KLEIN

송민호

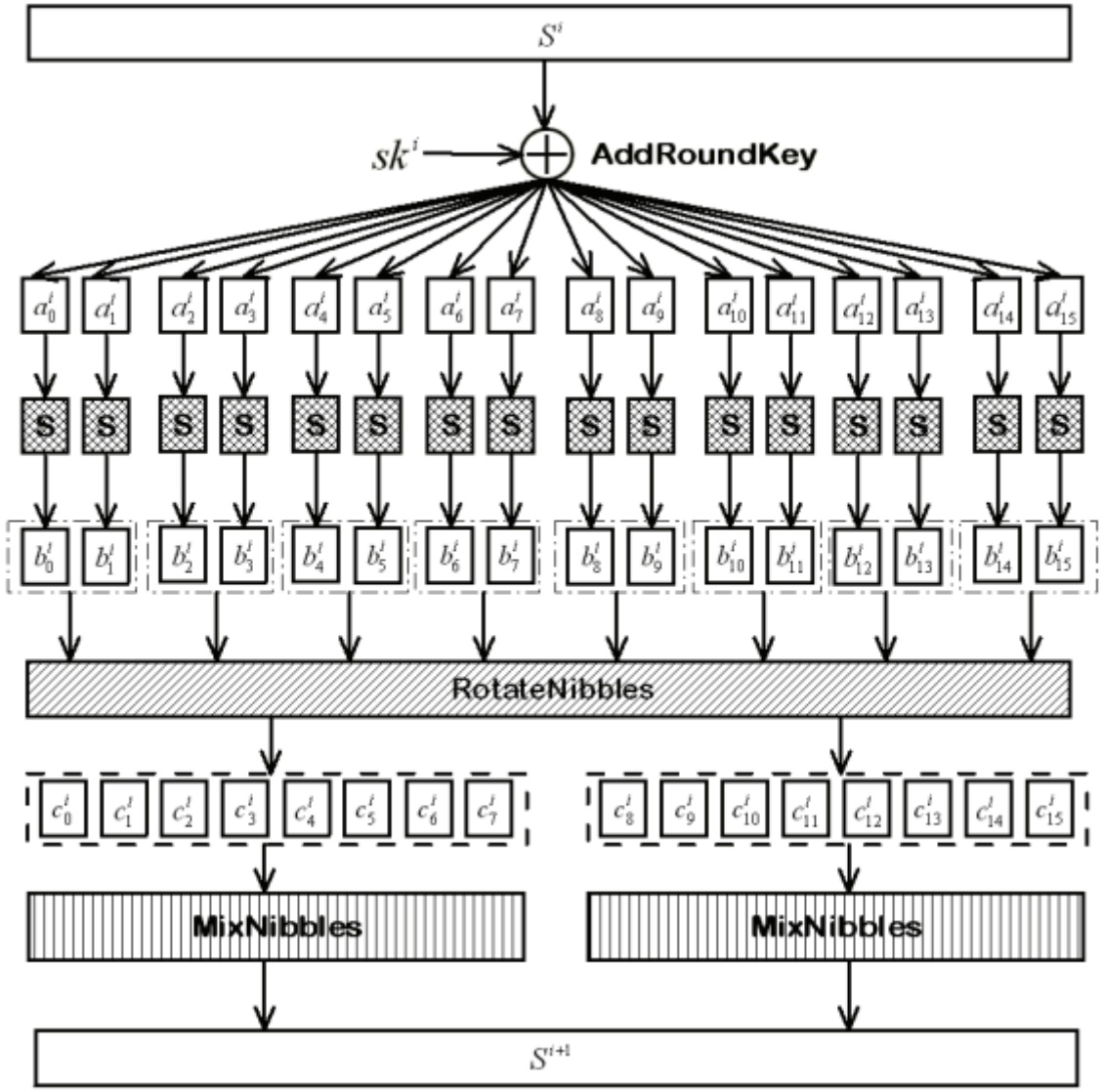
유튜브 주소: https://youtu.be/Plv5q9nGfPI

KLEIN

- SPN 구조
- 64비트 블록 사이즈
- 64, 80, 96비트의 키 사이즈
  - 12, 16, 20라운드 연산

$sk^1 \leftarrow \text{KEY};$
$\text{STATE} \leftarrow \text{PLAINTEXT};$
for $i = 1 \ to \ N_R$ do
  $AddRoundKey(\text{STATE}, sk^i);$
  $SubNibbles(\text{STATE});$
  $RotateNibbles(\text{STATE});$
  $MixNibbles(\text{STATE});$
  $sk^{i+1} = KeySchedule(sk^i, i);$
end for
$\text{CIPHERTEXT} \leftarrow AddRoundKey(\text{STATE}, sk^{N_R+1});$

KLEIN


S^i
sk^i
AddRoundKey
S
RotateNibbles
MixNibbles
MixNibbles
S^{i+1}

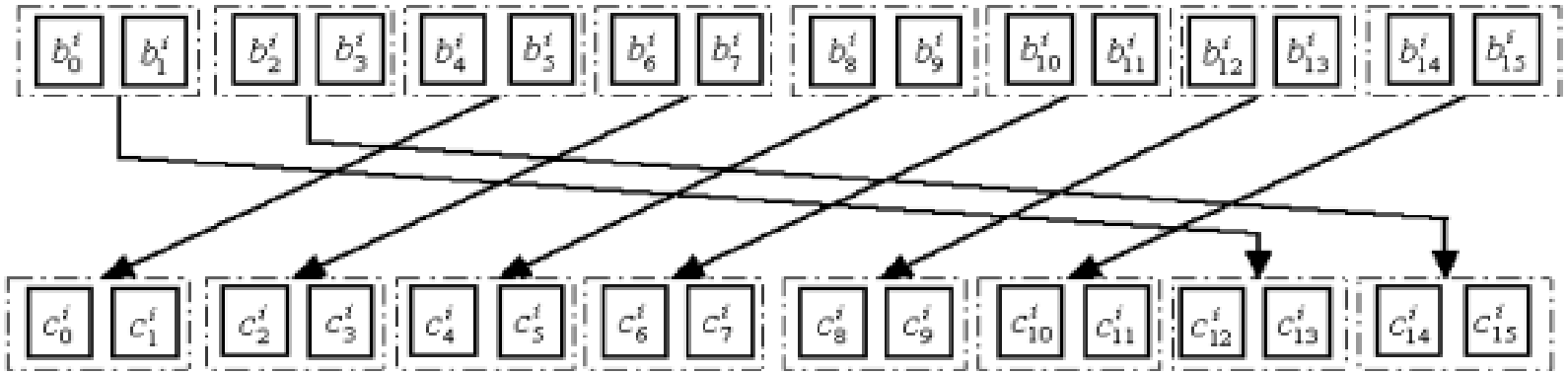
KLEIN

- SubNibbles
  - 4비트 S-box 사용

Table 1: The 4-Bit S-box used in KLEIN.

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

- RotateNibbles
  - 8비트 Rotate



KLEIN

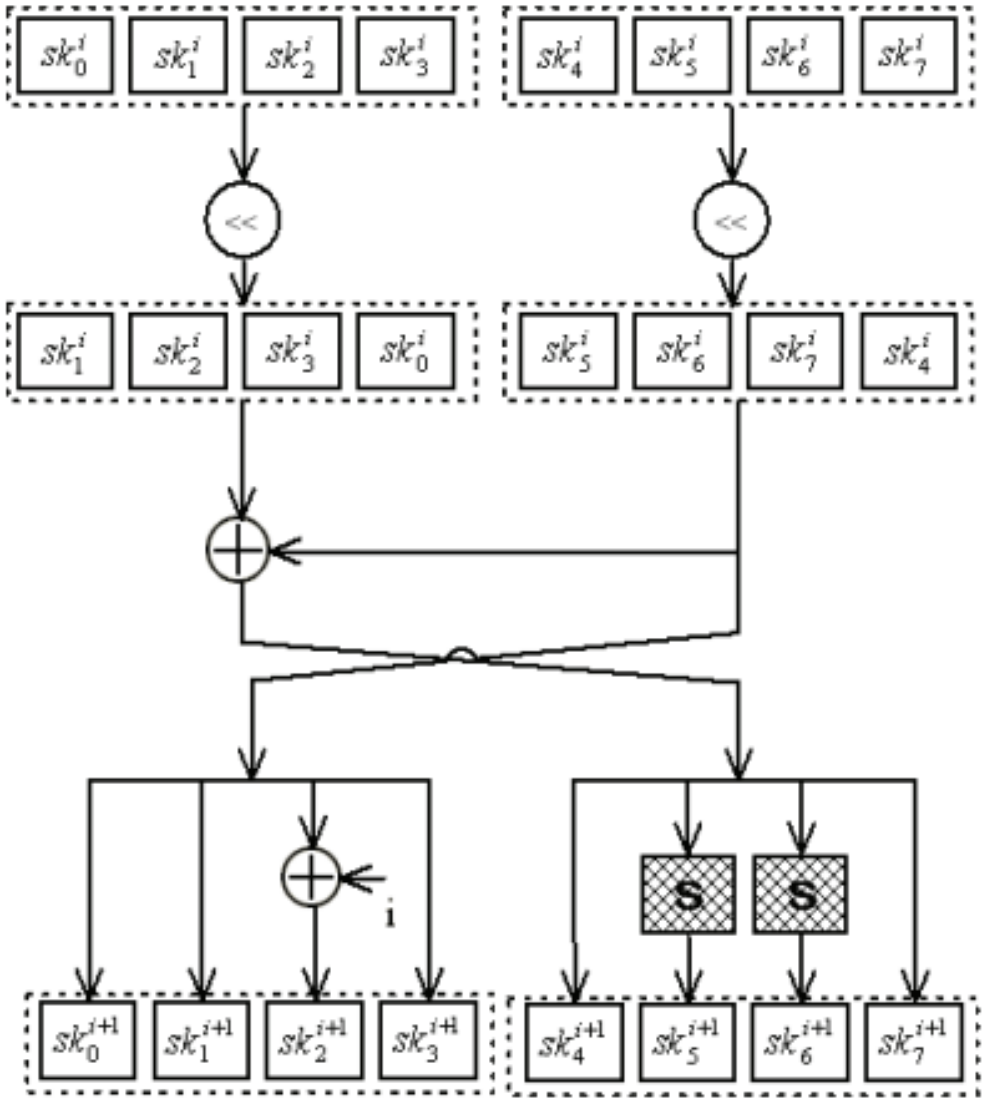
- MixNibbles
  - MixColumns 연산
  - 2부분으로 나눠서 진행

- 연산
  - Modulo $x^4 + 1$
  - $c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$

$$\begin{bmatrix} s_0^{i+1} || s_1^{i+1} \\ s_2^{i+1} || s_3^{i+1} \\ s_4^{i+1} || s_5^{i+1} \\ s_6^{i+1} || s_7^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} c_0^i || c_1^i \\ c_2^i || c_3^i \\ c_4^i || c_5^i \\ c_6^i || c_7^i \end{bmatrix}, \begin{bmatrix} s_8^{i+1} || s_9^{i+1} \\ s_{10}^{i+1} || s_{11}^{i+1} \\ s_{12}^{i+1} || s_{13}^{i+1} \\ s_{14}^{i+1} || s_{15}^{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} c_8^i || c_9^i \\ c_{10}^i || c_{11}^i \\ c_{12}^i || c_{13}^i \\ c_{14}^i || c_{15}^i \end{bmatrix}$$

KLEIN

- KeySchedule
  - Shift 연산
  - XOR 연산
  - Sbox 연산



KLEIN

```
//add round key;
state[0] = state[0] ^ round_key[0];
state[1] = state[1] ^ round_key[1];
state[2] = state[2] ^ round_key[2];
state[3] = state[3] ^ round_key[3];
state[4] = state[4] ^ round_key[4];
state[5] = state[5] ^ round_key[5];
state[6] = state[6] ^ round_key[6];
state[7] = state[7] ^ round_key[7];

//key schedule;
temp_state[0] = round_key[0];
temp_state[1] = round_key[1];
temp_state[2] = round_key[2];
temp_state[3] = round_key[3];
temp_state[4] = round_key[4];

round_key[0] = round_key[5];
round_key[1] = round_key[6];
round_key[2] = round_key[7] ^ i;
round_key[3] = round_key[4];

round_key[4] = temp_state[1] ^ round_key[5];
round_key[5] = sbox8[temp_state[2] ^ round_key[6]];
round_key[6] = sbox8[temp_state[3] ^ round_key[7]];
round_key[7] = temp_state[0] ^ temp_state[4];
```

```
//substitute nibbles with the byte-oriented sbox;
state[0] = sbox8[state[0]];
state[1] = sbox8[state[1]];
state[2] = sbox8[state[2]];
state[3] = sbox8[state[3]];
state[4] = sbox8[state[4]];
state[5] = sbox8[state[5]];
state[6] = sbox8[state[6]];
state[7] = sbox8[state[7]];

//the RotateNibbles step, left shift two bytes;
temp_state[0] = state[2];
temp_state[1] = state[3];
temp_state[2] = state[4];
temp_state[3] = state[5];
temp_state[4] = state[6];
temp_state[5] = state[7];
temp_state[6] = state[0];
temp_state[7] = state[1];
```

KLEIN

```
//an efficient MixNibbles implementation for AES, Book Page 54;
u = temp_state[0] ^ temp_state[1] ^ temp_state[2] ^ temp_state[3];
v = temp_state[0] ^ temp_state[1];
v = multiply2[v];
state[0] = temp_state[0] ^ v ^ u;

v = temp_state[1] ^ temp_state[2];
v = multiply2[v];
state[1] = temp_state[1] ^ v ^ u;

v = temp_state[2] ^ temp_state[3];
v = multiply2[v];
state[2] = temp_state[2] ^ v ^ u;

v = temp_state[3] ^ temp_state[0];
v = multiply2[v];
state[3] = temp_state[3] ^ v ^ u;
```

```
u = temp_state[4] ^ temp_state[5] ^ temp_state[6] ^ temp_state[7];
v = temp_state[4] ^ temp_state[5];
v = multiply2[v];
state[4] = temp_state[4] ^ v ^ u;

v = temp_state[5] ^ temp_state[6];
v = multiply2[v];
state[5] = temp_state[5] ^ v ^ u;

v = temp_state[6] ^ temp_state[7];
v = multiply2[v];
state[6] = temp_state[6] ^ v ^ u;

v = temp_state[7] ^ temp_state[4];
v = multiply2[v];
state[7] = temp_state[7] ^ v ^ u;
```

Q & A