# ASCON

https://www.youtube.com/watch?v=u8nkSnau-sE

# ASCON

- **ASCON** – NIST 경량암호 표준화로 선정된 암호 제품군
  - NIST의 권장사항은 Ascon-128 또는 Ascon-128a와 결합된 Ascon-Hash.
  - 모든 체계는 128비트 보안을 제공하고 내부적으로 동일한 320비트 순열을 사용하므로 AEAD와 Hash모두 구현하기 좋음.

    - **AEAD**
      - **ASCON -128**
      - **ASCON - 128a**
    - **Hash**
      - **ASCON - HASH**
      - **ASCON - XOF**
    - **Variant**
      - **ASCON - 80pq**

# ASCON parameter

- **ASCON AEAD**

| Name | Algorithms | Bit size | | | | Rounds | |
|---|---|---|---|---|---|---|---|
| | | Key | Nonce | Tag | block | $p^a$ | $p^b$ |
| ASCON-128 | $\mathcal{E}, D_{128,64,12,6}$ | 128 | 128 | 128 | 64 | 12 | 6 |
| ASCON-128a | $\mathcal{E}, D_{128,128,12,8}$ | 128 | 128 | 128 | 128 | 12 | 8 |

- **ASCON HASH**

| Name | Algorithms | Bit size | | Rounds | |
|---|---|---|---|---|---|
| | | Hash | block | $p^a$ | $p^b$ |
| ASCON-HASH | $\chi_{256,64,12}$ with $\ell = 256$ | 256 | 64 | 12 | 0 |

# ASCON - AEAD

- *Initialization, processing Associated Data, Processing Plaintext, Finalization*

  모든 프로세스마다 Permutation 포함.

- $IV_{k,r,a,b} \leftarrow k \parallel r \parallel a \parallel b \parallel 0^{160-k} = \begin{cases} 80400c0600000000 & \text{for ASCON-128} \\ 80800c0800000000 & \text{for ASCON-128a} \\ a0400c06 & \text{for ASCON-80pq} \end{cases}$

  ($k$ = key size , $r$ = data block , ($a$ , $b$) = round number)

- $S \leftarrow IV_{k,r,a,b} \parallel K \parallel N$

  ($K$ = key , $N$ = Nonce)

**Initialization**
$S \leftarrow \mathrm{IV}_{k,r,a,b} \parallel K \parallel N$
$S \leftarrow p^a(S) \oplus (0^{320-k} \parallel K)$
**Processing Associated Data**
**if** $|A| > 0$ **then**
$A_1 \ldots A_s \leftarrow r$-bit blocks of $A \| 1 \| 0^*$
**for** $i = 1, \ldots, s$ **do**
$S \leftarrow p^b((S_r \oplus A_i) \parallel S_c)$
$S \leftarrow S \oplus (0^{319} \parallel 1)$
**Processing Plaintext**
$P_1 \ldots P_t \leftarrow r$-bit blocks of $P \| 1 \| 0^*$
**for** $i = 1, \ldots, t-1$ **do**
$S_r \leftarrow S_r \oplus P_i$
$C_i \leftarrow S_r$
$S \leftarrow p^b(S)$
$S_r \leftarrow S_r \oplus P_t$
$\tilde{C}_t \leftarrow \lfloor S_r \rfloor_{|P| \bmod r}$
**Finalization**
$S \leftarrow p^a(S \oplus (0^r \parallel K \parallel 0^{320-r-k}))$
$T \leftarrow \lceil S \rceil^{128} \oplus \lceil K \rceil^{128}$
**return** $C_1 \parallel \ldots \parallel C_{t-1} \parallel \tilde{C}_t, T$

# ASCON - HASH

- *Initialization, Absorbing, Squeezing*

  모든 프로세스마다 Permutation 포함.

- $IV_{k,r,a,b} \leftarrow 0^8 \parallel r \parallel a \parallel 0^8 \parallel h =$ $\begin{cases} 00400c0000000000 & \text{for ASCON-XOF} \\ 00400c0000000100 & \text{for ASCON-HASH} \end{cases}$

  ($r$ = data block , $a$ = round number, $h$ = output length limit)

- $S \leftarrow p^a(IV_{k,r,a,b} \parallel 0^{256})$

- $S \leftarrow \begin{matrix} ee9398aadb67f03d \parallel \\ 8bb21831c60f1002 \parallel \\ b48a92db98d5da62 \parallel \\ 43189921b8f8e3e8 \parallel \\ 348fa5c9d525e140 \parallel \end{matrix}$

**Initialization**
$S \leftarrow p^a(\mathrm{IV}_{h,r,a} \parallel 0^c)$
**Absorbing**
$M_1 \ldots M_s \leftarrow M \parallel 1 \parallel 0^*$
**for** $i = 1, \ldots, s$ **do**
$S \leftarrow p^a((S_r \oplus M_i) \parallel S_c)$
**Squeezing**
**for** $i = 1, \ldots, t = \lceil \ell / r \rceil$ **do**
$H_i \leftarrow S_r$
$S \leftarrow p^a(S)$
**return** $\lfloor H_1 \parallel \ldots \parallel H_t \rfloor_\ell$

# Permutation

- **Permutation**

  ASCON의 주요 구성요소는 320bit 순열

  $P = P_L \circ P_S \circ P_C$

  320비트 S는 5개의 64비트 레지스터 워드 $x_i, S = x_0 \parallel x_1 \parallel x_2 \parallel x_3 \parallel x_4$로 분할 ( $x_0 = MSB$ , $x_4 = LSB$)

  - **Addition of Constants**($P_c$)

    $x_2$ 에 constant $C_r$ add. $x_2 \leftarrow x_2 \oplus C_r$

    $$r = i \ for \ p_a \quad \& \quad r = i + a - b \ for \ p_b$$

| $p^{12}$ | $p^8$ | $p^6$ | Constant $c_r$ | $p^{12}$ | $p^8$ | $p^6$ | Constant $c_r$ |
|---|---|---|---|---|---|---|---|
| 0 | | | 00000000000000000f0 | 6 | 2 | 0 | 00000000000000000096 |
| 1 | | | 00000000000000000e1 | 7 | 3 | 1 | 00000000000000000087 |
| 2 | | | 00000000000000000d2 | 8 | 4 | 2 | 00000000000000000078 |
| 3 | | | 00000000000000000c3 | 9 | 5 | 3 | 00000000000000000069 |
| 4 | 0 | | 00000000000000000b4 | 10 | 6 | 4 | 0000000000000000005a |
| 5 | 1 | | 00000000000000000a5 | 11 | 7 | 5 | 0000000000000000004b |

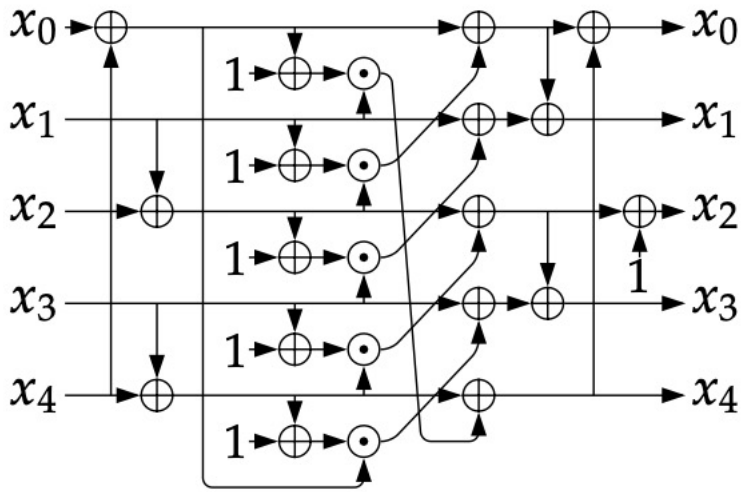
# Permutation

- **Substitution Layer ($P_S$)**

  5비트 S-box 를 64개의 병렬로 구성하여 상태 S를 5개 레지스터의 각 비트 슬라이스로 변환

Table 5: Ascon's 5-bit S-box $\mathcal{S}$ as a lookup table.

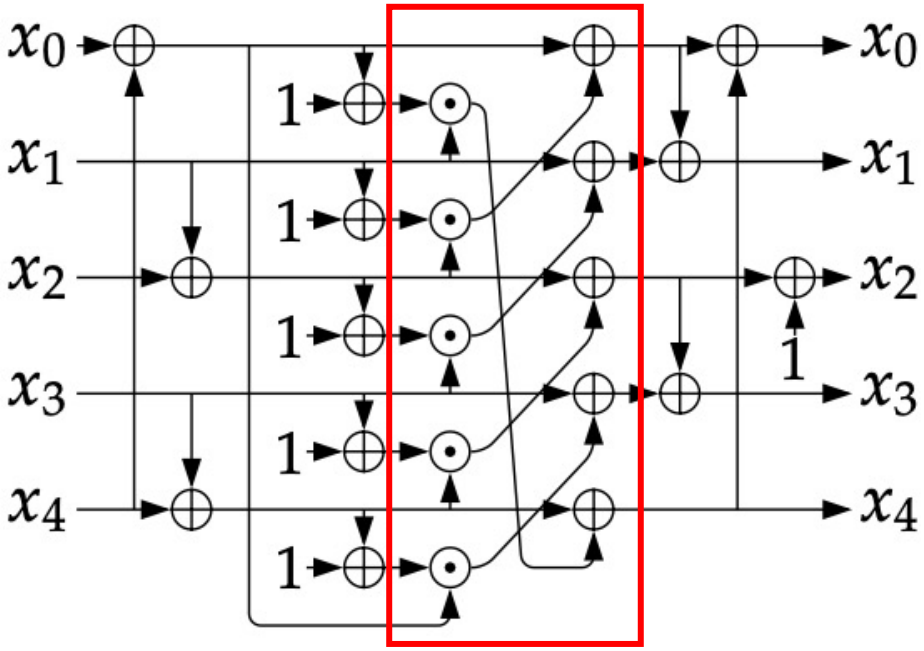
| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1a | 1b | 1c | 1d | 1e | 1f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{S}(x)$ | 4 | b | 1f | 14 | 1a | 15 | 9 | 2 | 1b | 5 | 8 | 12 | 1d | 3 | 6 | 1c | 1e | 13 | 7 | e | 0 | d | 11 | 18 | 10 | c | 1 | 19 | 16 | a | f | 17 |



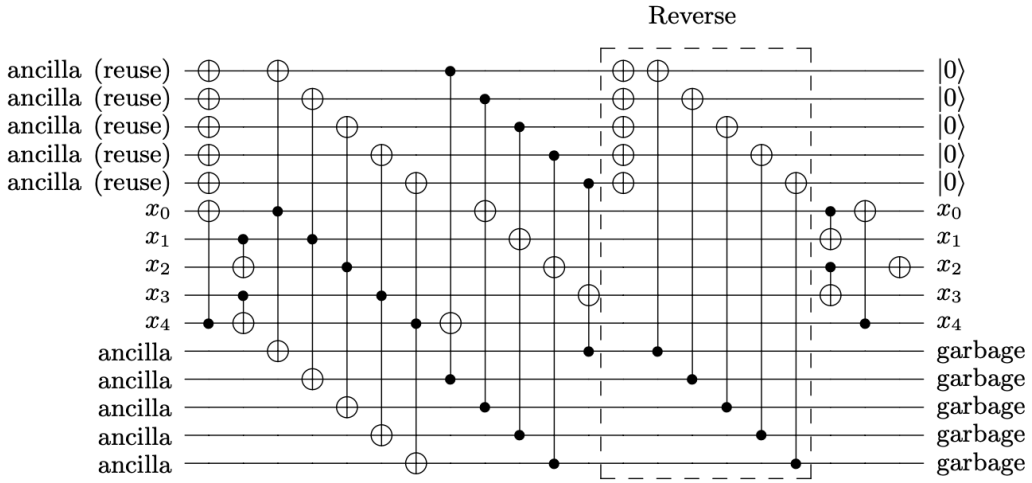

$$
\begin{aligned}
&x_0 = x_0 \oplus x_4,\ x_4 = x_4 \oplus x_3,\ x_2 = x_2 \oplus x_1,\\
&t_0 = x_0,\ t_1 = x_1,\ t_2 = x_2,\ t_3 = x_3,\ t_4 = x_4,\\
&t_0 = \sim t_0,\ t_1 = \sim t_1,\ t_2 = \sim t_2,\ t_3 = \sim t_3,\ t_4 = \sim t_4,\\
&t_0 = t_0 \cdot x_1,\ t_1 = t_1 \cdot x_2,\ t_2 = t_2 \cdot x_3,\ t_3 = t_3 \cdot x_4,\ t_4 = t_4 \cdot x_0,\\
&x_0 = x_0 \oplus t_1,\ x_1 = x_1 \oplus t_2,\ x_2 = x_2 \oplus t_3,\ x_3 = x_3 \oplus t_4,\\
&x_4 = x_4 \oplus t_0,\ x_1 = x_1 \oplus x_0,\ x_0 = x_0 \oplus x_4,\\
&x_3 = x_3 \oplus x_2,\ x_2 = \sim x
\end{aligned}
$$

# Permutation

- **Substitution Layer ($P_S$)**




Ancilla qubit를 더 사용하여 병렬로 연산
→ Toffoli depth 와 Full depth 부분에서 최적화




```
for i in range(64):
    Toffoli_gate(eng, ancilla_x1[i], new_ancilla_x2[i], x0[i])
for i in range(64):
    Toffoli_gate(eng, ancilla_x2[i], new_ancilla_x3[i], x1[i])
for i in range(64):
    Toffoli_gate(eng, ancilla_x3[i], new_ancilla_x4[i], x2[i])
for i in range(64):
    Toffoli_gate(eng, ancilla_x0[i], new_ancilla_x1[i], x4[i])
for i in range(64):
    Toffoli_gate(eng, ancilla_x4[i], new_ancilla_x0[i], x3[i])
```

# Permutation

- **Linear Layer ($P_L$)**

  각 64비트 레지스터 워드 $x_i$ 내에서 선형 연산 진행

$$
\begin{aligned}
x_0 &\leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &\leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &\leftarrow x_2 \oplus (x_2 \ggg 01) \oplus (x_2 \ggg 06) \\
x_3 &\leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &\leftarrow x_4 \oplus (x_4 \ggg 07) \oplus (x_4 \ggg 41)
\end{aligned}
$$

```
def LinearDiffusion_Layer(eng,x0,x1,x2,x3,x4):
    new_x0 = eng.allocate_qureg(64)
    new_x1 = eng.allocate_qureg(64)
    new_x2 = eng.allocate_qureg(64)
    new_x3 = eng.allocate_qureg(64)
    new_x4 = eng.allocate_qureg(64)
```

- **Permutation 함수는 라운드 수만큼 $P_C$ - $P_S$ - $P_L$ 순서로 반복**

# 현재 진행사항

- **ASCON AEAD 중 ASCON-128에 대해 양자 구현 완료**
  - **ASCON -128a 에 대해서도 구현해 볼 예정**

- **ASCON-HASH 양자 구현 진행중**
  - **구현 후 기존 구현과 자원 추정 비교**
  - **Permutation 구현에서의 최적화 중요**

# Q & A