NIST MAXDEPTH + Berlekamp Massey Decoding

장경배

https://youtu.be/oYXWWI02-VY

HANSUNG UNIVERSITY
CryptoCraft LAB

NIST MAXDEPTH

- **NIST는 잠재적 양자 공격에 대해 running time, circuit depth 제한을 정의하고 있음**
  → **MAXDEPTH**
  - 매우 긴, Serial한 양자 연산에 있어서의 어려움에 기반

- MAXDEPTH
  - $2^{40} \leq 2^{64} \leq 2^{96}$, 세 가지 기준으로 나뉨
    - $\mathbf{2^{40}}$: 현재 구상 중인 양자 컴퓨팅 아키텍처가 **1년 동안** Serial하게 수행할 것으로 예상되는 대략적인 게이트 수
    - $\mathbf{2^{64}}$: 현재의 고전적인 컴퓨팅 아키텍처가 **10년 동안** Serial로 수행할 수 있는 대략적인 게이트 수
    - $\mathbf{2^{96}}$: 광 전파 시간의 속도로 atomic-scale 큐비트가 **천년 동안** 수행할 수 있는 게이트의 대략적인 수

NIST MAXDEPTH

- **지금 제시되고 있는 양자 분석들은 대부분 logical level이며, 실제로 수행되기 어려움**
  - 양자 공격에 대한 어려움에 있어, MAXDEPTH 까지 고려해야함

- **양자 공격에 있어 MAXDEPTH (2^{96})를 초과한다면?**
  - Grover 알고리즘의 경우, Parallel한 접근이 요구됨

1. **Outer-Parallelization** → 비교적 간단한 방법
   - Grover 반복을 끝까지 수행하지 않고, **도중에 중단 후 솔루션을 관측**
   - 확률을 충분히 높이지 않았기 때문에 (Ex, 1/4) 솔루션이 아닌 경우, 솔루션을 찾을 때 까지 Grover search를 수행
     - **Parallel하게 Grover search를 수행** → 여러 개의 후보 솔루션에서, 진짜 솔루션을 찾아낼 수 있음

Panjin Kim, Kyung Chul Jeong, and Daewan Han, "Time-Space Complexity of Quantum Search Algorithms in Symmetric Cryptanalysis", Quantum Information Processing, 2018.

NIST MAXDEPTH

2. Inner-Parallelization

- Search space(N)를 축소시킴으로써 iteration 수를 감소시킴
  - N의 search space를 **S개의 Subset으로 쪼갬**
    - Optimal iteration number: $\frac{\pi}{4}\sqrt{N} \rightarrow \frac{\pi}{4}\sqrt{N/S}$
  - 회로 Depth를 줄일 수 있음

- Ex) 64-bit key 중 8-bit key에 대해서만 Diffusion operator 적용 ($S = 8$)
  - 8개의 Grover search(S)를 Parallel하게 동작, iteration number: $\frac{\pi}{4}\sqrt{2^n/8}$
    - 8개의 Grover search 중 한 개는 올바른 솔루션을 관측함 (높은 확률로)
    - **$\sqrt{S}$ 만큼 Depth를 줄일 수 있음**

Panjin Kim, Kyung Chul Jeong, and Daewan Han, “Time-Space Complexity of Quantum Search Algorithms in Symmetric Cryptanalysis”, Quantum Information Processing, 2018.

NIST MAXDEPTH

- **Grover search에 대한 MAXDEPTH 제한은 Parallelization을 통해 극복할 수 있음**
  - Depth를 줄일 순 있지만, 양자 게이트의 수 (큐비트 또한)가 늘어남

- NIST의 양자 후 보안 레벨 기준
  - AES에 대한 공격 비용 (Grassl et al's AES)과 MAXDEPTH를 상호 비교

AES 128	2^{170}/MAXDEPTH quantum gates or 2^{143} classical gates
SHA3-256	2^{146} classical gates
AES 192	2^{233}/MAXDEPTH quantum gates or 2^{207} classical gates
SHA3-384	2^{210} classical gates
AES 256	2^{298}/MAXDEPTH quantum gates or 2^{272} classical gates
SHA3-512	2^{274} classical gates

NIST MAXDEPTH

- **AES-128의 경우, Parallelization 없이 MAXDEPTH (2^{96})안에 들어옴**

Table 8: Quantum resources required for Grover's search for AES (this work).

AES			r	#qubits (M) ❂	Total gates (Decomposed) ✱	Full depth ✱	Cost (Complexity) ✱ × ✱
128	✩	✿	1	3,937	$1.597 \cdot 2^{82}$	$1.046 \cdot 2^{75}$	$1.671 \cdot 2^{157}$
	◎			6,369	$1.527 \cdot 2^{82}$	$1.501 \cdot 2^{74}$	$1.146 \cdot 2^{157}$
	◈			7,521	$1.599 \cdot 2^{82}$	$1.226 \cdot 2^{74}$	$1.960 \cdot 2^{156}$
	✩	❀		5,177	$1.664 \cdot 2^{83}$	$1.002 \cdot 2^{75}$	$1.668 \cdot 2^{158}$
	◎			8,849	$1.586 \cdot 2^{83}$	$1.454 \cdot 2^{74}$	$1.153 \cdot 2^{158}$
	◈			10,001	$1.619 \cdot 2^{83}$	$1.18 \cdot 2^{74}$	$1.909 \cdot 2^{157}$
192	✩	✿	2	7,841	$1.683 \cdot 2^{115}$	$1.248 \cdot 2^{107}$	$1.05 \cdot 2^{223}$
	◎			12,225	$1.619 \cdot 2^{115}$	$1.801 \cdot 2^{106}$	$1.457 \cdot 2^{222}$
	◈			15,041	$1.706 \cdot 2^{115}$	$1.465 \cdot 2^{106}$	$1.25 \cdot 2^{222}$
	✩	❀		10,073	$1.753 \cdot 2^{116}$	$1.195 \cdot 2^{107}$	$1.048 \cdot 2^{224}$
	◎			16,689	$1.682 \cdot 2^{116}$	$1.746 \cdot 2^{106}$	$1.469 \cdot 2^{223}$
	◈			19,505	$1.722 \cdot 2^{116}$	$1.41 \cdot 2^{106}$	$1.214 \cdot 2^{223}$
256	✩	✿	2	8,417	$1.012 \cdot 2^{148}$	$1.463 \cdot 2^{139}$	$1.481 \cdot 2^{287}$
	◎			12,737	$1.955 \cdot 2^{147}$	$1.056 \cdot 2^{139}$	$1.032 \cdot 2^{287}$
	◈			16,065	$1.03 \cdot 2^{148}$	$1.715 \cdot 2^{138}$	$1.766 \cdot 2^{286}$
	✩	❀	2	10,649	$1.055 \cdot 2^{149}$	$1.401 \cdot 2^{139}$	$1.477 \cdot 2^{288}$
	◎			17,201	$1.018 \cdot 2^{149}$	$1.024 \cdot 2^{139}$	$1.042 \cdot 2^{288}$
	◈			20,529	$1.041 \cdot 2^{149}$	$1.65 \cdot 2^{138}$	$1.719 \cdot 2^{287}$

✩: Regular version.
◎: Shallow version.
◈: Shallow/low depth version.
✿: Using S-box with Toffoli depth 4.
❀: Using S-box with Toffoli depth 3.

AES 128	2^{170}/MAXDEPTH quantum gates or 2^{143} classical gates
SHA3-256	2^{146} classical gates
AES 192	2^{233}/MAXDEPTH quantum gates or 2^{207} classical gates
SHA3-384	2^{210} classical gates
AES 256	2^{298}/MAXDEPTH quantum gates or 2^{272} classical gates
SHA3-512	2^{274} classical gates

- $2^{170} / 2^{75} = 2^{95} > 2^{82}$
- **Depth가 MAXDEPTH를 초과할 때는 Parallelization에 대한 비용까지 계산**해야 하지만 **어쨌든 Cost * Depth 비용이 관건**

Classic McEliece : Decryption (Decoding)

```c
    for (i = 0; i < SYND_BYTES; i++)       r[i] = c[i]; //Set C0
    for (i = SYND_BYTES; i < SYS_N/8; i++) r[i] = 0;    //Padding 0

    for (i = 0; i < SYS_T; i++) { g[i] = load2(sk); g[i] &= GFMASK; sk += 2; }
    g[ SYS_T ] = 1; //load g(z), and move pointer sk = {g(z), alpha,

    support_gen(L, sk); //Load alpha with private key(the next value after creating g).

    synd(s, g, L, r); // Generate parity matrix H using private keys g(x),L and multiply by v=(C0,0 padding)

    bm(locator, s); //Find error locator polynomial (x-o1)(x-o2)(x-o3)..(x-ot)

    root(images, locator, L);   //input: polynomial f and list of field elements L */
                                //output: out = [ f(a) for a in L ] */
                                //Check field elements(alpha) make error locator polynomial to zero

    //
    for (i = 0; i < SYS_N/8; i++)
        e[i] = 0;

    for (i = 0; i < SYS_N; i++)
    {
        t = gf_iszero(images[i]) & 1; // 이미지 값이 0인지 체크, 0이면 개가 오류위치

        e[ i/8 ] |= t << (i%8);
        w += t;

    }

#ifdef KAT
  {
    int k;
    printf("decrypt e: positions");
    for (k = 0;k < SYS_N;++k)
      if (e[k/8] & (1 << (k&7)))
        printf(" %d",k);
    printf("\n");
  }
#endif

    synd(s_cmp, g, L, e); //H(e)

    //

    check = w;
    check ^= SYS_T;

    for (i = 0; i < SYS_T*2; i++)
        check |= s[i] ^ s_cmp[i]; // (H(e) + (H(v)))

    check -= 1;
    check >>= 15;

    return check ^ 1;
}
```

- 오류위치 다항식을 찾는 **Berlekamp Massey 디코딩** (bm) 알고리즘이 핵심

Classic McEliece : Berlekamp Massey Decoding

LFSR Synthesis Algorithm (Berlekamp Iterative Algorithm):

1) $1 \rightarrow C(D)$ $\quad 1 \rightarrow B(D)$ $\quad 1 \rightarrow x$
   $0 \rightarrow L$ $\quad 1 \rightarrow b$ $\quad 0 \rightarrow N$
2) If $N = n$, stop. Otherwise compute
$$\boxed{d = s_N + \sum_{i=1}^{L} c_i s_{N-i}.}$$
3) If $d = 0$, then $x + 1 \rightarrow x$, and go to 6).
4) If $d \neq 0$ and $2L > N$, then
   $C(D) - d\, b^{-1} D^x B(D) \rightarrow C(D)$
   $x + 1 \rightarrow x$
   and go to 6).
5) If $d \neq 0$ and $2L \leq N$, then
   $\boxed{C(D) \rightarrow T(D)\ \text{[temporary storage of } C(D)]}$
   $C(D) - \boxed{d\, b^{-1}}\,\boxed{D^x}\,\boxed{B(D)} \rightarrow C(D)$
   $N + 1 - L \rightarrow L$
   $\boxed{T(D) \rightarrow B(D)}$
   $d \rightarrow b$
   $\boxed{1 \rightarrow x.}$
6) $N + 1 \rightarrow N$ and return to 2).

```
for (N = 0; N < 2 * SYS_T; N++)
{
    d = 0;

    for (i = 0; i <= min(N, SYS_T); i++)
        d ^= gf_mul(C[i], s[ N-i]);

    mne = d; mne -= 1;   mne >>= 15; mne -= 1;
    mle = N; mle -= 2*L; mle >>= 15; mle -= 1;
    mle &= mne; //mle = 1111 1111 1111 1111 아니면 0

    for (i = 0; i <= SYS_T; i++)
        T[i] = C[i];

    f = gf_frac(b, d);

    for (i = 0; i <= SYS_T; i++)
        C[i] ^= gf_mul(f, B[i]) & mne;

    L = (L & ~mle) | ((N+1-L) & mle);

    for (i = 0; i <= SYS_T; i++)
        B[i] = (B[i] & ~mle) | (T[i] & mle);
                     (4)                (5)

    b = (b & ~mle) | (d & mle);

    for (i = SYS_T; i >= 1; i--) B[i] = B[i-1];
    B[0] = 0;
}

for (i = 0; i <= SYS_T; i++)
    out[i] = C[ SYS_T-i ];
```

→ End

*
```
B[1] = C[0] = 1;
```

*
```
B[1] = C[0] = 1;
```

Constant-time

```c
void PQCLEAN_MCELIECE348864_CLEAN_bm(gf *out, gf *s) {
    int i;
    uint16_t N = 0;
    uint16_t L = 0;
    uint16_t mle;
    uint16_t mne;

    gf T[ SYS_T + 1  ];
    gf C[ SYS_T + 1 ];
    gf B[ SYS_T + 1 ];

    gf b = 1, d, f;

    for (i = 0; i < SYS_T + 1; i++) {
        C[i] = B[i] = 0;
    }

    B[1] = C[0] = 1;

    for (N = 0; N < 2 * SYS_T; N++) {
        d = 0;

        for (i = 0; i <= min(N, SYS_T); i++) {
            d ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(C[i], s[N - i]);
        }
        mne = d; mne -= 1; mne >>= 15; mne -= 1;
        mle = N; mle -= 2 * L; mle >>= 15; mle -= 1;
        mle &= mne;

        for (i = 0; i <= SYS_T; i++) {
            T[i] = C[i];
        }

        f = PQCLEAN_MCELIECE348864_CLEAN_gf_frac(b, d);

        for (i = 0; i <= SYS_T; i++) {
            C[i] ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(f, B[i]) & mne;
        }

        L = (L & ~mle) | ((N + 1 - L) & mle);

        for (i = 0; i <= SYS_T; i++) {
            B[i] = (B[i] & ~mle) | (T[i] & mle);
        }

        b = (b & ~mle) | (d & mle);

        for (i = SYS_T; i >= 1; i--) {
            B[i] = B[i - 1];
        }
        B[0] = 0;
    }

    for (i = 0; i <= SYS_T; i++) {
        out[i] = C[ SYS_T - i ];
    }
}
```

→

Branch

```c
void PQCLEAN_MCELIECE348864_CLEAN_bm_branch(gf *out, gf *s) {
    int i;
    uint16_t N = 0;
    uint16_t L = 0;

    gf T[ SYS_T + 1  ];
    gf C[ SYS_T + 1 ];
    gf B[ SYS_T + 1 ];
    gf b = 1, d, f;

    for (i = 0; i < SYS_T + 1; i++) {
        C[i] = B[i] = 0;
    }

    B[1] = C[0] = 1;
    for (N = 0; N < 2 * SYS_T; N++) {
        d = 0;

        for (i = 0; i <= min(N, SYS_T); i++) {
            d ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(C[i], s[N - i]);
        }
        if(2*L <= N){
            for (i = 0; i <= SYS_T; i++) { // This part is inefficient in quantum(1)
                    T[i] = C[i];
            }
        }
        f = PQCLEAN_MCELIECE348864_CLEAN_gf_frac(b, d);
        if(d != 0){
            if(2*L > N){
                for (i = 0; i <= SYS_T; i++) {
                    C[i] ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(f, B[i]);
                }
            }
        if(2*L <= N){

            for (i = 0; i <= SYS_T; i++) {
                C[i] ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(f, B[i]);
                L = (N + 1 - L);
            }
            for (i = 0; i <= SYS_T; i++) {  // This part is inefficient in quantum(2)
                B[i] = T[i];
            }
            b=d;
            }
        }
        for (i = SYS_T; i >= 1; i--) {
            B[i] = B[i - 1];
        }
        B[0] = 0;
    }
    for (i = 0; i <= SYS_T; i++) {
        out[i] = C[ SYS_T - i ];
    }
}
```

Result

```
s:
    7f7e7d7c7b7a797877767574737271706f6e6d6c6b6a696867666564636261605f5e5d5c5b5a595857565554535251504f4e4d4c4b4a49
    4847464544434241403f3e3d3c3b3a393837363534333231302f2e2d2c2b2a292827262524232221201f1e1d1c1b1a1918171615141312
    1110fedcba9876543210
locator:
    1904b05904f07904b05904703904b05904f07904b05904702904b05904f07904b05904703904b05904f07904b05904700904b05904f079
    04b05904703904b05904f07904b05904702904b05904f07904b05904703904b05904f07904b05904705
s (branch):
    7f7e7d7c7b7a797877767574737271706f6e6d6c6b6a696867666564636261605f5e5d5c5b5a595857565554535251504f4e4d4c4b4a49
    4847464544434241403f3e3d3c3b3a393837363534333231302f2e2d2c2b2a292827262524232221201f1e1d1c1b1a1918171615141312
    1110fedcba9876543210
locator (branch):
    1904b05904f07904b05904703904b05904f07904b05904702904b05904f07904b05904703904b05904f07904b05904700904b05904f079
    04b05904703904b05904f07904b05904702904b05904f07904b05904703904b05904f07904b05904705
Program ended with exit code: 0
```

Classical

```c
void PQCLEAN_MCELIECE348864_CLEAN_bm_branch(gf *out, gf *s) {
    int i;
    uint16_t N = 0;
    uint16_t L = 0;

    gf T[ SYS_T + 1  ];
    gf C[ SYS_T + 1 ];
    gf B[ SYS_T + 1 ];
    gf b = 1, d, f;

    for (i = 0; i < SYS_T + 1; i++) {
        C[i] = B[i] = 0;
    }

    B[1] = C[0] = 1;
    for (N = 0; N < 2 * SYS_T; N++) {
        d = 0;

        for (i = 0; i <= min(N, SYS_T); i++) {
            d ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(C[i], s[N - i]);
        }
        if(2*L <= N){
            for (i = 0; i <= SYS_T; i++) { // This part is inefficient in quantum(1)
                    T[i] = C[i];
            }
        }
        f = PQCLEAN_MCELIECE348864_CLEAN_gf_frac(b, d);
        if(d != 0){
            if(2*L > N){
                for (i = 0; i <= SYS_T; i++) {
                    C[i] ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(f, B[i]);
                }
            }
            if(2*L <= N){

                for (i = 0; i <= SYS_T; i++) {
                    C[i] ^= PQCLEAN_MCELIECE348864_CLEAN_gf_mul(f, B[i]);
                    L = (N + 1 - L);
                }
                for (i = 0; i <= SYS_T; i++) {  // This part is inefficient in quantum(2)
                    B[i] = T[i];
                }
                b=d;
            }
        }
        for (i = SYS_T; i >= 1; i--) {
            B[i] = B[i - 1];
        }
        B[0] = 0;
    }
    for (i = 0; i <= SYS_T; i++) {
        out[i] = C[ SYS_T - i ];
    }
}
```

Quantum

```python
X | b[0]
X | C[0][0]
X | B[1][0]
t_count = 0
L = 0
for N in range(2*SYS_T):
    d = eng.allocate_qureg(n)
    print(N)
    for i in range(min(N, SYS_T)+1):
        Karatsuba_12_Toffoli_Depth_1_XOR(eng, C[i], s[N-i], r_a, r_b, rr_a, rr_b, rrr, d) ##
        t_count = t_count+1
    #print_state(eng, d, n//4)

    if(2*L <= N):
        for i in range(SYS_T+1):
            qulist = eng.allocate_qureg(12)
            T[i] = qulist
            Copy(eng, C[i], T[i], n)

    f = []
    f = Inversion(eng, b, r_a, r_b, rr_a, rr_b, rrr, d)
    t_count = t_count + 5

    if(2*L > N):
        #print('check2')
        for i in range(SYS_T+1):
            Karatsuba_12_Toffoli_Depth_1_XOR(eng, f, B[i], r_a, r_b, rr_a, rr_b, rrr, C[i])
            t_count = t_count + 1

    if (2 * L <= N):
        #print('check3')
        for i in range(SYS_T+1):
            Karatsuba_12_Toffoli_Depth_1_XOR(eng, f, B[i], r_a, r_b, rr_a, rr_b, rrr, C[i])
            t_count = t_count + 1
            L  = (N + 1 - L)

        for i in range(SYS_T+1):
            B[i] = T[i]

        b = d

    for i in range(SYS_T):
        B[SYS_T-i] = B[SYS_T-1-i]

    qlist = eng.allocate_qureg(n)
    B[0] = qlist

print("Locator:")
if (resource_check != 1):
    for i in range(SYS_T+1):
```

ProjectQ 시뮬레이션 자원 이슈

- **Binary Field Arithmetic: Key pair, Decryption**

Field	Arithmetic	Method	Qubits	Clifford	T gates	T-depth	Full depth
$\mathbb{F}_{2^{12}}$	Addition	·	24	12	·	·	1
	Squaring	·	12	7	·	·	2
	Multiplication	Schoolbook	36	921	1,008	136	307
		Montgomery	47	1,702	1,932	624	1,224
		WISA'22	162	761	378	4	37
	Inversion	Itoh-Tsujii + WISA'22	402	4,758	1,890	20	194
$\mathbb{F}_{2^{13}}$	Addition	·	26	13	·	·	1
	Squaring	·	13	7	·	·	2
	Multiplication	Schoolbook	42	1,110	1,183	148	333
		Montgomery	51	1,950	2,275	728	1,430
		WISA'22	198	966	462	4	54
	Inversion	Itoh-Tsujii + WISA'22	422	4,988	1,848	16	369

- **Information Set Decoding: Cryptanalysis**

Matrix size	8 x 16
Method	QISD
Qubits	384
Clifford gates	11258
T gates	12212
Multi-Controlled Swap	7,816
Depth	3,219

- **Gauss-Jordan Elimination: Key pair, Information Set Decoding (ISD)**

Matrix size	Method	Qubits	X	CX (CNOT)	CCX (Toffoli)	CCCX	Multi-Controlled Swap	Depth
8 x 8	Gauss-Jordan Elimination	88	56	70	140	546	1,064	1,404

- **Matrix Vector Multiplication: Encryption**

Matrix size	Method	Qubits	CNOT	Toffoli	Full Depth
8 x 16	Q-Q	152	·	128	147
	C-Q (Naïve)	24	45	·	14
	C-Q (PLU Decomposition)	16	37	·	13

BIKE도 마찬가지
Field size > 12,000

- **Berlekamp-Massey Decoding: Decryption**

Berlekamp-Massey decoding	Qubits	Clifford gates	T gates	T-depth	Full depth
mceliece348864	888,492	12,823,392	579,384	60,800	363,696

감사합니다