SPARKLE

발표자: 양유진

링크: https://youtu.be/YZ-Bj3otUw8

1. Introduction

- 사물인터넷(IoT)이 발달 → 보안 문제 대두됨
- 메모리 크기, 전력 소비량 등 제약 받는 환경에서 사용할 수 있는 경량암호를 표준화하기 위하여 NIST에서 2015년부터 NIST 경량암호 공모전 LWC Standardization을 열었음
- SPARKLE은 LWC Finalist에 든 알고리즘임
- 현재 결선 진출자까지 발표되었고, 2022년 5월에 5차 경량 암호화 워크샵이 개최될 예정임

2. Sponge construction

Sponge 구조 기반

- 임의의 길이의 입력(bit stream)을 사용하여 원하는 길이의 출력(bit stream)을 생성하는 유한한 내부 상태를 가짐.
- 안전성은 내부상태길이에 의존함
- 해시함수, AEAD, MAC, 스트림 암호, 의사 난수 생성기 등을 만드는 데 사용할 수 있음.
- [장점1] bitrate, capacity를 적절히 설정 → 효율성, 안전성 유연하게 조절 가능
- [장점2] 키 스케줄이 불필요함 → 메모리 관점에서 이득 ⇒ 경량기기에 적합함.

작동과정

1) 흡수단계(absorbing phase)

메시지를 초기상태(state), 내부상태(치환을 거쳐 생성됨)와 XOR 연산함

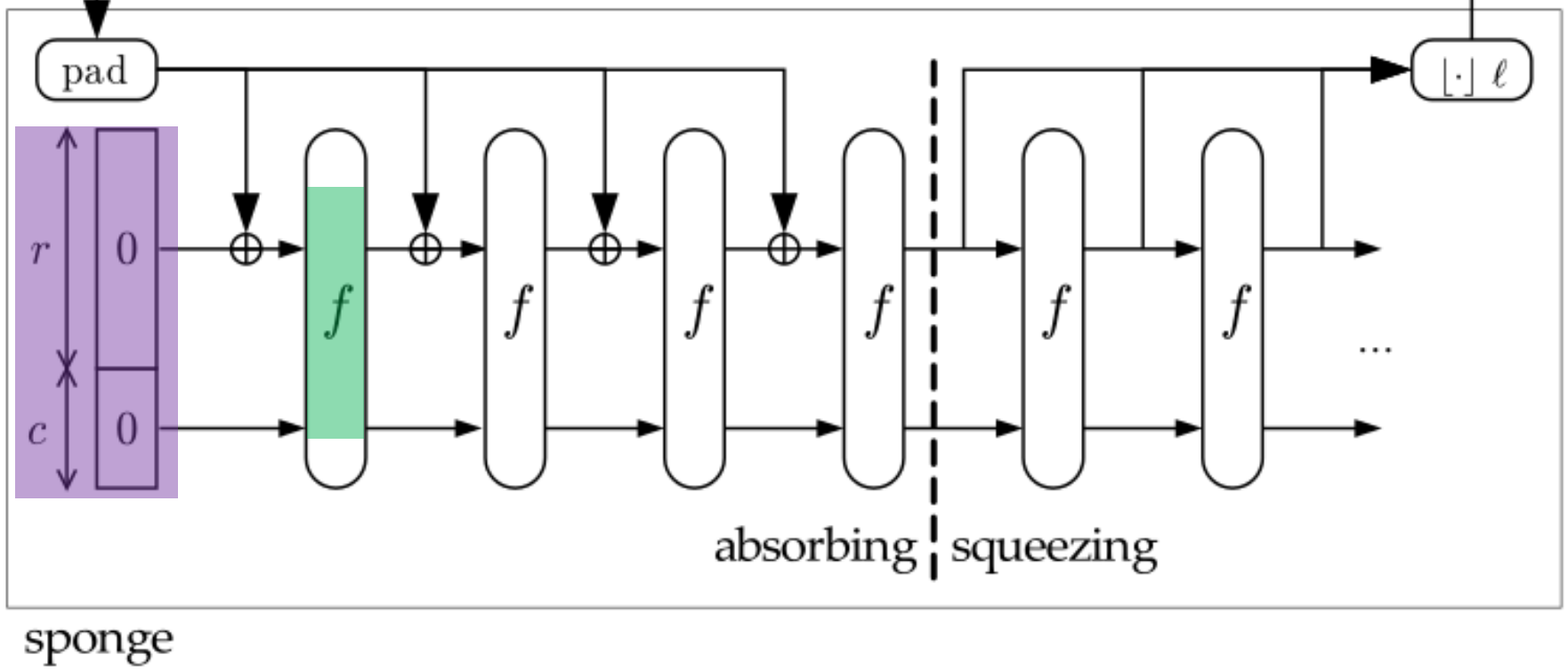
2) 압착단계(squeezing phase)

모든 메시지에서 1단계를 수행한 이후, 생성된 내부상태의 일부 bit를 추출하여 출력값 생성함

2. Sponge construction

M 입력 메시지 블록

Z 해시된 출력 블록



상태 메모리

r: bit-rate
c: capacity

상태 메모리 갱신함
$f: \{0,1\}^b \rightarrow \{0,1\}^b$

3. SPARKLE

SPARKLE (SPARx, but Key LEss)

- ARX 설계를 기반으로 하는 암호화 순열 제품군(family of cryptographic permutation)
- 블록암호 Sparx에서 뻗어나왔음

Efficiency

- 상태(state) 크기가 작음: 상태 크기는 대칭알고리즘에서 RAM 소비량, 실리콘 영역을 크게 결정함
- 극도로 가벼운 순열: 작은 실리콘 영역, 낮은 전력 소비 → 최적화 가능
- 보안 수준 전반에 걸쳐 일관성을 갖도록 설계됨: 소프트웨어 구현을 용이하게 함
- 병렬처리 → 빠른 속도

Security

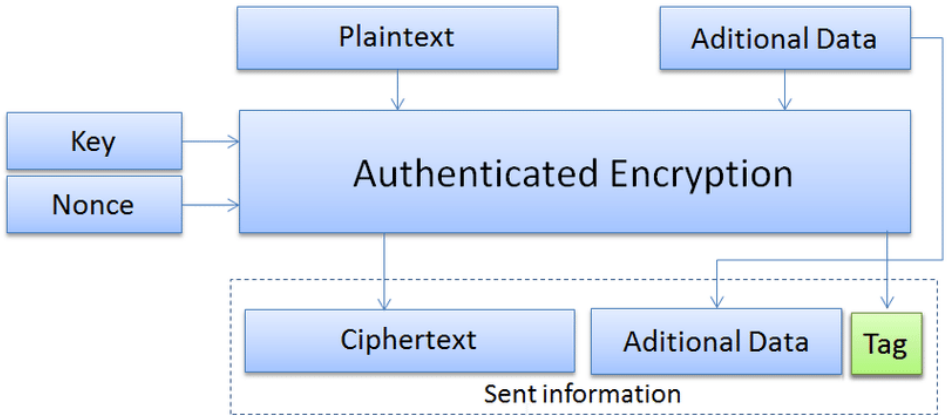
- Sponge 기반 방식의 보안 사용
- Schwaemm의 모든 인스턴스는 192bit보다 큰 nonce 크기 허용함 (Schwaemm128-128 제외)
- 블록암호 Sparx의 설계 전략(long trail strategy)에 의존
- 구성 요소들이 사용 사례에 맞게 조정됨

3. SPARKLE

알고리즘

1) ESCH(Efficient Sponge-based, Cheap Hashing) $['\varepsilon\int]$
   - 룩셈부르크 대학교와 가까운 룩셈부르크 남부의 작은 마을 이름(Esch-sur-Alzette)의 일부
   - 해시 함수
2) Schwaemm(Sponge-based Cipher for Hardened but Weightless Authenticated Encryption on Many Microcontrollers)
   - "sponges"의 룩셈부르크어 (Schwämmen) $['\int v\varepsilon m]$
   - AEAD 방식

     AEAD (Authenticated Encryption with Associated Data)
     - key, nonce(fixed length) & message, associated data(arbitrary size)를 사용함
     - 기밀성과 무결성을 충족해야 함



출처:
https://www.researchgate.net/publication/321137002_Pipeline_Oriented_Implementation_of_NORX_for_ARM_Processors

3. SPARKLE - SPARKLE structure

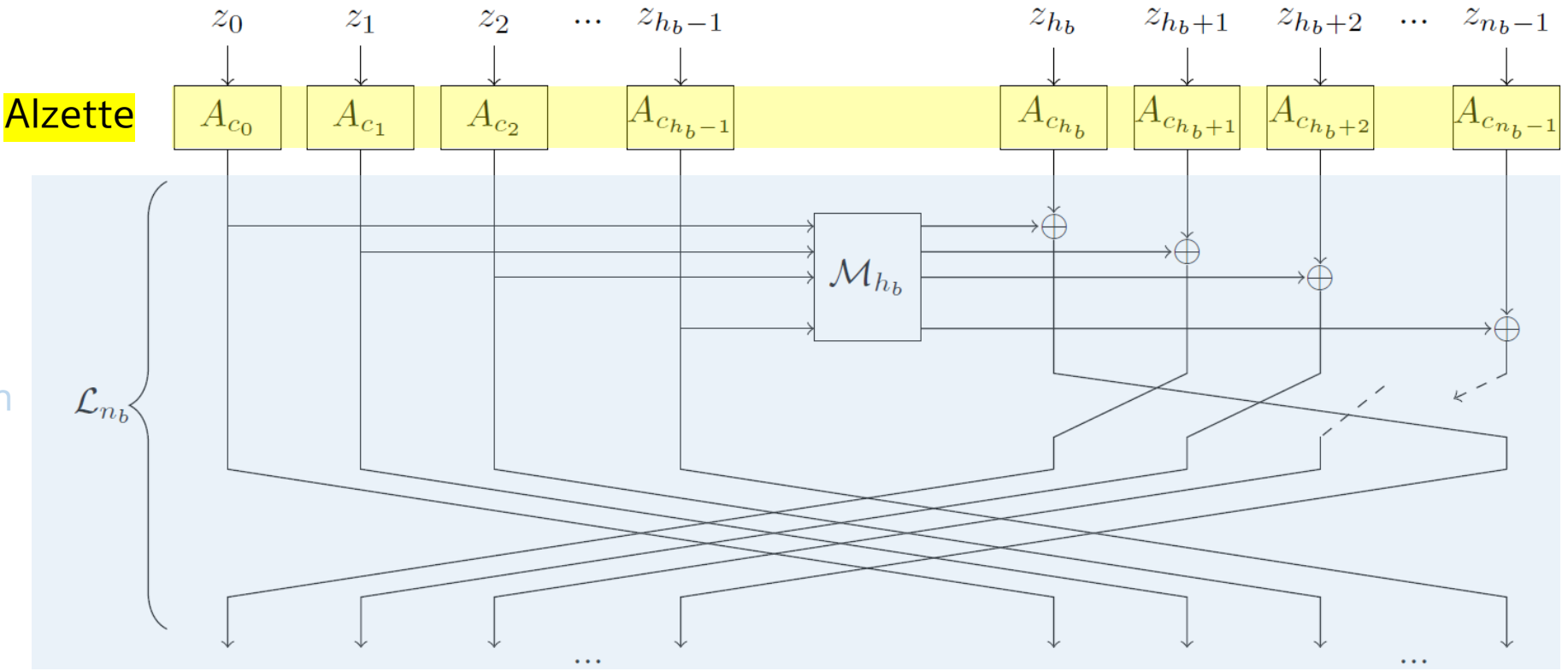


Figure 2.1: The overall structure of a step of SPARKLE. z_i denotes the 64-bit input (x_i, y_i) to the corresponding Alzette instance.

4. SPARKLE256 (1) ARX-box Alzette

- 64bit block 암호.
- 라운드마다 회전량이 다른 4라운드 반복 블록 암호.
- 간단한 구조(≈Feistel)를 가지고 있기 때문에 역계산이 간단함.
- 각 라운드 후에 32bit 상수(key)는 왼쪽 단어에 XOR 됨.

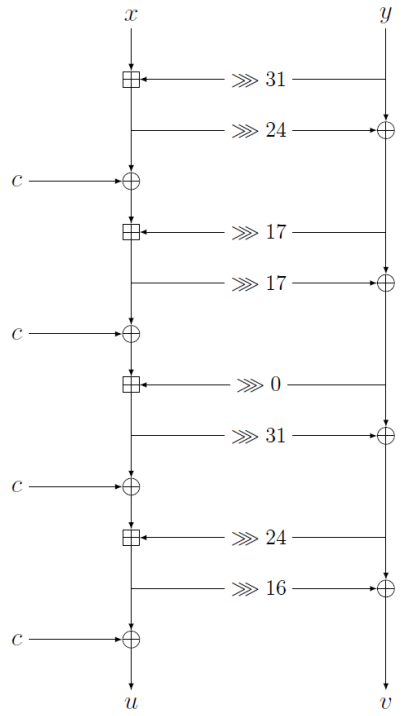


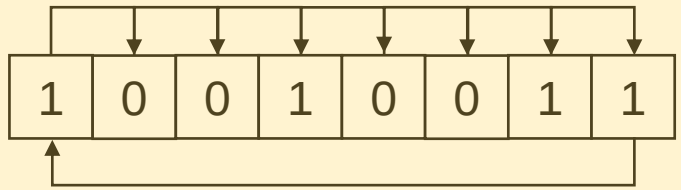
Figure 2.2: The structure of the Alzette instance A_c.

Algorithm 2.4 A_c

Input/Output: $(x, y) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$

$x \leftarrow x + (y \ggg 31)$
$y \leftarrow y \oplus (x \ggg 24)$
$x \leftarrow x \oplus c$
$x \leftarrow x + (y \ggg 17)$
$y \leftarrow y \oplus (x \ggg 17)$
$x \leftarrow x \oplus c$
$x \leftarrow x + (y \ggg 0)$
$y \leftarrow y \oplus (x \ggg 31)$
$x \leftarrow x \oplus c$
$x \leftarrow x + (y \ggg 24)$
$y \leftarrow y \oplus (x \ggg 16)$
$x \leftarrow x \oplus c$
return (x, y)

rotate right




```
#define ROT(x, n) (((x) >> (n)) | ((x) << (32-(n))))
#define ARXBOX(x, y, c)                                  \
  (x) += ROT((y), 31), (y) ^= ROT((x), 24), \
  (x) ^= (c),                                            \
  (x) += ROT((y), 17), (y) ^= ROT((x), 17), \
  (x) ^= (c),                                            \
  (x) += (y),          (y) ^= ROT((x), 31), \
  (x) ^= (c),                                            \
  (x) += ROT((y), 24), (y) ^= ROT((x), 16), \
  (x) ^= (c)
```

4. SPARKLE256 (2) Diffusion Layer

- 확산층(Diffusion layer) $\mathcal{L}_{n_b}$
- permutation에 사용되는 확산층은 총 3가지($\mathcal{L}_4$, $\mathcal{L}_6$, $\mathcal{L}_8$)이다.

$$\mathcal{M}_w\big((x_0, y_0), \ldots, (x_{w-1}, y_{w-1})\big) = \big((u_0, v_0), \ldots, (u_{w-1}, v_{w-1})\big)$$

$$t_y \leftarrow \bigoplus_{i=0}^{w-1} y_i \ , \ t_x \leftarrow \bigoplus_{i=0}^{w-1} x_i \ ,$$

$$u_i \leftarrow x_i \oplus \ell(t_y), \ \forall i \in \{0, \ldots, w-1\} \ ,$$

$$v_i \leftarrow y_i \oplus \ell(t_x), \ \forall i \in \{0, \ldots, w-1\} \ ,$$

$$\ell(x) = (x \lll 16) \oplus (x \& \texttt{0xffff})$$

4. SPARKLE256 (2) Diffusion Layer

Algorithm 2.5 $\mathcal{L}_4$

Input/Output: $\big((x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)\big) \in (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^4$

▷ Feistel round

$(t_x, t_y) \leftarrow (x_0 \oplus x_1, y_0 \oplus y_1)$
$(t_x, t_y) \leftarrow \big((t_x \oplus (t_x \ll 16)) \lll 16,\ (t_y \oplus (t_y \ll 16)) \lll 16\big)$
$(y_2, y_3) \leftarrow (y_2 \oplus y_0 \oplus t_x,\ y_3 \oplus y_1 \oplus t_x)$
$(x_2, x_3) \leftarrow (x_2 \oplus x_0 \oplus t_y,\ x_3 \oplus x_1 \oplus t_y)$

▷ Branch permutation

$(x_0, x_1, x_2, x_3) \leftarrow (x_3, x_2, x_0, x_1)$
$(y_0, y_1, y_2, y_3) \leftarrow (y_3, y_2, y_0, y_1)$
return $\big((x_0, y_0), \ldots, (x_3, y_3)\big)$

```
#define ELL(x) (ROT(((x) ^ ((x) << 16)), 16))
```

```
void linear_layer(SparkleState *state, int brans)
{
  int i, b = brans/2;
  uint32_t *x = state->x, *y = state->y;
  uint32_t tmp;

  // Feistel function (adding to y part)
  tmp = 0;
  for(i = 0; i < b; i++)
    tmp ^= x[i];  t_x
  tmp = ELL(tmp);
  for(i = 0; i < b; i ++)
    y[i+b] ^= (tmp ^ y[i]);

  // Feistel function (adding to x part)
  tmp = 0;
  for(i = 0; i < b; i++)
    tmp ^= y[i];  t_y
  tmp = ELL(tmp);
  for(i = 0; i < b; i ++)
    x[i+b] ^= (tmp ^ x[i]);
```

```
  // Branch swap of the x part
  tmp = x[0];
  for (i = 0; i < b - 1; i++) {
    x[i] = x[i+b+1];   x_0 = x_3
    x[i+b+1] = x[i+1]; x_3 = x_1
  }
  x[b-1] = x[b];  x_1 = x_2
  x[b] = tmp;     x_2 = x_0

  // Branch swap of the y part
  tmp = y[0];
  for (i = 0; i < b - 1; i++) {
    y[i] = y[i+b+1];
    y[i+b+1] = y[i+1];
  }
  y[b-1] = y[b];
  y[b] = tmp;
}
```

감사합니다