

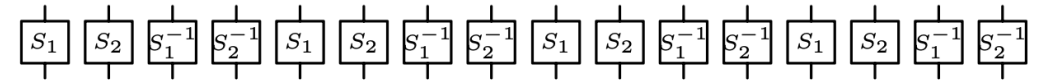
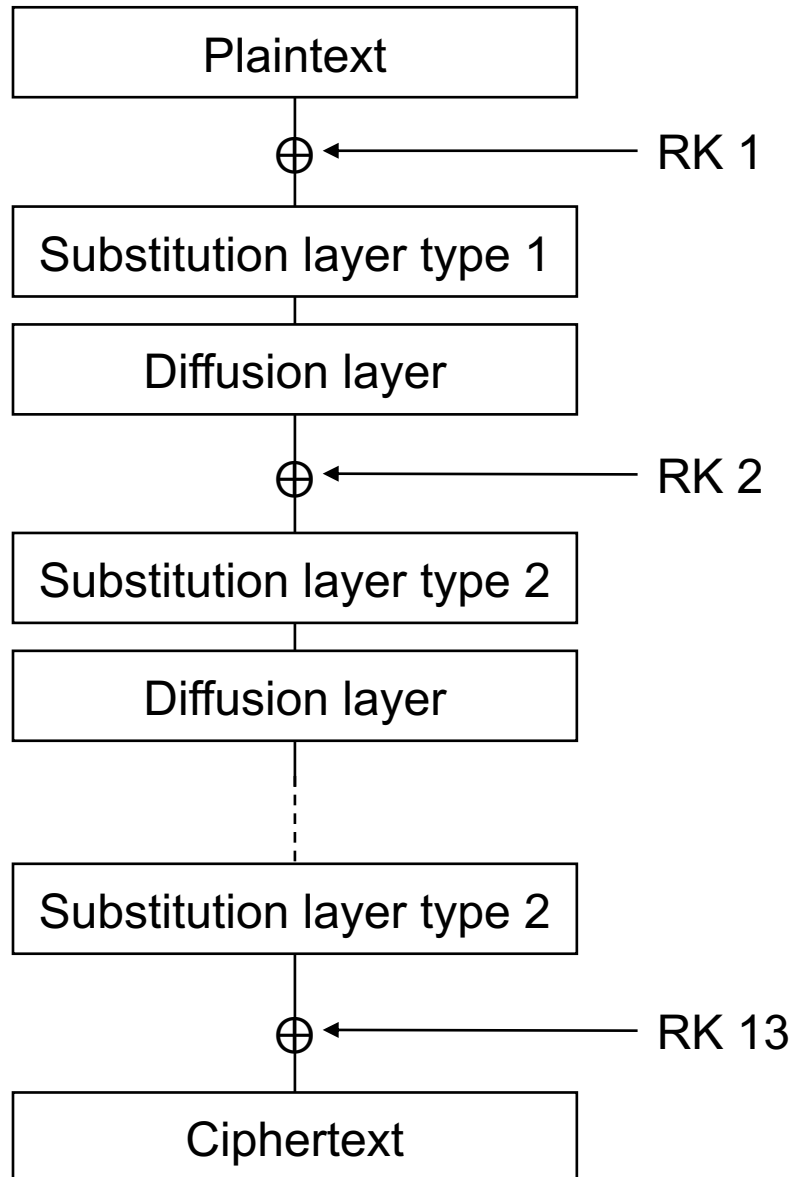
ARMv8 ARIA 블록 암호 병렬 구현

<https://youtu.be/ROwUPIfHllg>

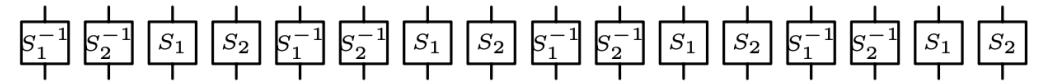
ARIA block cipher 소개

- 국내의 국가보안기술연구소에서 개발한 블록 암호.
- 블록 길이는 128-bit를 사용하며, 키 길이는 128, 192, 256-bit 3가지 길이를 지원하며, 각 키 길이에 따라 12, 14, 16라운드를 진행함.

ARIA block cipher 소개



(a) S-box layer type 1

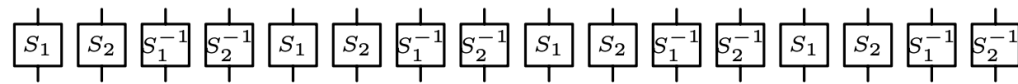


(b) S-box layer type 2

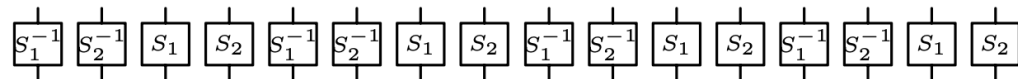
$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix}$$

ARMv8 ARIA

- Substitution layer에서 서로 다른 4개의 Sbox를 사용하기 때문에 TBL 명령어로 한번에 치환하는 것에 어려움이 있음.
- 따라서 같은 Sbox를 사용하는 state를 하나의 레지스터에 모아서 구현을 진행.
- 4PT / 16PT 두가지 병렬 구현



(a) S-box layer type 1

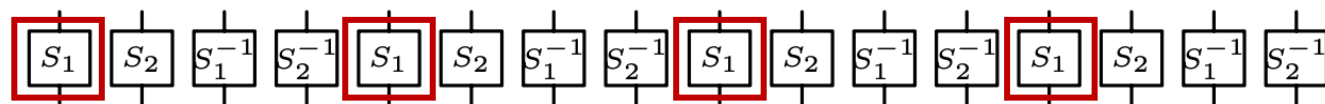


(b) S-box layer type 2

ARMv8 ARIA 4PT

- 4PT는 4개의 블록을 병렬 암호화함.

V0	A0	A4	A8	A12	B0	B4	B8	B12	C0	C4	C8	C12	D0	D4	D8	D12
V1	A1	A5	A9	A13	B1	B5	B9	B13	C1	C5	C9	C13	D1	D5	D9	D13
V2	A2	A6	A10	A14	B2	B6	B10	B14	C2	C6	C10	C14	D2	D6	D10	D14
V3	A3	A7	A11	A15	B3	B7	B11	B15	C3	C7	C11	C15	D3	D7	D11	D15



(a) S-box layer type 1

ARMv8 ARIA 4PT

- Diffusion layer와 addroundkey 구현을 위해 레지스터 정렬을 추가하여 구현

$$\begin{aligned}
 T_1 &= x_4 \oplus x_5 \oplus x_{10} \oplus x_{15}, & T_2 &= x_3 \oplus x_6 \oplus x_9 \oplus x_{16}, \\
 y_1 &= x_7 \oplus x_9 \oplus x_{14} \oplus T_1, & y_2 &= x_8 \oplus x_{10} \oplus x_{13} \oplus T_2, \\
 y_6 &= x_2 \oplus x_{11} \oplus x_{16} \oplus T_1, & y_5 &= x_1 \oplus x_{12} \oplus x_{15} \oplus T_2, \\
 y_{12} &= x_3 \oplus x_8 \oplus x_{13} \oplus T_1, & y_{11} &= x_4 \oplus x_7 \oplus x_{14} \oplus T_2, \\
 y_{15} &= x_1 \oplus x_6 \oplus x_{12} \oplus T_1, & y_{16} &= x_2 \oplus x_5 \oplus x_{11} \oplus T_2,
 \end{aligned}$$

$$\begin{aligned}
 T_3 &= x_2 \oplus x_7 \oplus x_{12} \oplus x_{13}, & T_4 &= x_1 \oplus x_8 \oplus x_{11} \oplus x_{14}, \\
 y_3 &= x_5 \oplus x_{11} \oplus x_{16} \oplus T_3, & y_4 &= x_6 \oplus x_{12} \oplus x_{15} \oplus T_4, \\
 y_8 &= x_4 \oplus x_9 \oplus x_{14} \oplus T_3, & y_7 &= x_3 \oplus x_{10} \oplus x_{13} \oplus T_4, \\
 y_{10} &= x_1 \oplus x_6 \oplus x_{15} \oplus T_3, & y_9 &= x_2 \oplus x_5 \oplus x_{16} \oplus T_4, \\
 y_{13} &= x_3 \oplus x_8 \oplus x_{10} \oplus T_3, & y_{14} &= x_4 \oplus x_{10} \oplus x_9 \oplus T_4.
 \end{aligned}$$

V0	A0	A1	A2	A3	B0	B1	B2	B3	C0	C1	C2	C3	D0	D1	D2	D3
V1	A4	A5	A6	A7	B4	B5	B6	B7	C4	C5	C6	C7	D4	D5	D6	D7
V2	A8	A9	A10	A11	B8	B9	B10	B11	C8	C9	C10	C11	D8	D9	D10	D11
V3	A12	A13	A14	A15	B12	B13	B14	B15	C12	C13	C14	C15	D12	D13	D14	D15

ARMv8 ARIA 4PT – plaintext load

- ARM vector 명령어에서 다양한 load 명령어를 지원함.
- 기존의 간단한 LD1 명령어를 사용

V0	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----

- LD4.s {v0-v3} [index], [address]

V0	A0	A1	A2	A3	B0	B1	B2	B3	C0	C1	C2	C3	D0	D1	D2	D3
V1	A4	A5	A6	A7	B4	B5	B6	B7	C4	C5	C6	C7	D4	D5	D6	D7
V2	A8	A9	A10	A11	B8	B9	B10	B11	C8	C9	C10	C11	D8	D9	D10	D11
V3	A12	A13	A14	A15	B12	B13	B14	B15	C12	C13	C14	C15	D12	D13	D14	D15

```
ld4.s {v0-v3}[0], [x0], #16  
ld4.s {v0-v3}[1], [x0], #16  
ld4.s {v0-v3}[2], [x0], #16  
ld4.s {v0-v3}[3], [x0]
```

ARMv8 ARIA 4PT – Diffusion layer

$$\begin{aligned} T_1 &= x_4 \oplus x_5 \oplus x_{10} \oplus x_{15}, & T_2 &= x_3 \oplus x_6 \oplus x_9 \oplus x_{16}, \\ y_1 &= x_7 \oplus x_9 \oplus x_{14} \oplus T_1, & y_2 &= x_8 \oplus x_{10} \oplus x_{13} \oplus T_2, \\ y_6 &= x_2 \oplus x_{11} \oplus x_{16} \oplus T_1, & y_5 &= x_1 \oplus x_{12} \oplus x_{15} \oplus T_2, \\ y_{12} &= x_3 \oplus x_8 \oplus x_{13} \oplus T_1, & y_{11} &= x_4 \oplus x_7 \oplus x_{14} \oplus T_2, \\ y_{15} &= x_1 \oplus x_6 \oplus x_{12} \oplus T_1, & y_{16} &= x_2 \oplus x_5 \oplus x_{11} \oplus T_2, \end{aligned}$$

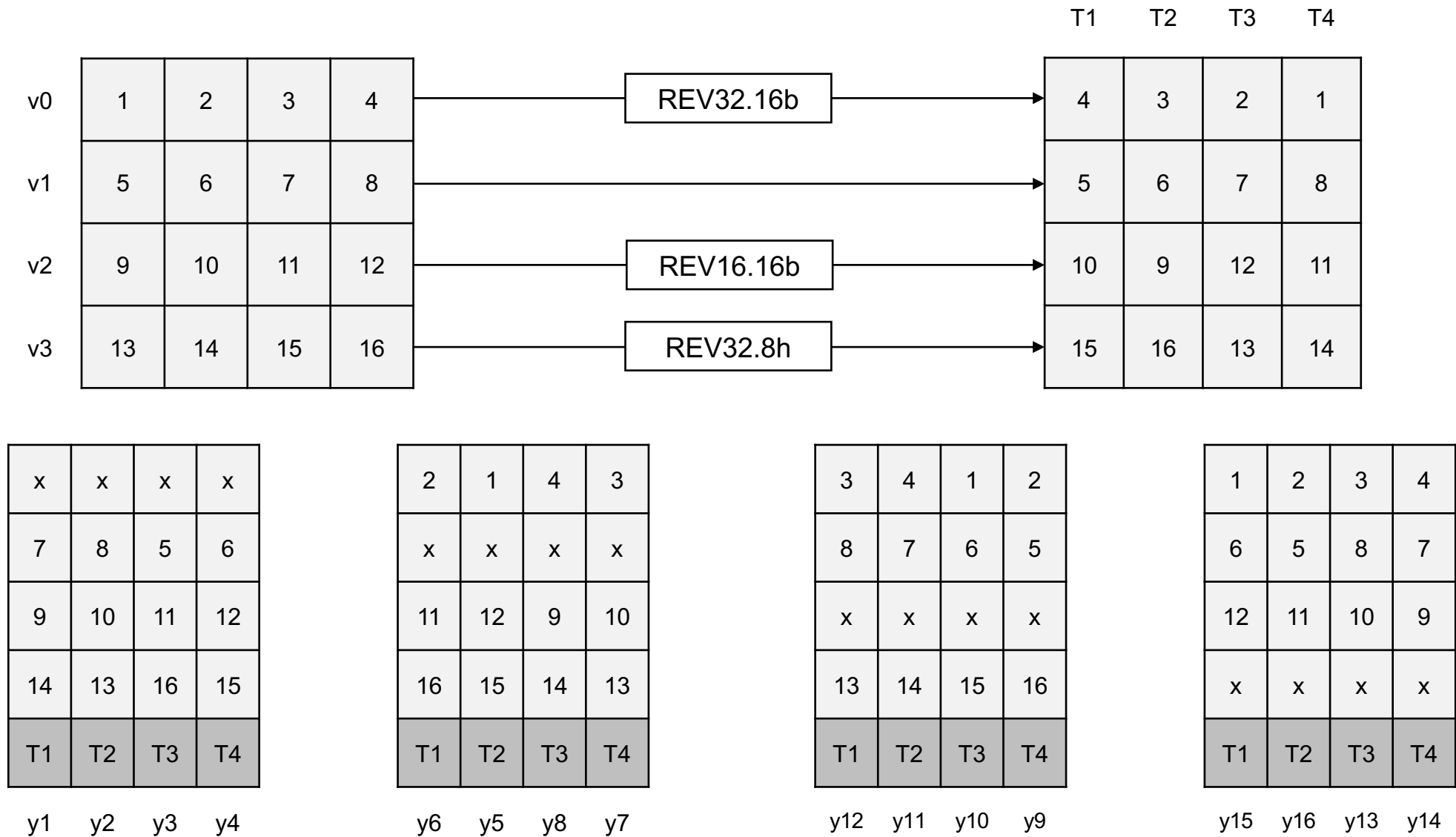
$$\begin{aligned} T_3 &= x_2 \oplus x_7 \oplus x_{12} \oplus x_{13}, & T_4 &= x_1 \oplus x_8 \oplus x_{11} \oplus x_{14}, \\ y_3 &= x_5 \oplus x_{11} \oplus x_{16} \oplus T_3, & y_4 &= x_6 \oplus x_{12} \oplus x_{15} \oplus T_4, \\ y_8 &= x_4 \oplus x_9 \oplus x_{14} \oplus T_3, & y_7 &= x_3 \oplus x_{10} \oplus x_{13} \oplus T_4, \\ y_{10} &= x_1 \oplus x_6 \oplus x_{15} \oplus T_3, & y_9 &= x_2 \oplus x_5 \oplus x_{16} \oplus T_4, \\ y_{13} &= x_3 \oplus x_8 \oplus x_{10} \oplus T_3, & y_{14} &= x_4 \oplus x_7 \oplus x_9 \oplus T_4. \end{aligned}$$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

T1	T2	T3	T4
4	3	2	1
5	6	7	8
10	9	12	11
15	16	13	14

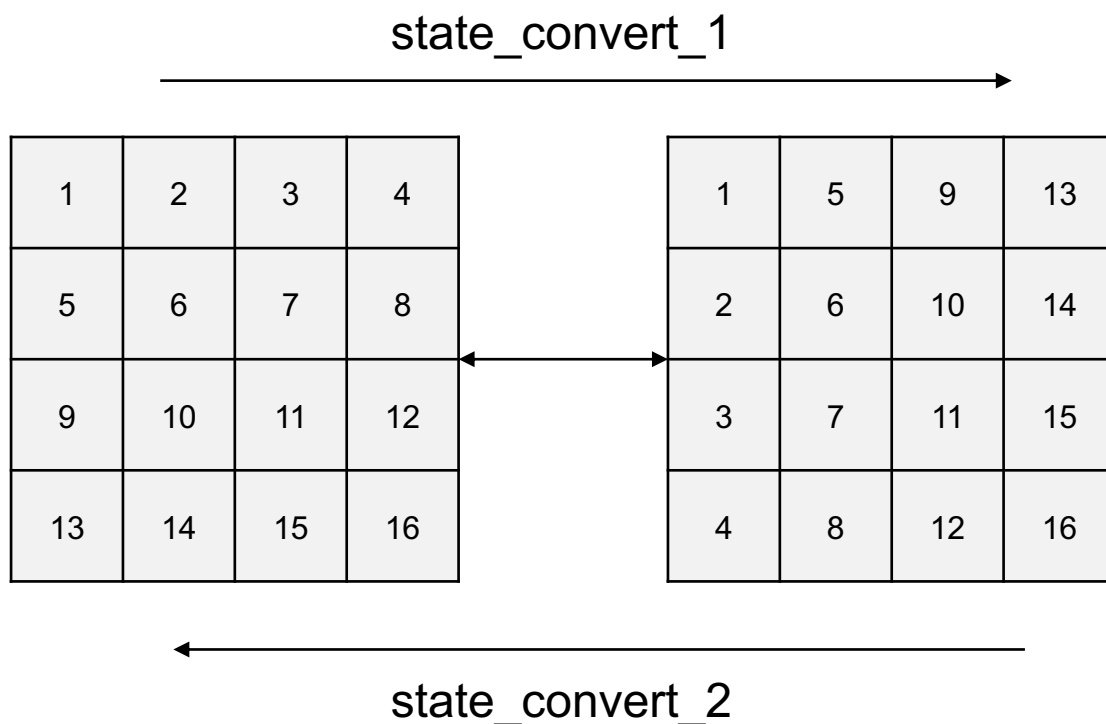
Vn	AT1	AT2	AT3	AT4	BT1	BT2	BT3	BT4	CT1	CT2	CT3	CT4	DT1	DT2	DT3	DT4
----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

ARMv8 ARIA 4PT – Diffusion layer



ARMv8 ARIA 4PT – state 재정렬

- Substitution을 진행하기 위해서 state 재정렬을 진행.
- 메모리에 현재 상태를 저장하고 다시 불러오면서 정렬



```
.macro state_convert_1
//state_convert_1 macro start-----
st4.s {v0-v3}[0], [x1], #16
st4.s {v0-v3}[1], [x1], #16
st4.s {v0-v3}[2], [x1], #16
st4.s {v0-v3}[3], [x1]

sub x1, x1, #48
ld4.16b {v0-v3}, [x1]
//state_convert_1 macro end-----
.endm

.macro state_convert_2
//state_convert_2 macro start-----
st4.16b {v0-v3}, [x1]

ld4.s {v0-v3}[0], [x1], #16
ld4.s {v0-v3}[1], [x1], #16
ld4.s {v0-v3}[2], [x1], #16
ld4.s {v0-v3}[3], [x1]

sub x1, x1, #48
//state_convert_2 macro end-----
.endm
```

ARMv8 ARIA 4PT – Substitution

```
.macro slayer_1
//slayer_1 macro start-----
state_convert_1
sbox v0, x3 //s1
sbox v1, x4 //s2
sbox v2, x5 //s1_i
sbox v3, x6 //s2_i
state_convert_2
//slayer_1 macro end-----
.endm

.macro slayer_2
//slayer_2 macro start-----
state_convert_1
sbox v0, x5 //s1_i
sbox v1, x6 //s2_i
sbox v2, x3 //s1
sbox v3, x4 //s2
state_convert_2
//slayer_2 macro end-----
.endm
```

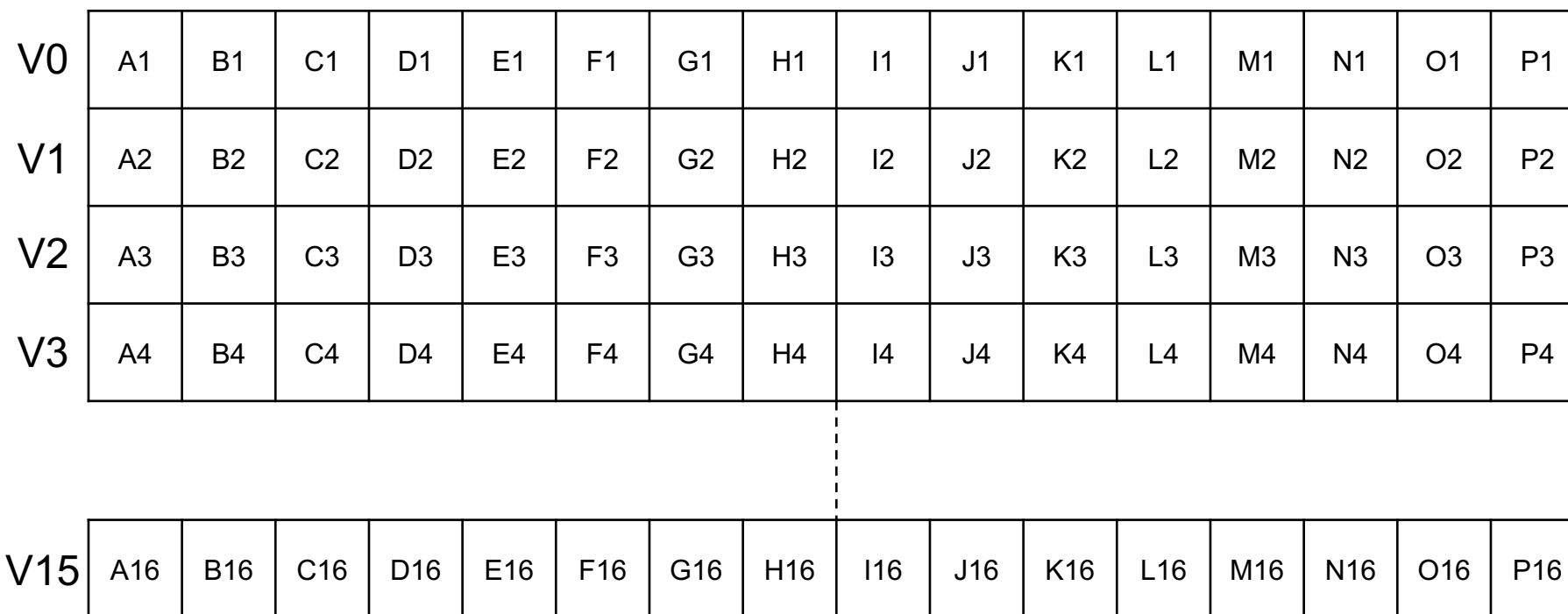
```
.macro sbox reg, address
//sbox macro start-----
ld1.16b {v16-v19}, [\address], #64 //sbox load
ld1.16b {v20-v23}, [\address], #64
ld1.16b {v24-v27}, [\address], #64
ld1.16b {v28-v31}, [\address]

sub \address, \address, #192

sub.16b v7, \reg, v15
tbl.16b \reg, { v16 - v19 }, \reg
sub.16b v6, v7, v15
tbx.16b \reg, { v20 - v23 }, v7
sub.16b v5, v6, v15
tbx.16b \reg, { v24 - v27 }, v6
tbx.16b \reg, { v28 - v31 }, v5
//sbox macro end-----
.endm
```

ARMv8 ARIA 16PT

- 4PT와 마찬가지로 같은 Sbox를 사용하는 state를 하나의 레지스터에 저장하여 구현
- 벡터 레지스터는 128-bit이기 때문에 8-bit 단위로 16개의 블록을 입력으로 병렬 암호화를 진행.



ARMv8 ARIA 16PT – plaintext load

```
.macro pt_load    index
//pt_load macro start -----
ld4.b {v0-v3}[\index], [x0], #4
ld4.b {v4-v7}[\index], [x0], #4
ld4.b {v8-v11}[\index], [x0], #4
ld4.b {v12-v15}[\index], [x0], #4
//pt_load macro end -----
.endm

.macro pt_store    index
//pt_store macro start -----
st4.b {v0-v3}[\index], [x1], #4
st4.b {v4-v7}[\index], [x1], #4
st4.b {v8-v11}[\index], [x1], #4
st4.b {v12-v15}[\index], [x1], #4
//pt_store macro end -----
.endm
```

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

ARMv8 ARIA 16PT - Substitution

```
.macro sbbox s0, s1, s2, s3, address, temp1, temp2, temp40
//sbbox macro start -----
mov.d x10, \temp1[0]
mov.d x11, \temp1[1]
mov.d x12, \temp2[0]
mov.d x13, \temp2[1]
mov.d x14, \temp40[0]
mov.d x15, \temp40[1]
movi.16b \temp40, #0x40

ld1.16b {v16-v19}, [\address], #64    //sbbox load
ld1.16b {v20-v23}, [\address], #64
ld1.16b {v24-v27}, [\address], #64
ld1.16b {v28-v31}, [\address]
sub \address, \address, #192

sub.16b \temp1, \s0, \temp40
tbl.16b \s0, { v16 - v19 }, \s0
sub.16b \temp2, \temp1, \temp40
tbx.16b \s0, { v20 - v23 }, \temp1
sub.16b \temp1, \temp2, \temp40
tbx.16b \s0, { v24 - v27 }, \temp2
tbx.16b \s0, { v28 - v31 }, \temp1

sub.16b \temp1, \s1, \temp40
tbl.16b \s1, { v16 - v19 }, \s1
sub.16b \temp2, \temp1, \temp40
tbx.16b \s1, { v20 - v23 }, \temp1
sub.16b \temp1, \temp2, \temp40
tbx.16b \s1, { v24 - v27 }, \temp2
tbx.16b \s1, { v28 - v31 }, \temp1
```

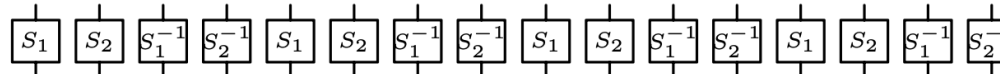
```
sub.16b \temp1, \s2, \temp40
tbl.16b \s2, { v16 - v19 }, \s2
sub.16b \temp2, \temp1, \temp40
tbx.16b \s2, { v20 - v23 }, \temp1
sub.16b \temp1, \temp2, \temp40
tbx.16b \s2, { v24 - v27 }, \temp2
tbx.16b \s2, { v28 - v31 }, \temp1

sub.16b \temp1, \s3, \temp40
tbl.16b \s3, { v16 - v19 }, \s3
sub.16b \temp2, \temp1, \temp40
tbx.16b \s3, { v20 - v23 }, \temp1
sub.16b \temp1, \temp2, \temp40
tbx.16b \s3, { v24 - v27 }, \temp2
tbx.16b \s3, { v28 - v31 }, \temp1

mov.d \temp1[0], x10
mov.d \temp1[1], x11
mov.d \temp2[0], x12
mov.d \temp2[1], x13
mov.d \temp40[0], x14
mov.d \temp40[1], x15
//sbbox macro end -----
.endm
```

```
.macro slayer_1
//slayer_1 macro start -----
sbbox v0, v4, v8, v12, x3, v1, v2, v3
sbbox v1, v5, v9, v13, x4, v0, v2, v3
sbbox v2, v6, v10, v14, x5, v0, v1, v3
sbbox v3, v7, v11, v15, x6, v0, v1, v2
//slayer_1 macro end -----
.endm

.macro slayer_2
//slayer_2 macro start -----
sbbox v0, v4, v8, v12, x5, v1, v2, v3
sbbox v1, v5, v9, v13, x6, v0, v2, v3
sbbox v2, v6, v10, v14, x3, v0, v1, v3
sbbox v3, v7, v11, v15, x4, v0, v1, v2
//slayer_2 macro end -----
.endm
```



(a) S-box layer type 1

ARMv8 ARIA 16PT – diffusion


$$\begin{aligned}T_1 &= x_4 \oplus x_5 \oplus x_{10} \oplus x_{15}, & T_2 &= x_3 \oplus x_6 \oplus x_9 \oplus x_{16}, \\y_1 &= x_7 \oplus x_9 \oplus x_{14} \oplus T_1, & y_2 &= x_8 \oplus x_{10} \oplus x_{13} \oplus T_2, \\y_6 &= x_2 \oplus x_{11} \oplus x_{16} \oplus T_1, & y_5 &= x_1 \oplus x_{12} \oplus x_{15} \oplus T_2, \\y_{12} &= x_3 \oplus x_8 \oplus x_{13} \oplus T_1, & y_{11} &= x_4 \oplus x_7 \oplus x_{14} \oplus T_2, \\y_{15} &= x_1 \oplus x_6 \oplus x_{12} \oplus T_1, & y_{16} &= x_2 \oplus x_5 \oplus x_{11} \oplus T_2,\end{aligned}$$

$$\begin{aligned}T_3 &= x_2 \oplus x_7 \oplus x_{12} \oplus x_{13}, & T_4 &= x_1 \oplus x_8 \oplus x_{11} \oplus x_{14}, \\y_3 &= x_5 \oplus x_{11} \oplus x_{16} \oplus T_3, & y_4 &= x_6 \oplus x_{12} \oplus x_{15} \oplus T_4, \\y_8 &= x_4 \oplus x_9 \oplus x_{14} \oplus T_3, & y_7 &= x_3 \oplus x_{10} \oplus x_{13} \oplus T_4, \\y_{10} &= x_1 \oplus x_6 \oplus x_{15} \oplus T_3, & y_9 &= x_2 \oplus x_5 \oplus x_{16} \oplus T_4, \\y_{13} &= x_3 \oplus x_8 \oplus x_{10} \oplus T_3, & y_{14} &= x_4 \oplus x_7 \oplus x_9 \oplus T_4.\end{aligned}$$

성능 결과

- 측정 방법

- $\text{double cycle} = \text{time}(\text{초단위}) / \text{반복횟수} / 3.2 \times 10^{10} (\text{동작주파수});$

General			Image ⇅	Semiconductor technology				Computer architecture		Performance core		
Name ⇅	Codename ⇅	Part No. ⇅		Node ⇅	Manufacturer ⇅	Transistors count ⇅	Die size ⇅	CPU ISA ⇅	Bit width ⇅	Core name ⇅	Core no. ⇅	Core speed ⇅
M1	APL1102	T8103				16 billion	120 mm ² <small>[152]</small>				4	3.20 GHz

```
start = (double)clock() /CLOCKS_PER_SEC;
for(i=0; i<10000000; i++)
    ARIA_Parallel_4PT(PT, OT, RK, S1, S2, S1_i, S2_i);
end = (double)clock()/CLOCKS_PER_SEC;
double time = end - start;
```

```
double cycle = time / 10000000 / 3.2*1000000000;
printf("result time = %lf\n", time);
printf("result cycle = %lf\n", cycle);
printf("result cpb = %lf\n", cycle/64);
```


성능 결과

- ARIA reference C code
 - Cycle : 226.20
 - Cpb : 14.14
- ARIA 4PT
 - Cycle : 111.52
 - Cpb : 1.74
- ARIA 16PT
 - Cycle : 146.20
 - Cpb : 0.57

감 사 합 니 다