SNOVA 코드 분석

유튜브 주소 : https://youtu.be/c_9RDWQg4Sc

SNOVA

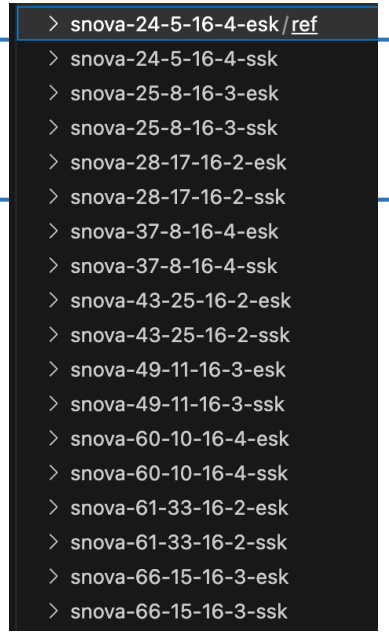
가산기

곱셈기

SNOVA

- SNOVA: Simple Noncommutative unbalanced Oil and Vinegar scheme with randomness Alignment
- NIST PQC Additional Signature Round 1 제출된 알고리즘
  - 다변수 기반 전자서명(1라운드 10개 제출)
  - UOV 스킴 기반(UOV 포함 6개 제출)
- UOV: Unbalanced Oil and Vinegar
  - 1999년 공개된 다변수 기반 알고리즘
  - 공개키와 개인키를 이차 다변수 다항식으로 구성
    - ‘비네거’ 변수와 ‘오일’ 변수를 사용
    - 개인키를 알아야만 ‘비네거’ 변수로부터 ‘오일’ 변수를 계산 가능한 매커니즘
  - 장점
    - 구현이 간결하며 서명 생성 및 검증 속도가 상대적으로 빠름
  - 단점
    - 상대적으로 서명의 크기가 크며 공개키의 크기가 매우 큼

SNOVA



- SNOVA: noncommutative-ring 기반 알고리즘
  - 공개키가 큰 UOV 알고리즘의 단점을 상쇄하여 설계
    - UOV 기반 알고리즘 중 가장 작은 공개키 크기를 지님
  - 다양한 보안 수준에 대해 서로 다른 파라미터 옵션을 제공
    - esk 9개, ssk 9개로 총 18개의 파라미터 옵션 제공
    - 리소스가 제한된 환경 및 높은 보안이 필요한 애플리케이션에 적용이 용이하도록 설계

Scheme	비밀키(esk)(bytes)	공개키(bytes)	서명(bytes)
UOV-I	348,704	412,160	96
SNOVA-I	**60,056**	**9,842**	**90**
PROV-I	203,752	68,326	160
QR_UOV-I	-	20,657	331
TUOV-I	350,272	412,160	80
VOX-I	35,056	9,104(cpk)	102

SNOVA

- 주요 연산: 가산기, 곱셈기
  - 유한체($GF2^4$)상에서의 연산 수행

```c
/**
 * Adding GF16 Matrices. c = a + b
 */
static inline void gf16m_add(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_add(get_gf16m(a, i, j), get_gf16m(b, i, j)));
        }
    }
}
```

```c
/**
 * Multiplying GF16 Matrices. c = a * b
 */
static inline void gf16m_mul(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_mul(get_gf16m(a, i, 0), get_gf16m(b, 0, j)));
            for (int k = 1; k < rank; ++k) {
                set_gf16m(c, i, j,
                          gf16_get_add(get_gf16m(c, i, j),
                                       gf16_get_mul(get_gf16m(a, i, k),
                                                    get_gf16m(b, k, j))));
            }
        }
    }
}
```

- 유한체 연산을 위한 룩업 테이블 생성
  - 곱셈 등의 효율적인 연산을 위함
    - 반복 계산 감소 및 리소스 절약 효과

```c
void init_gf16_tables() {
    uint8_t F_star[15] = {1, 2,  4, 8,  3,  6,  12, 11,
                          5, 10, 7, 14, 15, 13, 9};  // Z2[x]/(x^4+x+1)
    for (int i = 0; i < 16; ++i) {
        mt(0, i) = mt(i, 0) = 0;
    }

    for (int i = 0; i < 15; ++i)
        for (int j = 0; j < 15; ++j)
            mt(F_star[i], F_star[j]) = F_star[(i + j) % 15];
    {
        int g = F_star[1], g_inv = F_star[14], gn = 1, gn_inv = 1;
        inv4b[0] = 0;
        inv4b[1] = 1;
        for (int index = 0; index < 14; index++)
            inv4b[(gn = mt(gn, g))] = (gn_inv = mt(gn_inv, g_inv));
    }
}
```

SNOVA 가산기

- 128bit matrix a XOR b -> c
  - 8bit 4*4 행렬 간의 덧셈
    - 유한체 상에서의 덧셈은 비트간 XOR 연산으로 구현 가능
    - 캐리연산 등 불필요
  - 각 위치(i, j)에 해당하는 행렬 a, b의 요소를 가져와 덧셈
    - 반복문의 i, j는 행렬의 인덱스를 설정
      - 인덱스는 get_gf16m 함수 내에서의 shift, xor 연산으로 설정
    - 덧셈은 gf16_get_add 함수를 통해 실행

```c
/**
 * Adding GF16 Matrices. c = a + b
 */
static inline void gf16m_add(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_add(get_gf16m(a, i, j), get_gf16m(b, i, j)));
        }
    }
}
```

```c
#define mt(p, q) mt4b[((p) << 4) ^ (q)]
#define inv(gf16) inv4b[(gf16)]
#define gf16_get_add(a, b) ((a) ^ (b))
#define gf16_get_mul(a, b) (mt((a), (b)))

static uint8_t mt4b[256] = {0};
static uint8_t inv4b[16] = {0};

typedef uint8_t gf16_t;
```

$$A + B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ a_{m1} & a_{m2} & \cdots & a_{mf} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2n} \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{in} \\ b_{m1} & b_{m2} & \cdots & b_{mf} & \cdots & b_{mn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{1j}+b_{1j} & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{2j}+b_{2j} & a_{2n}+b_{2n} \\ a_{i1}+b_{i1} & a_{i2}+b_{i2} & a_{ij}+b_{ij} & a_{in}+b_{in} \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & a_{mj}+b_{mj} & a_{mn}+b_{mn} \end{bmatrix} = \begin{bmatrix} a_{ij} + b_{ij} \end{bmatrix}$$

```c
#elif rank == 4
#define get_gf16m(gf16m, x, y) (gf16m[(((x) << 2) ^ (y))])
#define set_gf16m(gf16m, x, y, value) (gf16m[(((x) << 2) ^ (y))] = value)
#endif

typedef gf16_t gf16m_t[sq_rank];
```

sq_rank: rank*rank(16)

SNOVA 가산기 binary field

- a[0~15] XOR b[0~15] => c에 Set
  - 즉, 단순한 a XOR b의 구조

```
#elif rank == 4
#define get_gf16m(gf16m, x, y) (gf16m[(((x) << 2) ^ (y))])
#define set_gf16m(gf16m, x, y, value) (gf16m[(((x) << 2) ^ (y))] = value)
#endif

typedef gf16_t gf16m_t[sq_rank];
```

(i, j)	0 0	0 1	0 2	0 3	1 0	1 1	1 2	1 3	2 0	2 1	2 2	2 3	3 0	3 1	3 2	3 3
binary	0000 0000	0000 0001	0000 0010	0000 0011	0001 0000	0001 0001	0001 0010	0001 0011	0010 0000	0010 0001	0010 0010	0010 0011	0011 0000	0011 0001	0011 0010	0011 0011
i left shift 2	0000 0000	0000 0001	0000 0010	0000 0011	0100 0000	0100 0001	0100 0010	0100 0011	1000 0000	1000 0001	1000 0010	1000 0011	1100 0000	1100 0001	1100 0010	1100 0011
I XOR j	0b0000	0b0001	0b0010	0b0011	0b0100	0b0101	0b0110	0b0111	0b1000	0b1001	0b1010	0b1011	0b1100	0b1101	0b1110	0b1111
Result	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
int add_count = 0;
static inline void gf16m_add(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            if (add_count < 16) {
                uint8_t value_a = get_gf16m(a, i, j);
                uint8_t value_b = get_gf16m(b, i, j);
                uint8_t result = gf16_get_add(value_a, value_b);
                printf("(%d,%d) A:%X B:%X C:%X\n",
                       i,j,value_a, value_b, result);
                set_gf16m(c, i, j, result);
                add_count++;
            } else {
                set_gf16m(c, i, j, gf16_get_add(get_gf16m(a, i, j), get_gf16m(b, i, j)));
```

```
(0,0) A:F B:A C:5
(0,1) A:0 B:2 C:2
(0,2) A:0 B:E C:E
(0,3) A:0 B:9 C:9
(1,0) A:0 B:2 C:2
(1,1) A:F B:E C:1
(1,2) A:0 B:9 C:9
(1,3) A:0 B:5 C:5
(2,0) A:0 B:E C:E
(2,1) A:0 B:9 C:9
(2,2) A:F B:5 C:A
(2,3) A:0 B:7 C:7
(3,0) A:0 B:9 C:9
(3,1) A:0 B:5 C:5
(3,2) A:0 B:7 C:7
(3,3) A:F B:B C:4
```

```
LD1.16b {v0}, [x0]
LD1.16b {v1}, [x1]

EOR.16b v1, v0, v1

ST1.16b {v1}, [x2]

RET
```

ARMv8 구현 예시

SNOVA 곱셈기

- GF16 상에서의 a * b 연산 수행
  - 룩업 테이블을 활용하여 연산 구현
    - 16 * 16의 모든 경우의 수를 미리 계산한 테이블
  - 인덱스를 계산해 값을 조회하는 방식의 구현

```c
#define mt(p, q) mt4b[((p) << 4) ^ (q)]
#define inv(gf16) inv4b[(gf16)]
#define gf16_get_add(a, b) ((a) ^ (b))
#define gf16_get_mul(a, b) (mt((a), (b)))

static uint8_t mt4b[256] = {0};
static uint8_t inv4b[16] = {0};

typedef uint8_t gf16_t;
```

```c
/**
 * Multiplying GF16 Matrices. c = a * b
 */
static inline void gf16m_mul(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_mul(get_gf16m(a, i, 0), get_gf16m(b, 0, j)));
            for (int k = 1; k < rank; ++k) {
                set_gf16m(c, i, j,
                          gf16_get_add(get_gf16m(c, i, j),
                                       gf16_get_mul(get_gf16m(a, i, k),
                                                    get_gf16m(b, k, j))));
            }
        }
    }
}
```

```c
void init_gf16_tables() {
    uint8_t F_star[15] = {1, 2,  4, 8,  3,  6,  12, 11,
                          5, 10, 7, 14, 15, 13, 9};  // Z2[x]/(x^4+x+1)
    for (int i = 0; i < 16; ++i) {
        mt(0, i) = mt(i, 0) = 0;
    }

    for (int i = 0; i < 15; ++i)
        for (int j = 0; j < 15; ++j)
            mt(F_star[i], F_star[j]) = F_star[(i + j) % 15];
    {
        int g = F_star[1], g_inv = F_star[14], gn = 1, gn_inv = 1;
        inv4b[0] = 0;
        inv4b[1] = 1;
        for (int index = 0; index < 14; index++)
            inv4b[(gn = mt(gn, g))] = (gn_inv = mt(gn_inv, g_inv));
    }
}
```

→ 기약다항식(x^4+x+1)에 의해 생성된 GF16 원소

→ 테이블 초기화 과정

→ 곱셈 테이블 생성 과정
각 원소의 곱셈 결과는 F_star 배열에서 두 인덱스의 합을 15로 모듈러 연산하여 얻은 값

→ 역원 생성 과정

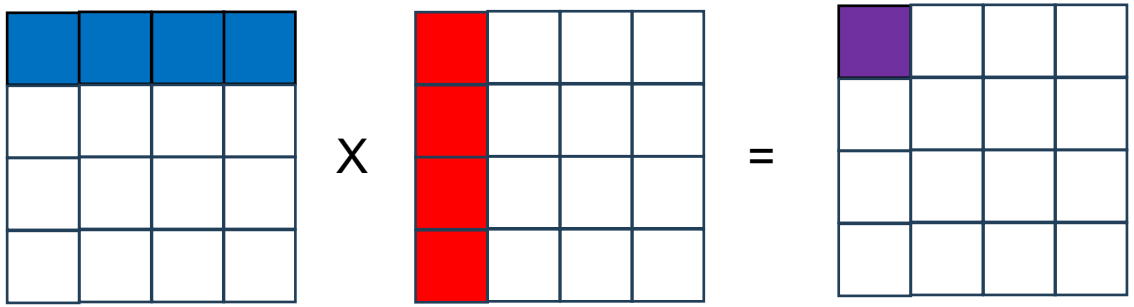
SNOVA 곱셈기

```
#define mt(p, q) mt4b[((p) << 4) ^ (q)]
#define inv(gf16) inv4b[(gf16)]
#define gf16_get_add(a, b) ((a) ^ (b))
#define gf16_get_mul(a, b) (mt((a), (b)))

static uint8_t mt4b[256] = {0};
static uint8_t inv4b[16] = {0};

typedef uint8_t gf16_t;
```

- i 루프: a행렬의 요소(행)를 반복
- j 루프: b행렬의 요소(열)를 반복



행렬 곱셈

```
/**
 * Multiplying GF16 Matrices. c = a * b
 */
static inline void gf16m_mul(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_mul(get_gf16m(a, i, 0), get_gf16m(b, 0, j)));
            for (int k = 1; k < rank; ++k) {
                set_gf16m(c, i, j,
                          gf16_get_add(get_gf16m(c, i, j),
                                       gf16_get_mul(get_gf16m(a, i, k),
                                                    get_gf16m(b, k, j))));
            }
        }
    }
}
```

각 행렬의 첫 번째 요소간의 곱셈
e.g. a[0][0] * b[0][0]

각 행렬의 2,3,4 번째 요소간의 곱셈
각 곱셈 결과를 누산하여 결과 도출

SNOVA 곱셈기 binary field

- GF16 상에서의 곱셈 진행
- 기약다항식 사용
  - $x^4 + x + 1 = 0$

```c
/**
 * Multiplying GF16 Matrices. c = a * b
 */
static inline void gf16m_mul(gf16m_t a, gf16m_t b, gf16m_t c) {
    for (int i = 0; i < rank; ++i) {
        for (int j = 0; j < rank; ++j) {
            set_gf16m(c, i, j,
                      gf16_get_mul(get_gf16m(a, i, 0), get_gf16m(b, 0, j)));
            for (int k = 1; k < rank; ++k) {
                set_gf16m(c, i, j,
                          gf16_get_add(get_gf16m(c, i, j),
                                       gf16_get_mul(get_gf16m(a, i, k),
                                                    get_gf16m(b, k, j))));
            }
        }
    }
}
```

```
1st mul (0,0): 8 * 8 = C
2 mul (0,0): C + (7 * 7 = 6) = A
3 mul (0,0): A + (6 * 6 = 7) = D
4 mul (0,0): D + (5 * 5 = 2) = F
1st mul (0,1): 8 * 7 = D
2 mul (0,1): D + (7 * 6 = 1) = C
3 mul (0,1): C + (6 * 5 = D) = 1
4 mul (0,1): 1 + (5 * 4 = 7) = 6
1st mul (0,2): 8 * 6 = 5
2 mul (0,2): 5 + (7 * 5 = 8) = D
3 mul (0,2): D + (6 * 4 = B) = 6
4 mul (0,2): 6 + (5 * 3 = F) = 9
1st mul (0,3): 8 * 5 = E
2 mul (0,3): E + (7 * 4 = F) = 1
3 mul (0,3): 1 + (6 * 3 = A) = B
4 mul (0,3): B + (5 * 2 = A) = 1
1st mul (1,0): 7 * 8 = D
2 mul (1,0): D + (6 * 7 = 1) = C
3 mul (1,0): C + (5 * 6 = D) = 1
4 mul (1,0): 1 + (4 * 5 = 7) = 6
1st mul (1,1): 7 * 7 = 6
2 mul (1,1): 6 + (6 * 6 = 7) = 1
3 mul (1,1): 1 + (5 * 5 = 2) = 3
4 mul (1,1): 3 + (4 * 4 = 3) = 0
1st mul (1,2): 7 * 6 = 1
2 mul (1,2): 1 + (6 * 5 = D) = C
3 mul (1,2): C + (5 * 4 = 7) = B
4 mul (1,2): B + (4 * 3 = C) = 7
1st mul (1,3): 7 * 5 = 8
2 mul (1,3): 8 + (6 * 4 = B) = 3
3 mul (1,3): 3 + (5 * 3 = F) = C
4 mul (1,3): C + (4 * 2 = 8) = 4
1st mul (2,0): 6 * 8 = 5
2 mul (2,0): 5 + (5 * 7 = 8) = D
3 mul (2,0): D + (4 * 6 = B) = 6
4 mul (2,0): 6 + (3 * 5 = F) = 9
1st mul (2,1): 6 * 7 = 1
2 mul (2,1): 1 + (5 * 6 = D) = C
3 mul (2,1): C + (4 * 5 = 7) = B
```

- 연산 원리 예시
  - 8 * 8 = C
    - 8 -> 1000 -> x^3
    - 8 * 8 -> $x^3 * x^3 = x^6$
    - x^6 을 기약다항식으로 나눈 나머지 계산
      - $x^4 + x + 1 = 0$ -> $x^4 = -x - 1$ -> $x^4 = x + 1$으로 치환 가능
    - $x^6 = (x^2 * x^4) = (x^2 * (x + 1)) = x^3 + x^2$ -> 1100 -> C
  - 7 * 6 = 1
    - 7 -> 0111 -> $x^2 + x + 1$, 6 -> 0110 -> $x^2 + x$
    - 7 * 6 -> $(x^2 + x + 1) * (x^2 + x) = x^4 + 2x^3 + 2x^2 + x = x^4 + x$ (2진수 연산이므로 $2x^3$, $2x^2$은 0으로 취급 가능)
    - $x^4 + x$를 기약다항식으로 나눈 나머지 계산
    - $x^4 + x = (x + 1) + x = 1$

Q & A