LED 암호 구현

송민호

유튜브 주소: https://youtu.be/PYDjJXXBX3U

LED

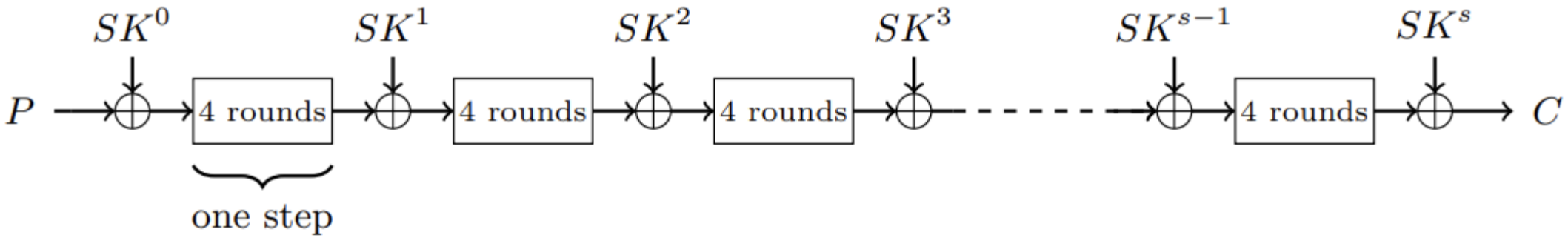
- 2011년 CHES에서 발표한 경량 암호

- 블록 길이 64-bit, 키 길이 64-bit, 80-bit, 96-bit, 128-bit
  - 키 길이가 64-bit인 경우, 8 라운드 연산
  - 그 외에는 12라운드 연산

- 4-bit를 이용한 4x4 배열을 4-bit 단위로 연산

LED

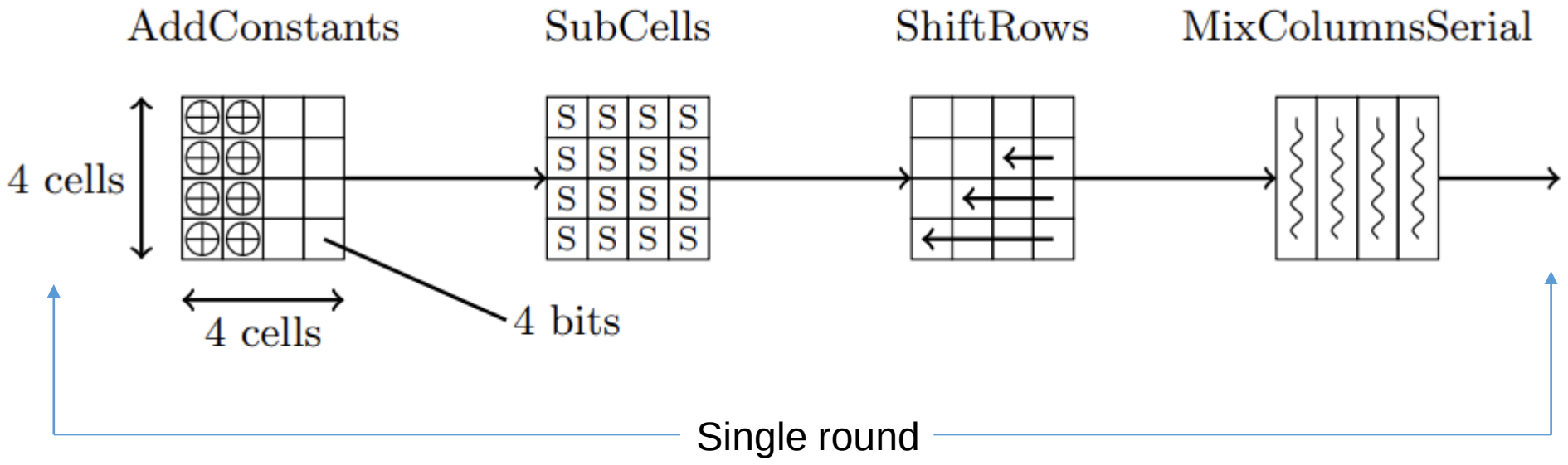
- 연산 순서



for $i = 0$ **to** $s - 1$ **do** {
    addRoundKey(STATE, SK^i)
    step(STATE)
}
addRoundKey(STATE, SK^s)

LED

- One step
  - Single round 4번 반복



LED

- LED 64/64
- Key 64-qubit
- Block 64-qubit
  - 첫 번째 행 b0 = 16-qubit

```
k = eng.allocate_qureg(64)
b0 = eng.allocate_qureg(16)
b1 = eng.allocate_qureg(16)
b2 = eng.allocate_qureg(16)
b3 = eng.allocate_qureg(16)

addRoundKey(eng, b0, b1, b2, b3, k)

cnt = 0

for i in range(0, rounds):
    for j in range(0, 4):
        AddConstants(eng, b0, b1, b2, b3, cnt)
        SubCells_layer(eng, b0, b1, b2, b3)
        ShiftRows(eng, b0, b1, b2, b3)
        MixColumns(eng, b0, b1, b2, b3)
        cnt += 4

    addRoundKey(eng, b0, b1, b2, b3, k)
```

addRoundKey

- Key와 평문을 XOR 연산

- 키 스케줄이 존재하지 않음
  - 매 라운드마다 주어진 키 그대로 사용

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix}$$

for 64-bit keys

```
def addRoundKey(eng, b0, b1, b2, b3, k):
    for i in range(16):
        CNOT | (k[i+48], b0[i])
        CNOT | (k[i+32], b1[i])
        CNOT | (k[i+16], b2[i])
        CNOT | (k[i], b3[i])
```

AddConstants

- Round Constants와 평문을 XOR 연산

```
def AddConstants(eng, b0, b1, b2, b3, cnt):
    rc = [0x4000,0x5100,0x2000,0x3100,0x4000,0x5300,
          0x4300,0x5700,0x2300,0x3700,0x4700,0x5600,
          0x4600,0x5700,0x2600,0x3700,0x4500,0x5700,
          0x4700,0x5100,0x2700,0x3100,0x4600,0x5300,
          0x4300,0x5500,0x2300,0x3500,0x4700,0x5200,
          0x4200,0x5600,0x2200,0x3600,0x4500,0x5400,
          0x4400,0x5100,0x2400,0x3100,0x4000,0x5200,
          0x4200,0x5700,0x2200,0x3700,0x4500,0x5600,

    Round_constant_XOR(eng, b0, rc[cnt], 16)
    Round_constant_XOR(eng, b1, rc[cnt+1], 16)
    Round_constant_XOR(eng, b2, rc[cnt+2], 16)
    Round_constant_XOR(eng, b3, rc[cnt+3], 16)
```

```
def Round_constant_XOR(eng, k, rc, bit):
    for i in range(bit):
        if (rc >> i & 1):
            X | k[i]
```

1000000000000000
-> 1000000000000000

SubCells

- Sbox 연산
  - Sbox LIGHTER-R을 통해 최적화
    - ./non-lin-search -qvw -o “C56B90AD3EF84712“ -f "libraries/NCT_gc.conf" -s “LED"
  - CCNOT2 = Toffoli
  - RNOT = X

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

```
// from : 00FF 0F0F 3333 5555

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

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

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

//to : 9B70 E16C 32E5 59A6
// Cost : 11
// Logic Library : NCT_gc
```

SubCells

- Endian 정렬

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

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

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

```
CNOT | (b[2], b[1])

Toffoli_gate(eng, b[1], b[2], b[3])
Toffoli_gate(eng, b[3], b[1], b[2])
Toffoli_gate(eng, b[0], b[2], b[1])
```

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

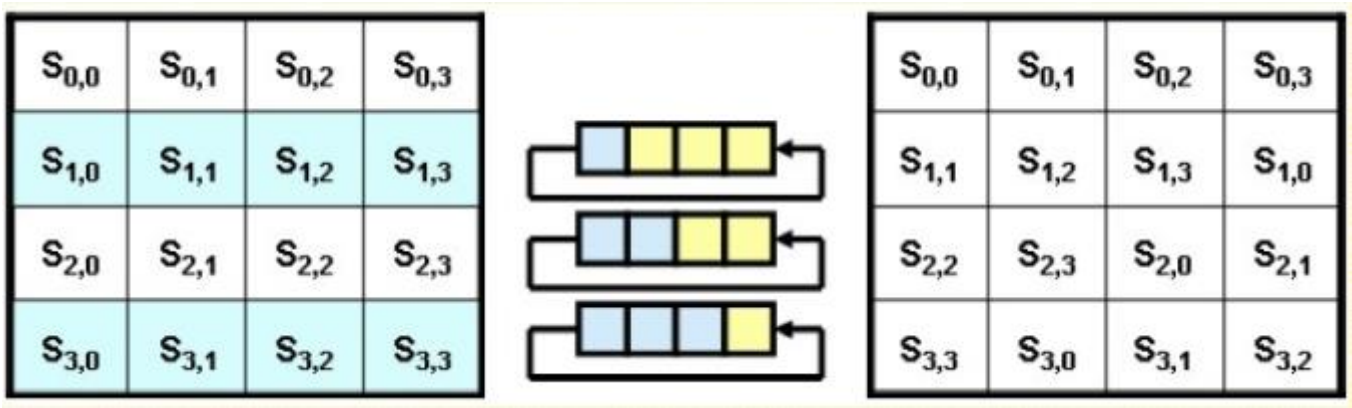
X[2] = F[0]
X[1] = F[3]
X[0] = F[1]
X[3] = F[2]

F[0] = X[2] -> F[0]
F[1] = X[0] -> F[2]
F[2] = X[3] -> F[3]
F[3] = X[1] -> F[1]

```
out = []
out.append(b[0])
out.append(b[2])
out.append(b[3])
out.append(b[1])
```

ShiftRows

- 각 행의 Cell을 Shift




```
for i in range(4): #5
    new_b1.append(b1[12+i])
for i in range(4): #6
    new_b1.append(b1[i])
for i in range(4):
    new_b1.append(b1[4+i])
for i in range(4): #4
    new_b1.append(b1[8+i])
```

MixColumnsSerial

- 행렬곱

$$\begin{pmatrix} 4\ 1\ 2\ 2 \\ 8\ 6\ 5\ 6 \\ B\ E\ A\ 9 \\ 2\ 2\ F\ B \end{pmatrix} = M.$$

4 1 2 2 8 6 5 6 B E A 9 2 2 F B	X	b0[0]	b0[1]	b0[2]	b0[3]
		b1[0]	b1[1]	b1[2]	b1[3]
		b2[0]	b2[1]	b2[2]	b2[3]
		b3[0]	b3[1]	b3[2]	b3[3]

(4 x b0[0]) ⊕ (1 x b1[0]) ⊕ (2 x b2[0]) ⊕ (2 x b3[0])

- Reduction 사용
  - 4-bit로 맞춰주기 위함
  - X^4 + X + 1

1	0	1	0	0	0
X^5	X^4	X^3	X^2	X	1
1	0	1	0	0	0
1	0	1	1	1	0
		1	**1**	**1**	**0**

MixColumnsSerial

$$\begin{matrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{matrix} \times \begin{matrix} b0[0] & b0[1] & b0[2] & b0[3] \\ b1[0] & b1[1] & b1[2] & b1[3] \\ b2[0] & b2[1] & b2[2] & b2[3] \\ b3[0] & b3[1] & b3[2] & b3[3] \end{matrix}$$

(4 x b0[0]) = 0100 x (b3 b2 b1 b0)

b3	b2	b1	b0	0	0
X^5	X^4	X^3	X^2	X	1
b3	b2	b1	b0	b2	b2
b3	b2	b1	b0 ⊕ b3	b2 ⊕ b3	b2
		b1	**b0 ⊕ b3**	**b2 ⊕ b3**	**b2**

```
#4
CNOT | (b0[0], out[2])
CNOT | (b0[1], out[3])

CNOT | (b0[2], out[0])
CNOT | (b0[2], out[1])
CNOT | (b0[3], out[1])
CNOT | (b0[3], out[2])
```

Q & A