AIMer 전자서명 내부의 AIM 대칭 프리미티브 고속 구현

유튜브 주소 : https://youtu.be/7oBBgSao9Yk

AIMer & AIM

제안 기법 및 구현

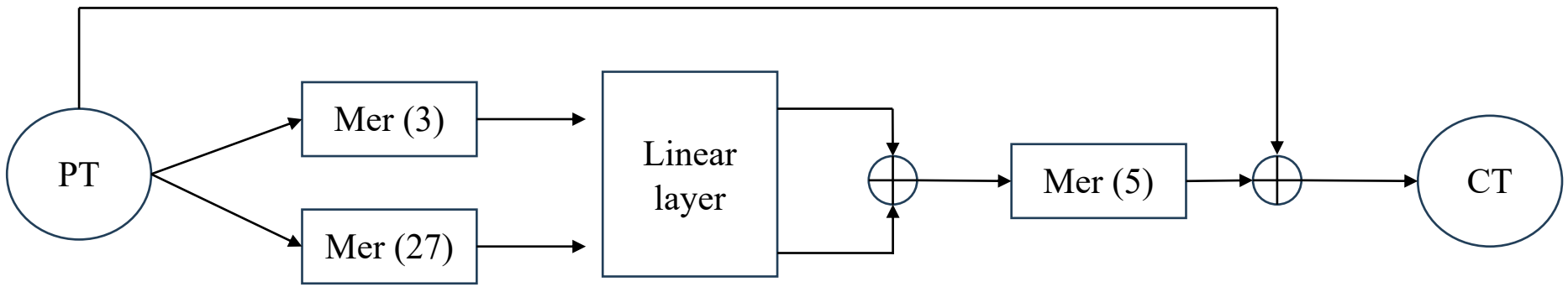
성능 측정

AIMer 전자 서명

- KpqC(Korea Post-Quantum Cryptography) 공모전에 제출된 전자서명 알고리즘
  - 2022년 12월 – 전자서명 알고리즘 9종이 Round 1 통과(AIMer 포함)
- 영지식 기반으로 개발
- 대칭 프리미티브 AIM과 BN++ 증명 시스템을 사용하는 서명 체계
- Binary extension field 상의 파워 매핑 기반 S-box를 사용
  - 해당 기법을 사용해 암호화 프리미티브를 개선
  - S-box는 Mersenne S-box를 기반으로 설계
- Groebner basis 및 XL(eXtended Linearization) 공격을 방어하는데 집중
  - Groebner basis : 가장 잘 알려진 다변수 다항식 계의 풀이 방법
- One-way function 구조를 사용해 참여자 간 데이터를 공유하지 않고 결과를 계산할 수 있는 MPCitH(Multi-Party Computation in the Head)의 호환성 확보
- 위 특징들을 통해 대수적 공격에 대한 내성을 확보

AIM 대칭 프리미티브

- AIMer에서 제안된 대칭 프리미티브
- 대수적 공격에 견딜 수 있도록 설계된 일방향 함수
- AIM 스킴은 3종류로 구성 : AIM-I, AIM-III, AIM-V
  - 본 논문에서는 AIM-I만 다룸
- S-box인 Mer와 선형 레이어로 설계되어있음
  - Mer : Mersenne numbers에 의한 거듭제곱을 계산
  - 선형 레이어 : 이진 행렬 곱셈을 수행



AIM-I 암호화 프로세스

제안 기법

- Mer 연산 최적화 : Combined Mer
  - 입력 평문에 수행하는 Mer 연산을 최적화
- AIM-I 암호화 : 동일한 입력 값(pt)이 복사되어 Mer(3)과 Mer(27)연산을 각각 수행
- Mer(3)과 Mer(27)연산을 각각 수행하는 것이 아닌 Combined Mer 연산 제안
- Combined Mer 연산은 Mer(27) 연산을 한 번만 수행하는 것과 동일한 복잡도 지님
- Mer(3) 연산과 같은 과정이 Mer(27) 연산 내부에서도 수행됨
  - 해당 특징을 활용해 Combined Mer 연산 구현 가능

```
// Mersenne layer
 mersenne_exp_3(input_GF, state[0]);
//printf("check");
mersenne_exp_27(input_GF, state[1],
    state[0]);
```

Mer(3) 함수 호출 코드

```
void mersenne_exp_3(const GF in, GF out)
{
  GF t1 = {0,};

  // t1 = a ^ (2^2 - 1)
  GF_sqr(in, t1);
  GF_mul(t1, in, t1);

  // out = a ^ (2^3 - 1)
  GF_sqr(t1, t1);
  GF_mul(t1, in, out);

}
```

Mer(3) 함수 동작 코드

제안 기법

- 왼쪽은 기존의 Mer 연산 코드
- 오른쪽은 제안된 기법 구현 코드
  - Mer(3) 연산값->state[0]에 저장
  - State[0]-> Mer(27)의 파라미터
  - Mer(27)내부 연산-> Mer(27)에서도 수행
  - 기존 코드 중간 연산값 저장 -> t2
  - 제안 기법은 state[0](out2)에 저장하여 구현

```
void mersenne_exp_3(const GF in, GF out)
{
  GF t1 = {0,};

  // t1 = a ^ (2^2 - 1)
  GF_sqr(in, t1);
  GF_mul(t1, in, t1);

  // out = a ^ (2^3 - 1)
  GF_sqr(t1, t1);
  GF_mul(t1, in, out);

}
```

Mer(3) 함수 동작 코드

```
// Mersenne layer
 mersenne_exp_3(input_GF, state[0]);
//printf("check");
mersenne_exp_27(input_GF, state[1],
    state[0]);
```

```
void mersenne_exp_27(const GF in, GF out,
    GF out2)
{
  int i;
  GF t1 = {0,};
  GF t2 = {0,};
  GF t3 = {0,};

  // t1 = a ^ (2^2 - 1)
  GF_sqr(in, t1);
  GF_mul(t1, in, t1);

  // t2 = a ^ (2^3 - 1)
  GF_sqr(t1, t1);
  GF_mul(t1, in, t2);

  //out2 = t2;
  // t3 = a ^ (2^6 - 1)
  GF_sqr(t2, t1);
  GF_sqr(t1, t1);
  GF_sqr(t1, t1);
  GF_mul(t1, t2, t3);

  // t3 = a ^ (2^12 - 1)
  GF_sqr(t3, t1);
  for (i = 1; i < 6; i++)
  {
    GF_sqr(t1, t1);
  }
  GF_mul(t1, t3, t3);

  // t3 = a ^ (2^24 - 1)
  GF_sqr(t3, t1);
  for (i = 1; i < 12; i++)
  {
    GF_sqr(t1, t1);
  }
  GF_mul(t1, t3, t3);
```

```
// Mersenne layer
// mersenne_exp_3(input_GF, state[0]);
//printf("check");
mersenne_exp_27(input_GF, state[1],
    state[0]);
```

```
int i;
  GF t1 = {0,};
  //GF t2 = {0,};
  GF t3 = {0,};

  // t1 = a ^ (2^2 - 1)
  GF_sqr(in, t1);
  GF_mul(t1, in, t1);

  // t2 = a ^ (2^3 - 1)
  GF_sqr(t1, t1);
  GF_mul(t1, in, out2);

  // t3 = a ^ (2^6 - 1)
  GF_sqr(out2, t1);
  GF_sqr(t1, t1);
  GF_sqr(t1, t1);
  GF_mul(t1, out2, t3);

  // t3 = a ^ (2^12 - 1)
  GF_sqr(t3, t1);
  for (i = 1; i < 6; i++)
  {
    GF_sqr(t1, t1);
  }
  GF_mul(t1, t3, t3);

  // t3 = a ^ (2^24 - 1)
  GF_sqr(t3, t1);
  for (i = 1; i < 12; i++)
  {
    GF_sqr(t1, t1);
  }
  GF_mul(t1, t3, t3);
```

제안 기법

- 선형 레이어 연산 간소화

- AIM의 선형 레이어 연산은 총 4개의 행렬-벡터 곱셈 수행
  - 초기 벡터의 해시값(SHAKE-128)을 이용해 128*128 바이너리 행렬 생성
  - 생성된 행렬은 이니셜 벡터에만 영향을 받고 평문에 영향을 받지 않음
  - 즉, 암호화 수행 시 마다 행렬을 생성할 필요가 없음

- 행렬 4개의 값을 Look-up table을 이용해 구현
  - 제안된 Look-up table은 64bit의 크기를 가지는 2개의 배열이 128개로 이루어진 2차원 배열의 형태

```
uint64_t state0_lower_tr[128][2] = {
    {0x1, 0x0}, {0x2, 0x0}, {0x7, 0x0},
        {0x8, 0x0}, {0x1b, 0x0}, {0x23,
        0x0}, {0x65, 0x0}, {0xbe, 0x0},
        {0x1ea, 0x0}, {0x363, 0x0}, {0x791,
        0x0}, {0xfe9, 0x0}, {0x1495, 0x0},
        {0x31a9, 0x0}, {0x46df, 0x0},
        {0xbc1c, 0x0}, {0x183ea, 0x0},
        {0x383c3, 0x0}, {0x5c9eb, 0x0},
        {0xff44d, 0x0}, {0x16dc2e, 0x0},
        {0x3d48cd, 0x0}, {0x756621, 0x0},
        {0xeb8633, 0x0}, {0x18357a2, 0x0},
        {0x21e4d45, 0x0}, {0x6191ca7, 0x0},
        {0xda9edfd, 0x0}, {0x10972df1, 0x0},
        {0x34ab3a98, 0x0}, {0x585678db,
        0x0}, {0xee8de9b8, 0x0},
        {0x1b6ff58c3, 0x0}, {0x2b955f709,
        0x0}, {0x7a53d18b1, 0x0},
        {0xd98dc3559, 0x0}, {0x1eee34a12c,
        0x0}, {0x3e305db00b, 0x0},
        {0x61d058eeca, 0x0}, {0x9815c483d0,
        0x0}, {0x1cf953e85f6, 0x0},
        {0x328863614ee, 0x0},
        {0x6ba4a07cae8, 0x0},
        {0xcdc61bdfc2a, 0x0},
        {0x14f6375f3091, 0x0},
        {0x3e3ea28c7856, 0x0},
        {0x5af63fe90759, 0x0},
        {0xf26d1388fc2e, 0x0},
        {0x14611e7d8a792, 0x0},
        {0x28182058d910c, 0x0},
        {0x6eb3c6c2e834c, 0x0},
        {0x8462eaf1716d3, 0x0},
        {0x11b84d2bdd622b, 0x0},
        {0x36ac12a1056c36, 0x0},
        {0x78650fec38cdd3, 0x0},
        {0xf3e0c7b9a5f8ea, 0x0},
        {0x151f47315add797, 0x0},
        {0x249813356bff641, 0x0},
```

Look-up table 구현 코드

제안 기법

- AIM의 선형 레이어는 affine layer와 feed-forward 두 가지 선형 구성 요소로 이루어짐
- Affine layer : matrix A로의 곱셈과 Vector B에 대한 덧셈으로 이루어짐
  - Matrix A: n*ln 크기의 랜덤 이진 행렬
  - Vector B: 이니셜 벡터에 영향을 받는 랜덤 상수
- 제안된 선형 레이어 간소화 기법에선 affine layer를 생성하지 않음
  - Matrix A와 Vector B가 생성되지 않음
- 그러나 Vector B는 Mer(5) 연산 이전에 state[0] 배열과 덧셈 연산 수행
- Vector B 값 또한 Look-up table 형태로 값을 명시해줘야 할 필요 있음

```
vector_B[1] = 0x9347b8e12b0971a1;
vector_B[0] = 0xcaf99a30fa2d6733;

GF_add(state[0], state[1], state[0]);
GF_add(state[0], vector_B, state[0]);
```

Look-up table 구현 코드

성능 측정

- 성능 측정 환경 : Apple M2 프로세서(최대 3.49MHz 속도)
  - 각 알고리즘을 1,000,000회 반복하여 측정된 시간들의 평균 값 사용
- 선형 레이어 연산 간소화를 적용한 구현물의 성능이 큰 차이를 보임
- 두 기법을 모두 적용한 구현물은 96.9%의 성능 향상
- Mer 연산 최적화 기법만 적용한 구현물의 성능 향상 정도가 비교적 낮게 측정됨
  - 선형 레이어 연산이 소모하는 비용이 너무 커 상대적으로 낮은 향상률이 측정된 것으로 사료됨
  - 선형 레이어 간소화 기법만 적용한 구현물과 두 기법 모두 적용한 구현물을 비교한 결과 7.42%의 성능 향상이 있었음을 확인
  - 이를 통해 Mer 연산 최적화 기법 또한 비용 감소에 있어 유의미한 결과를 도출했음을 확인

	Reference	Combined Mer	Linear Layer	Combined Mer + Linear Layer
Ms	38481.699	38171.393	1268.132	1180.554
Diff.	0	0.81	96.7	96.9

성능 측정 결과

Q & A