\ ] \oplus$ round_counter (i)**

: round_counter 값 i는 k의 $k_{19}k_{18}k_{17}k_{16}k_{15}$비트와 XOR 됨, 이때 round_counter의 최하위 비트가 오른쪽에 위치

k_2 유도에서는 0001
k_3유도에서는 0010

PRESENT 키스케줄 구현

1. $[k_{79},k_{78},\cdots,k_1,k_0] = [k_{18},k_{17},\cdots,k_{20},k_{19}]$: 키 레지스터는 61 비트 위치만큼 왼쪽으로 순환됨.




```
def keyscheduling (eng, k):
    for i in range (0, 3):
        Swap (k[0], k[3-i])
```

Swap을 이용하여 비트를 순환시킴

(61bit 왼쪽으로 순환 = 19bit오른쪽으로 순환)

80bit의 레지스터 순환 방식을 찾기 어려워 4bit 레지스터로 우선 해본 다음 확장시키는 방식을 이용함.

19bit 의 순환방식을 찾기 어려워 1bit의 순환의 규칙을 찾아 19번 반복하는 방식을 이용하였음.

여러 방식을 해본 결과 차례대로 한 bit씩 결과에 맞춰 Swap 하는 방식에서 규칙을 찾을 수 있었음.

PRESENT 키스케줄 구현

1. $[k_{79},k_{78},\cdots,k_1,k_0] = [k_{18},k_{17},\cdots,k_{20},k_{19}]$: 키 레지스터는 61 비트 위치만큼 왼쪽으로 순환됨.

```
def keySchedule(eng, k, ctr):
    ctr = ctr + 1
    for j in range(0,19):
        for i in range(0,79):
            Swap(k[0],k[(79-i)])

    p_sbox(eng, k[76:80])

    CTR_XOR(eng, k[15:20], ctr)
```

4bit 레지스터를 오른쪽으로 1bit 순환 방법 확장

→80bit 레지스터를 오른쪽으로 19bit 순환

PRESENT 키스케줄 구현

2. $[k_{79},k_{78},k_{77},k_{76}] = S[k_{79},k_{78},k_{77},k_{76}]$: 제일 왼쪽의 4개의 비트는 PRESENT S-Box에 입력됨.

```
def keySchedule(eng, k, ctr):
    ctr = ctr + 1
    for j in range(0,19):
        for i in range(0,79):
            Swap|(k[0],k[(79-i)])

    p_sbox(eng, k[76:80])

    CTR_XOR(eng, k[15:20], ctr)
```

k[76:80] = $k_{76},k_{77},k_{78},k_{79}$ 를 S-Box에 입력값으로 넣는다.

PRESENT 키스케줄 구현

3. $[k_{19},k_{18},k_{17},k_{16},k_{15}] = [k_{19},k_{18},k_{17},k_{16},k_{15}]] \oplus$ round_counter (i)

```
def keySchedule(eng, k, ctr):
    ctr = ctr + 1
    for j in range(0,19):
        for i in range(0,79):
            Swap|(k[0],k[(79-i)])

    p_sbox(eng, k[76:80])

    CTR_XOR(eng, k[15:20], ctr)
```

```
def CTR_XOR(eng, k, ctr):

    if (ctr >= 16):
        X | (k[4])
        ctr = ctr - 16
    if (ctr >= 8):
        X | (k[3])
        ctr = ctr - 8
    if (ctr >= 4):
        X | (k[2])
        ctr = ctr - 4
    if (ctr >= 2):
        X | (k[1])
        ctr = ctr - 2
    if (ctr >= 1):
        X | (k[0])
        ctr = ctr - 1
```

round_counter 와 xor하는 연산.

k와 round_counter 을 단순 XOR하므로

round_counter 의 비트가 1인 자리를 Not gate 한다.

PRESENT 동작 결과

```
Gate class counts:
    AllocateQubitGate : 144
    CCCXGate : 1054
    CCXGate : 2108
    CXGate : 3629
    DeallocateQubitGate : 144
    MeasureGate : 64
    SwapGate : 48825
    XGate : 1134

Gate counts:
    Allocate : 144
    CCCX : 1054
    CCX : 2108
    CX : 3629
    Deallocate : 144
    Measure : 64
    Swap : 48825
    X : 1134

 Depth : 1824.
```

코드실행을 통해 모든 라운드를 동작하며 사용한 Qubit 과 각 Gate를 확인 할 수 있다.

Q & A