ARMv8 상에서 HAETAE NEON 최적화

https://youtu.be/aiiu3kVlKRY

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

HAETAE

- 격자기반 전자서명 알고리즘
- KpqC 2라운드 진출 & NIST PQC Additional Digital Signature 공모전 1라운드 진출
- Learning With Error(LWE)와 Short Integer Solution(SIS)의 어려움에 기반
- NIST PQC 표준으로 선정된 CRYSTALS-Dilithium에서 영감을 받아 설계
  - Rejection sampling에 대해 Bimodal distribution을 사용
  - Hyperball Uniform Distributions을 사용
- CRYSTALS-Dilithium보다 구현이 복잡하지만, 서명 크기가 작음
  - Dilithium과 동일한 보안 수준에서 ~~30~40%~~ 39% 작은 서명 크기와 ~~20%~~ 25% 작은 검증 크기를 가짐
- NTT 등의 곱셈 연산 코드를 분석해본 결과 일부 파라미터 값을 제외하고 동일함을 확인

Table 1: Parameters of HAETAE($\mathbf{n}$ is ring dimension, $\mathbf{q}$ is fully split modulo integer, $\mathbf{S}$ is smallmagnitude matrix that makes up the secret key, η is infinity norm of the secret key, and τ is hamming weight of the binary challenge.)

Scheme	n	q	η	τ	Security level	Verify key(byte)	Verify key(byte)	Signature(byte)
HAETAE120	256	64,513	1	58	2	992	1,376	1,474
HAETAE180	256	64,513	1	80	3	1,472	2,080	2,349
HAETAE260	256	64,513	1	128	5	2,080	2,720	2,948

ARMv8프로세서

- ARM(**A**dvanced **R**ISC **M**achine)
  - ISA(Instruction Set Architecture) 고성능 임베디드 프로세서
  - 1958년 Acorn사에서 개발한 ARM1에서 시작
    추후 ARM Holdings를 설립하여 개발 시작

- ARMv8 프로세서
  31개의 64비트 general 레지스터(x0~x30)와
  128-bit 32개 벡터 레지스터(v0~v31) 지원

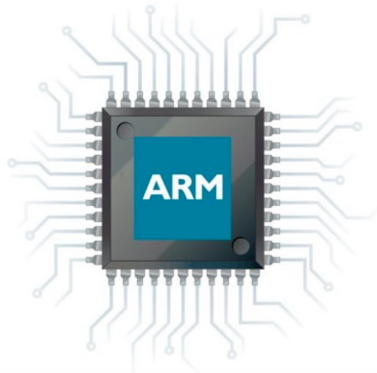
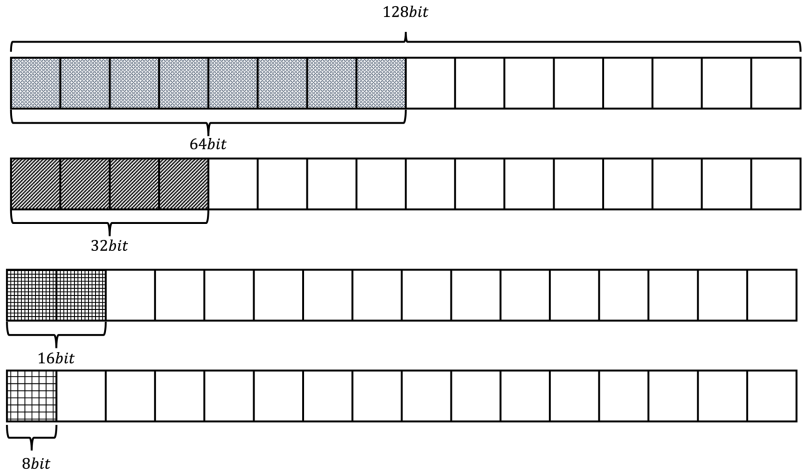


Table 2: Summarized instruction set of ARMv8 for HAETAE; `Xd`, `Vd`: destination register (general, vector), `Xn`, `Vn`, `Vm`: source register (general, vector, vector), `Vt`: transferred vector register.

asm	Operands	Description	Operation
ADD	Vd.T, Vn.T, Vm.T	Add	Vd ← Vn + Vm
LD1	Vt.T, [Xn]	Load multiple single-element structures	Vt ← [Xn]
LD1R	Vt.T, [Xn]	Load single 1-element structure and replicate to all lanes (of one register).	Vt.T ← [Xn]
MOV	Xd, #imm	Move(immediate)	Xd ← imm
MOV	Vd.T, Vn.T	Move(vector)	Vd ← Vn
MOVI	Vt.T, #imm	Move immediate (vector)	Vt ← #imm
MUL	Vd, Vn, Vm	Multiply	Vd ← Vn × Vm
SMULL	Vd, Vn, Vm	Signed Multiply Long(lower half)	Vd ← Vn × Vm
SMULL2	Vd, Vn, Vm	Signed Multiply Long(upper half)	Vd ← Vn × Vm
SMLSL	Vd, Vn, Vm	Signed Multiply-Substract Long(lower half)	Vd ← Vn × Vm
SMLSL2	Vd, Vn, Vm	Signed Multiply-Substract Long(upper half)	Vd ← Vn × Vm
RET	{Xn}	Return from subroutine	Return
SHL	Vd.T, Vn.T, #shift	Shift Left immediate (vector)	Vd ← Vn <<#shift
SSHR	Vd.T, Vn.T, #shift	Signed Shift Right and immediate (vector)	Vd ← Vn >>#shift
ST1	Vt.T, [Xn]	Store multiple single-element structures from one, two, three, or four registers	[Xn] ← Vt
SUB	Xd, Xn, #imm	Subtract immediate	Xd ← Xn - #imm
SUB	Vd, Vn, Vm	Subtract	Vd ← Vn - Vm
REV32	Vd.T, Vn.T	Reverse elements in 32-bit words	Vd ← Vn of Reverse
CBNZ	Wt, Label	Compare and Branch on Nonzero	Go to Label
ZIP1	Vd.T, Vn.T, Vm.T	Zip vectors primary	Vd ← Vn[even], Vm[even] Vd ← Vn[odd], Vm[odd]
ZIP2	Vd.T, Vn.T, Vm.T	Zip vectors secondary	Vd ← Vn[odd], Vm[odd] Vd ← Vn[odd], Vm[odd]
XTN, XTN2	Vd.T, Vn.T, Vm.T	Extracted Narrow	???
TRN1	Vd.T, Vn.T, Vm.T	Transpose vectors primary	Vd ← Vn[even], Vm[even] Vd ← Vn[odd], Vm[odd]
TRN2	Vd.T, Vn.T, Vm.T	Transpose vectors secondary	Vd ← Vn[odd], Vm[odd] Vd ← Vn[odd], Vm[odd]

ARMv8프로세서

SMULL	Vd, Vn, Vm	Signed Multiply Long(lower half)
SMULL2	Vd, Vn, Vm	Signed Multiply Long(upper half)
SMLSL	Vd, Vn, Vm	Signed Multiply-Substract Long(lower half)
SMLSL2	Vd, Vn, Vm	Signed Multiply-Substract Long(upper half)
XTN, XTN2	Vd.T, Vn.T, Vm.T	Extracted Narrow




```
int32_t b[8] = {11111, 33333, 3, 3,
    22222, 55555, 2, 2};
```

```
ld1         {v1.4s}, [x1],#16
ld1         {v2.4s}, [x1],#16

SMULL       v7.2d, v1.2s, v2.2s
SMULL2      v27.2d, v1.4s, v2.4s
XTN         v6.2s,   v7.2d
XTN2        v6.4s,   v27.2d
```

```
//11111 33333 3 3 (v1.4s)  11111 33333 0 0 (v1.2s)
//22222 55555 2 2 (v2.4s)  22222 55555 0 0 (v2.2s)
//246908642 0 1851814815 0 -> [11111*22222] 0 [33333*55555] 0 SMULL
//6         0 6          0 ->[3*2] 0 [3*2] 0                 SMULL2
//246908642 1851814815 0 0
//246908642 1851814815 6 6
```

구현 기법

SMULL	Vd, Vn, Vm	Signed Multiply Long(lower half)	Vd ← Vn × Vm
SMULL2	Vd, Vn, Vm	Signed Multiply Long(upper half)	Vd ← Vn × Vm
SMLSL	Vd, Vn, Vm	Signed Multiply-Substract Long(lower half)	Vd ← Vn × Vm
SMLSL2	Vd, Vn, Vm	Signed Multiply-Substract Long(upper half)	Vd ← Vn × Vm

```
/*************************************************
* Name:       poly_pointwise_montgomery
*
* Description: Pointwise multiplication of polynomials in NTT domain
*                     representation and multiplication of resulting polynomial
*                     by 2^{-32}.
*
* Arguments:   - poly *c: pointer to output polynomial
*                     - const poly *a: pointer to first input polynomial
*                     - const poly *b: pointer to second input polynomial
*************************************************/
#ifdef CLANG
void poly_pointwise_montgomery(poly *c, const poly *a, const poly *b) {
    unsigned int i;

    for (i = 0; i < N; ++i)
        c->coeffs[i] = montgomery_reduce((int64_t)a->coeffs[i] * b->coeffs[i]);
}
#endif
```

```
int32_t montgomery_reduce(int64_t a) {
    int32_t t;

    t = (int64_t)(int32_t)a * QINV;
    t = (a - (int64_t)t * Q) >> 32;
    return t;
}
```

```
.macro mk_Q_Inv
    movi.4s v4, #0x38
    movi.4s v5, #0x0f
    rev32   v4.16b, v4.16b
    shl.4s  v5, v5, #16
    orr.16b v4, v4, v5

    movi.4s v5, #0x04
    rev16   v5.16b, v5.16b
    orr.16b v4, v4, v5
    movi.4s v5, #0x01
    orr.16b v4, v4, v5
.endm

.macro mk_Q
    movi.4s v3, #0xfc
    rev16   v3.16b, v3.16b
    movi.4s v5, #0x01
    orr.16b v3, v3, v5

.endm
```

```
mk_Q //v3
mk_Q_Inv  //v4

mov         x5, #64

loop_i:

    ld1     {v1.4s}, [x1], #16
    ld1     {v2.4s}, [x2], #16

    SMULL   v7.2d, v1.2s, v2.2s
    SMULL2  v27.2d, v1.4s, v2.4s
    XTN  v6.2s, v7.2d
    XTN2 v6.4s, v27.2d

    mul.4s v6, v6, v4

    smlsl v7.2d, v6.2s, v3.2s //하위
    sshr.2d v7, v7, #32

    smlsl2 v27.2d, v6.4s, v3.4s //상위
    sshr.2d v27, v27, #32

    XTN  v6.2s, v7.2d
    XTN2 v6.4s, v27.2d


    ST1 {v6.4s}, [x0], #16

    add         x5, x5, #-1
    cbnz        x5, loop_i
```

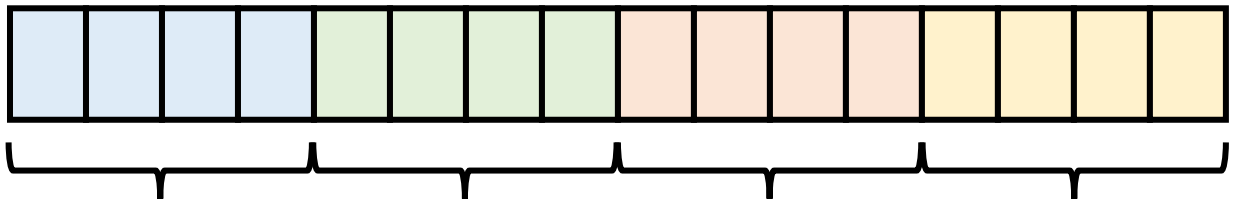
NEON 구현 기법

```
void ntt(int32_t a[N]) {
    unsigned int len, start, j, k;
    int32_t zeta, t;

    k = 0;
    for (len = 128; len > 0; len >>= 1) {
        for (start = 0; start < N; start = j + len) {
            zeta = zetas[++k];
            for (j = start; j < start + len; ++j) {
                t = montgomery_reduce((int64_t)zeta * a[j + len]);
                a[j + len] = a[j] - t;
                a[j] = a[j] + t;
            }
        }
    }
}
```

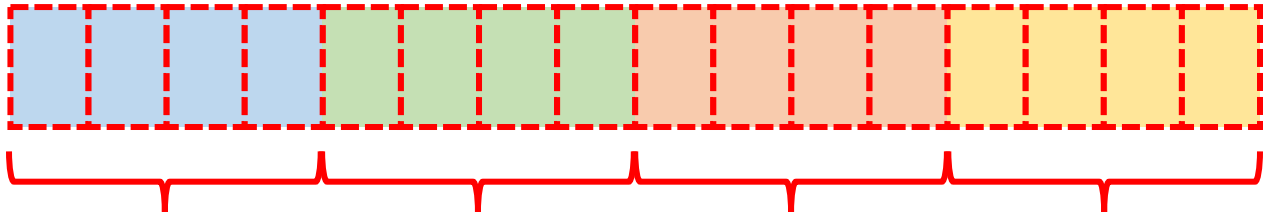
```
.macro len128
    mov         x15, #32            //32*4=128 n = 256
    ld1R        {v2.4s}, [x1], #4         //zeta
    loop_i128:
        ld1         {v1.4s}, [x0]     //a[j]
        add         x0, x0, #512       //4*128=512  4(32bit)
        ld1         {v0.4s}, [x0]      //a[j+len]

        SMULL       v7.2d, v0.2s, v2.2s
        SMULL2      v27.2d, v0.4s, v2.4s
        XTN         v6.2s, v7.2d
        XTN2        v6.4s, v27.2d              //t = v6

        mont_reduce

        sub.4s      v0, v1,v6
        ST1         {v0.4s}, [x0]
        add         x0, x0, #-512
        add.4s      v1, v1, v6
        ST1         {v1.4s}, [x0], #16

        add         x15, x15, #-1
        cbnz        x15, loop_i128
.endm
```

구현 기법

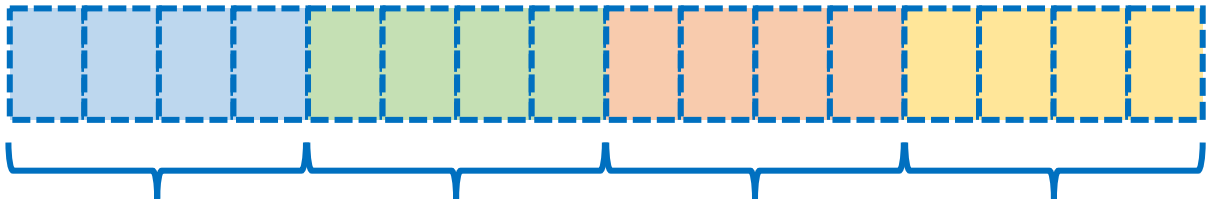
[j] 0, [j+len] 2 [zeta] 11591	[j] 0, [j+len] 1 [zeta] -27989
[j] 1, [j+len] 3 [zeta] 11591	[j] 2, [j+len] 3 [zeta] 3350
[j] 4, [j+len] 6 [zeta] 7814	[j] 4, [j+len] 5 [zeta] 787
[j] 5, [j+len] 7 [zeta] 7814	[j] 6, [j+len] 7 [zeta] -13857
[j] 8, [j+len] 10 [zeta] 12697	[j] 8, [j+len] 9 [zeta] 1657



$zeta[0]$ $zeta[1]$ $zeta[2]$ $zeta[3]$



$a[0]$ $a[1]$ $a[2]$ $a[3]$



$a[4]$ $a[5]$ $a[6]$ $a[7]$

```
.macro len1 //8개 한 번에
mov         x13, #32
loop_zeta7:
    ld1         {v2.4s}, [x1],#16          //zeta


    ld1         {v1.2s}, [x0],#8      //a[j]
    ld1         {v0.2s}, [x0],#8       //a[j+len]

    trn1.4s          v17, v1, v0
    trn2.4s          v18, v1, v0                masking

    ld1         {v1.2s}, [x0],#8      //a[j]
    ld1         {v0.2s}, [x0]        //a[j+len]
    trn1.2s        v12, v1, v0
    trn2.2s        v19, v1, v0

    zip1.2d          v8, v17, v12      //a[j]
    zip1.2d          v9, v18, v19     //a[j+len]

    SMULL       v7.2d, v9.2s, v2.2s
    SMULL2      v27.2d, v9.4s, v2.4s
    XTN         v6.2s, v7.2d
    XTN2        v6.4s, v27.2d                 //t = v6

    mul.4s      v6, v6, v4

    smlsl       v7.2d, v6.2s, v3.2s //하위
    sshr.2d     v7, v7, #32

    smlsl2      v27.2d, v6.4s, v3.4s //상위
    sshr.2d     v27, v27, #32

    XTN         v6.2s, v7.2d
    XTN2        v6.4s, v27.2d

    sub.4s      v9, v8, v6
    add.4s      v8, v8, v6

    zip1.4s          v0, v8, v9
    zip2.4s          v1, v8, v9                unmasking

    add         x0, x0, #-24
    ST1         {v0.4s}, [x0], #16
    ST1         {v1.4s}, [x0], #16

    add         x13, x13, #-1
    cbnz        x13, loop_zeta7

.endm
```

구현 기법

```
void invntt_tomont(int32_t a[N]) {
    unsigned int start, len, j, k;
    int32_t t, zeta;
    const int32_t f = -29720; // mont^2/256

    k = 256;
    for (len = 1; len < N; len <<= 1) {
        for (start = 0; start < N; start = j + len) {
            zeta = -zetas[--k];
            for (j = start; j < start + len; ++j) {
                t = a[j];
                a[j] = t + a[j + len];
                a[j + len] = t - a[j + len];
                a[j + len] = montgomery_reduce((int64_t)zeta * a[j + len]);
            }
        }
    }

    for (j = 0; j < N; ++j) {
        a[j] = montgomery_reduce((int64_t)f * a[j]);
    }
}
static const int32_t zetas[N] = {
    0,      26964,  -16505, 22229,  30746,  20243,  19064,  -31218, 9395,
    -30985, 22859,  -8851,  32144,  13744,  21408,  17599,  -16039, -22946,
    6241,   -19553, 10681,  22935,  22431,  -29104, 28147,  -27527, -29133,
    -20035, 20143,  -11361, 30820,  25252,  -22562, -6789,  -10049, 9383,
    16304,  -12296, 16446,  18239,  -1296,  -19725, -32076, 11782,  -17941,
    29643,  -8577,  7893,   -21464, -19646, -15130, -2391,  30608,  -23970,
    -16608, 19616,  -7941,  26533,  -19129, 27690,  7597,   -11459, 10615,
    -9430,  11591,  7814,   12697,  32114,  -3761,  -9604,  19813,  20353,
    17456,  -16267, -19555, 598,    -29942, 4538,   835,    15546,  3970,
    -27685, 1488,   8311,   -12442, 31352,  -17631, 1806,   -5342,  9790,
    29068,  16507,  -29051, 22131,  6759,   15510,  -14941, 28710,  1160,
    -31327, 24985,  11261,  -10623, -27727, 21502,  18731,  -16186, -4127,
    -18832, 12050,  -14501, 7929,   29563,  -31064, 5913,   5322,   -16405,
    2844,   29439,  5876,   -9522,  -18586, -9874,  23844,  30362,  -21442,
    9560,   17671,  -27989, 3350,   787,    -13857, 1657,   -21224, -7374,
    -9190,  2464,   25555,  -3529,  -28772, 16588,  -15739, 23475,  13666,
    5764,   30980,  13633,  -7401,  -30317, 28847,  7682,   -11808, -8796,
    14864,  -24162, -19194, 689,    -1311,  -31332, -16319, 1025,   10971,
    -23016, -2648,  -21900, -12543, -25921, 28254,  28521,  -16160, 12380,
    -12882, -30332, -16630, 23439,  7742,   17182,  17494,  5920,   13642,
    7382,   -18166, 21422,  -30274, -28190, 13283,  -20316, -9939,  10672,
    21454,  6080,   -17374, -29735, -25912, -10170, 3808,   10639,  -26985,
    -10865, 25636,  17261,  -26851, -8253,  -3304,  18282,  -2202,  -31368,
    -22243, 13882,  12069,  -11242, -7729,  -10226, 1761,   -27298, -4800,
    -17737, -22805, -3528,  65,     10770,  8908,   -23751, 26934,  21921,
    -27010, -21944, 8889,   -1035,  23224,  -9488,  -5823,  -994,   -20206,
    7655,   -16251, -22820, -27740, 15822,  23078,  13803,  -8099,  2931,
    9217,   -21126, -14203, 25492,  -12831, 7947,   17463,  -12979, 29003,
    31612,  26554,  8241,   -20175}; // q = 64513
```

```
#ifdef ntt_clang
#define ntt HAETAE_NAMESPACE(ntt)
void ntt(int32_t a[N]);
#define invntt_tomont HAETAE_NAMESPACE(invntt_tomont)
void invntt_tomont(int32_t a[N]);

#else
extern void ntt(int32_t a[N], int32_t *b);
extern void invntt_tomont(int32_t a[N], int32_t *b);
#endif
```

```
static const int32_t inv_zetas[N] = {
    20175, -8241, -26554, -31612, -29003, 12979, -17463, -7947, 12831,
    -25492, 14203, 21126, -9217, -2931, 8099, -13803, -23078, -15822,
    27740, 22820, 16251, -7655, 20206, 994, 5823, 9488, -23224,
    1035, -8889, 21944, 27010, -21921, -26934, 23751, -8908, -10770,
    -65, 3528, 22805, 17737, 4800, 27298, -1761, 10226, 7729,
    11242, -12069, -13882, 22243, 31368, 2202, -18282, 3304, 8253,
    26851, -17261, -25636, 10865, 26985, -10639, -3808, 10170, 25912,
    29735, 17374, -6080, -21454, -10672, 9939, 20316, -13283, 28190,
    30274, -21422, 18166, -7382, -13642, -5920, -17494, -17182, -7742,
    -23439, 16630, 30332, 12882, -12380, 16160, -28521, -28254, 25921,
    12543, 21900, 2648, 23016, -10971, -1025, 16319, 31332, 1311,
    -689, 19194, 24162, -14864, 8796, 11808, -7682, -28847, 30317,
    7401, -13633, -30980, -5764, -13666, -23475, 15739, -16588, 28772,
    3529, -25555, -2464, 9190, 7374, 21224, -1657, 13857, -787,
    -3350, 27989, -17671, -9560, 21442, -30362, -23844, 9874, 18586,
    9522, -5876, -29439, -2844, 16405, -5322, -5913, 31064, -29563,
    -7929, 14501, -12050, 18832, 4127, 16186, -18731, -21502, 27727,
    10623, -11261, -24985, 31327, -1160, -28710, 14941, -15510, -6759,
    -22131, 29051, -16507, -29068, -9790, 5342, -1806, 17631, -31352,
    12442, -8311, -1488, 27685, -3970, -15546, -835, -4538, 29942,
    -598, 19555, 16267, -17456, -20353, -19813, 9604, 3761, -32114,
    -12697, -7814, -11591, 9430, -10615, 11459, -7597, -27690, 19129,
    -26533, 7941, -19616, 16608, 23970, -30608, 2391, 15130, 19646,
    21464, -7893, 8577, -29643, 17941, -11782, 32076, 19725, 1296,
    -18239, -16446, 12296, -16304, -9383, 10049, 6789, 22562, -25252,
    -30820, 11361, -20143, 20035, 29133, 27527, -28147, 29104, -22431,
    -22935, -10681, 19553, -6241, 22946, 16039, -17599, -21408, -13744,
    -32144, 8851, -22859, 30985, -9395, 31218, -19064, -20243, -30746,
    -22229, 16505, -26964, -29720
};
```

성능 평가

- 구현 환경
  - Apple M1 칩이 탑재된 Apple Macbook Pro 13(3.2GHz)
  - Framework: Xcode Integrated Development Environment
  - Compiled with –O3 option(i.e. fastest)

- 평가 방법
  - ARMv8 상에서 HAETAE 최적화 구현이 없기 때문에 성능은 KPQClean 프로젝트 reference-C와 비교
  - 추가적으로 작년 동계 구현물과 성능 비교
  - 동작시간 및 clock cycle을 측정하기 위해 Multiplier를 **1,000,000**회 반복
  - 동작시간 및 clock cycle을 측정하기 위해 HAETAE알고리즘을 **10,000**회 반복

성능 평가

ms : 작년에 사용하던 성능 측정 기법 동일하게 사용
Clock cycle : m1cycles 코드 사용

1,000,000회 반복 ms			
Algorithm	Reference-C(KpqClean)	**This work**	**Diff.**
NTT	1,339	**1,174**	**1.14x**
Inverse NTT	3,904	**1,349**	**2.89x**
poly pointwise montgomery	917	**337**	**2.72x**
poly add	438	**201**	**2.18.x**
poly sub	437	**202**	**2.16x**

10,000회 반복										
Scheme	Unit	Reference-C(KpqClean)			**This work**			**Diff.**		
		Keygen	Sign	Verify	**Keygen**	**Sign**	**Verify**	**Keygen**	**Sign**	**Verify**
HAETAE2	ms	493	12,646	12,016	**461**	**12,302**	**11,702**	**1.07x**	**1.03x**	**1.03x**
	cc	172,550	4,426100	4,205,600	**161,350**	**4,305,700**	**4,095,700**			
HAETAE3	ms	948	25,115	24,033	**913**	**24,786**	**23,934**	**1.04x**	**1.01x**	**1.00x**
	cc	331,800	8,790,250	8,411,550	**319,550**	**8,675,100**	**8,376,900**			
HAETAE5	ms	2,494	59,705	58,013	**2,463**	**59,561**	**57,870**	**1.01x**	**1.00x**	**1.00x**
	cc	872,900	20,896,750	20,304,550	**862,050**	**20,846,350**	**20,254,500**			

Q & A