SmartM2M
스마트엠투엠

KpqC 암호의 경량 프로세서 구현 특성 분석

- KpqC 6차 워크숍 -

(주)스마트엠투엠
김 호 원

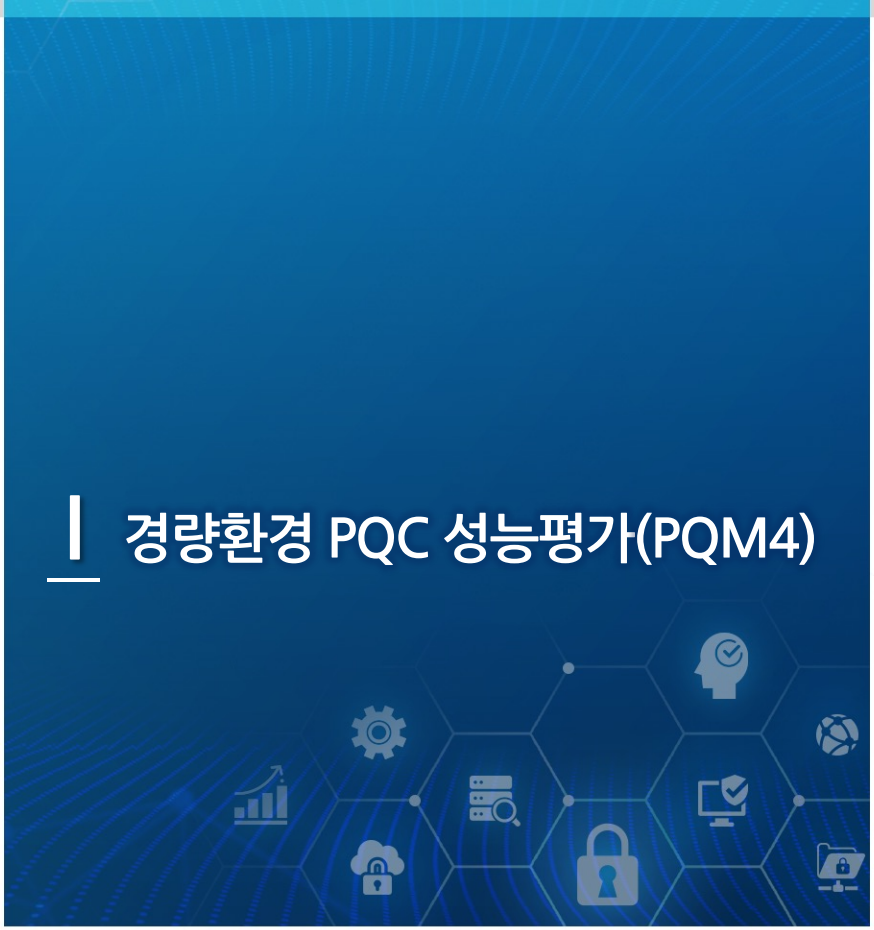
2023.07.13

CONTENTS

KpqC 암호 경량 프로세서 구현 특성 분석

1. 경량 환경 PQC 성능평가(PQM4)
2. 경량 환경 KpqC 구현 고려사항
3. 경량 환경 KpqC 성능평가

Ⅰ 경량환경 PQC 성능평가(PQM4)



SmartM2M
스마트엠투엠

1 경량 환경 PQC 성능평가(PQM4)

1.1 KpqC 제안 알고리즘에 대한 경량환경에서의 성능 평가

- 경량 환경인 Cortex-M4 보드(STM32F407G-DISC1)를 사용하여, KpqC 알고리즘에 대한 성능 평가 수행
  - 이는 NIST PQC 성능 평가를 위한 경량 환경과 동일함

번호	구분		암호 알고리즘	Principal Submitter
1	공개키암호	Graph	IPCC	Yongjin Yeom
2	공개키암호	Code	REDOG	JonLark Kim
3	공개키암호/ KEM	Code	PALOMA	Dongchan Kim
4	KEM	Code	Layered ROLLO-I	Chanki Kim
5	KEM	Lattice	NTRU+	Jonghwan Park (상명대학교)
6	KEM	Lattice	SMAUG	Junghee Cheon (서울대학교)
7	KEM	Lattice	TiGER	Seunghwan Park
8	전자서명	MPC-in-the-Head	AIMer	Jooyoung Lee
9	전자서명	Code	Enhanced pqsigRM	Jongseon No (서울대학교)
10	전자서명	Isogeny	FIBS	Suhri Kim
11	전자서명	Lattice	GCKSign	Jonghwan Park (상명대학교)
12	전자서명	Lattice	HAETAE	Junghee Cheon (서울대학교)
13	전자서명	UOV	MQ-Sign	Kyungah Shim
14	전자서명	Lattice	NCC-Sign	Kyung-Ah Shim
15	전자서명	Lattice	Peregrine	Young-Sik Kim
16	전자서명	Lattice	SOLMAE	Kwangjo Kim

KpqC Round 1 제안 알고리즘 목록

STM32F407G-DISC1

- STM32F407G-DISC1 주요 사양
  - 32-bit ARM Cortex M4 with FPU core
  - 1-Mbyte Flash Memory
  - 192-Kbyte RAM

STM32F407G-DISC1 정보 : https://www.st.com/en/evaluation-tools/stm32f4discovery.html
F: Floating Point, 407: 모델명, G: variant 모델명, DISC1: Discovery 보드 형태(개발 보드를 의미함)

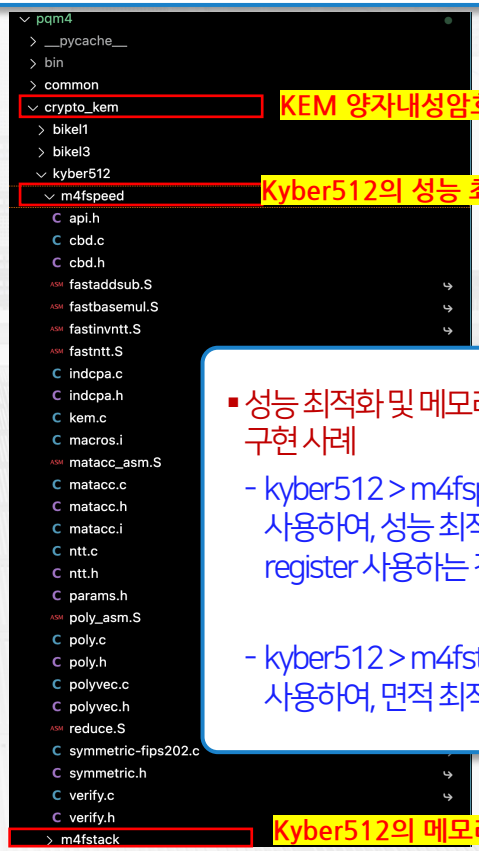
1 경량 환경 PQC 성능평가(PQM4)

1.1 KpqC 제안 알고리즘에 대한 경량환경에서의 성능 평가

- PQM4에서 사용된 성능 평가 기법을 활용하여, KpqC에 대한 성능 및 메모리 특성 평가
  - PQM4에서는 성능 및 메모리 사용량 측정을 위한 프레임워크 제공함 → 이를 kpqC 성능 평가에 사용
  - kpqC 구현시, PQM4에서 제공하는 SHA-2, SHA-3, AES를 활용함
  - "구현수준: Clean" 기반의 kpqC 성능 평가를 수행하고, (일부에 대한) 성능 향상 제안

구현 수준	설명
clean	NIST 제출 코드에서 외부 라이브러리 의존성을 삭제하고 Cortex-M4 에서 동작가능하도록 수정한 코드
ref	NIST에 제출된 소스코드 버전
opt	NIST에 제출된 소스코드의 C언어 레벨 최적 구현 버전
m4	Cortex-M4 대상 어셈블리 레벨 최적 구현 버전
m4f	Cortex-M4 대상 어셈블리 레벨(floating-point 레지스터 활용) 최적 구현 버전

〈 사례: PQM4에서 제공하는 다섯가지 NIST PQC 구현 수준 〉



- 성능 최적화 및 메모리 최적화로 구현된 kyber512 구현 사례
  - kyber512 > m4fspeed: Floating point 레지스터를 사용하여, 성능 최적화된 코드 (Floating point register 사용하는 경우)
  - kyber512 > m4fstack: Floating point 레지스터를 사용하여, 면적 최적화된 코드

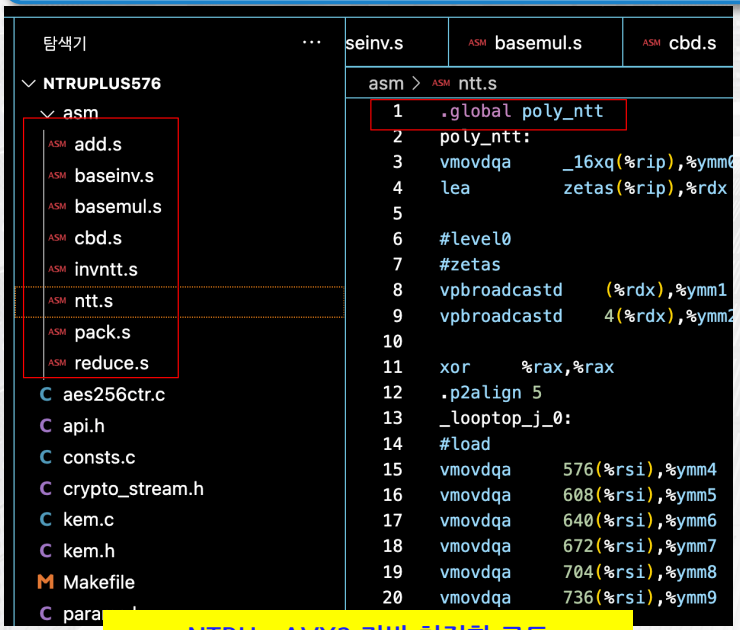
Ⅱ 경량환경 KpqC 구현 고려사항

SmartM2M
스마트엠투엠

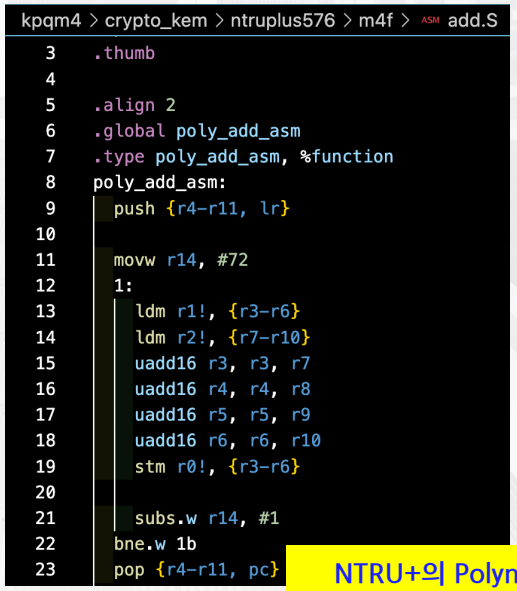
2 경량환경 KpqC 구현 고려사항

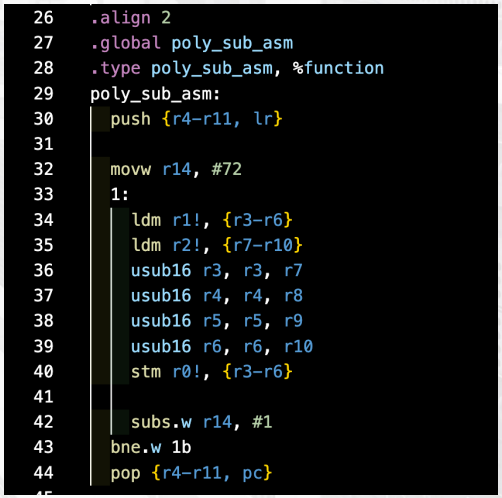
2.1 다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- **Lattice 기반 KpqC 암호 고속화를 위해서는 다항식 곱셈과 Modular reduction 최적화 필요**
  - 다항식 곱셈 연산(NTT, FFT, Toom-Cook 등), Modular reduction 연산(Montgomery, Barret Reduction 등)
  - PQM4에서는 대부분의 NIST PQC를 Assembly 레벨로 최적 구현하여, 성능을 평가함
- **ARM SIMD 명령어를 사용한 성능 향상 (병렬 처리 가능한 다항식 연산(add, sub 등)의 경우)**
  - NTRU+의 경우 AVX2 기반 다항식 연산 최적화 수행(kpqC 제안 버전)
    → ARM SIMD 기반 고속화 가능
    → 다항식 연산(ADD/SUB)을 어셈블리로 구현할 경우, Encap, Decap 에서 5000 cycles의 성능 향상 가능
  - NTT, INTT를 어셈블리어로 구현 시, 더 많은 성능 향상 기대



NTRU+ AVX2 기반 최적화 코드



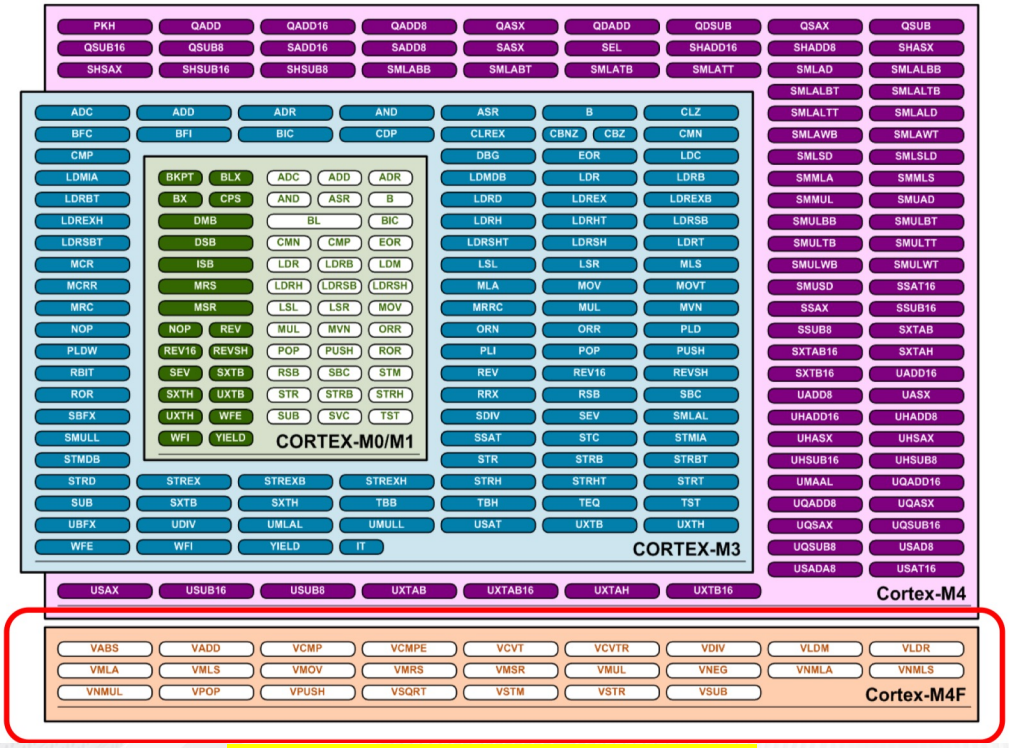


NTRU+의 Polynomial ADD/SUB 연산을
ARM SIMD 명령어를 활용하여 구현한 코드

2 경량환경 KpqC 구현 고려사항

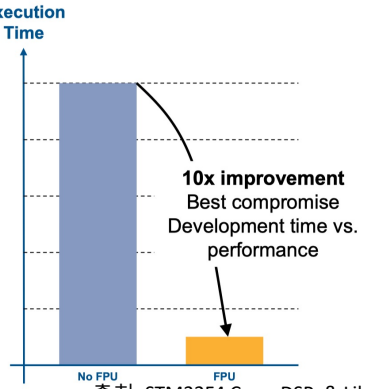
2.2 부동소수점(Floating Point) 연산 사용

- 부동소수점을 사용하는 KpqC 암호는 FPU(Floating Point Unit) 지원이 필수적
- FPU를 지원하는 Cortex-M 보드의 경우 부동 소수점 연산을 효율적으로 할 수 있음
  - PQM4 오픈소스 프로젝트에서도 FPU를 활용한 소스코드가 존재(BIKE, Falcon, Kyber 등)
  - 경량환경의 FPU 지원 유무에 따라 성능은 10배 가까이 차이날 수 있음
  - 부동소수점 연산을 사용하지 않더라도 FPU 레지스터를 활용하여 성능향상 등에 활용 가능



Cortex-M4 FPU Instructions

Time execution comparison for a 29 coefficient FIR on float 32 with and without FPU (CMSIS library)



출처: STM32F4 Core, DSP, & Library, STMicroelectronics

FPU 유무에 따른 부동소수점 연산 성능 차이

2 경량환경 KpqC 구현 고려사항

2.2 부동소수점(Floating Point) 연산 사용

- **FPU 유무에 따른 Falcon 성능 비교**
  - 경량환경에서 보드에서 지원하는 FPU를 활용한 경우와 Software Emulation을 통해 구현한 부동소수점 연산을 활용하는 경우의 키생성은 1.66배, Sign은 6.16배의 차이가남 (Cortex-M7 환경)
  - FFT의 경우10배 이상의 성능 차이가 남[1,2)]

Table 2: Benchmarking performances of Falcon on the ARM Cortex M7 using the STM32F767ZI NUCLEO-144 development board.

Parameter Set	Operation	FPU/EMU	Min (KCyc)	Avg (KCyc)	Max (KCyc)	SDev/Err (KCyc)	Avg (ms)
Falcon-512	Key Gen	FPU	44,196	77,475	256,115	226k/7k	358.7
Falcon-512	Key Gen	EMU	76,809	128,960	407,855	303k/9k	597.0
FPU vs EMU	Key Gen	-	1.74x	**1.66x**	1.59x	-/-	**1.66x**
M7 vs M4	Key Gen	-	2.32x	**2.21x**	2.26x	-/-	**2.84x**
Falcon-1024	Key Gen	FPU	127,602	193,707	807,321	921k/29k	896.8
Falcon-1024	Key Gen	EMU	202,216	342,533	1,669,083	2.4m/76k	1585.8
FPU vs EMU	Key Gen	-	1.58x	**1.76x**	2.07x	-/-	**1.77x**
M7 vs M4	Key Gen	-	2.14x	**2.56x**	1.71x	-/-	**3.41x**
Falcon-512	Sign Dyn	FPU	4,705	4,778	4,863	149/4	22.1
Falcon-512	Sign Dyn	EMU	29,278	29,447	29,640	188/6	136.3
FPU vs EMU	Sign Dyn	-	6.22x	**6.16x**	6.10x	-/-	**6.17x**
M7 vs M4	Sign Dyn	-	8.24x	**8.16x**	8.07x	-/-	**11.66x**
Falcon-1024	Sign Dyn	FPU	10,144	10,243	10,361	1408/44	47.4
Falcon-1024	Sign Dyn	EMU	64,445	64,681	64,957	3k/101	299.5
FPU vs EMU	Sign Dyn	-	6.35x	**6.31x**	6.27x	-/-	**6.32x**
M7 vs M4	Sign Dyn	-	8.36x	**8.31x**	8.19x	-/-	**11.80x**
Falcon-512	Verify	FPU	558	559	561	2.9/0.1	2.6
Falcon-512	Verify	EMU	561	565	570	98/3	2.6
FPU vs EMU	Verify	-	1.01x	**1.01x**	1.02x	-/-	**1.0x**
M7 vs M4	Verify	-	0.83x	**0.85x**	0.86x	-/-	**1.16x**
Falcon-1024	Verify	FPU	1,135	1,136	1,141	23/0.7	5.3
Falcon-1024	Verify	EMU	1,129	1,130	1,135	6.4/0.2	5.2
FPU vs EMU	Verify	-	0.99x	**0.99x**	0.99x	-/-	**0.98x**
M7 vs M4	Verify	-	0.85x	**0.86x**	0.87x	-/-	**1.16x**

Table 8: Profiling Falcon on the ARM Cortex M7 using the STM32F767ZI NUCLEO-144 development board. All values reported are in KCycles.

Key Generation	**512-FPU**	**512-EMU**	**Vs.**	**1024-FPU**	**1024-EMU**	**Vs.**
total ntru gen	77,095 (99%)	127,828 (100%)	**1.66x**	186,120 (100%)	332,876 (100%)	**1.79x**
—poly small mkgauss	34,733 (45%)	34,805 (27%)	1.00x	56,509 (30%)	57,033 (17%)	1.00x
—poly small sqnorm	28 (0.04%)	29 (0.02%)	1.04x	94 (0.05%)	94 (0.03%)	1.00x
—poly small to fp	40 (0.05%)	306 (0.24%)	**7.65x**	132 (0.07%)	989 (0.30%)	**7.50x**
—fft multiply	609 (0.80%)	10,496 (8%)	**17.2x**	2,277 (1%)	38,681 (12%)	**17.00x**
—poly invnorm2 fft	110 (0.14%)	1,446 (1%)	**13.2x**	421 (0.22%)	4,777 (1%)	**11.00x**
—poly adj fft	23 (0.03%)	12 (0.01%)	0.52x	70 (0.04%)	43 (0.01%)	0.60x
—poly mulconst	69 (0.09%)	354 (0.28%)	**5.13x**	218 (0.12%)	1,168 (0.35%)	**5.36x**
—poly mul autoadj fft	63 (0.08%)	383 (0.30%)	**6.08x**	237 (0.13%)	1272 (0.38%)	**5.37x**
—ifft multiply	683 (0.90%)	10,666 (8%)	**15.6x**	2,544 (1.36%)	39,071 (12%)	**15.4x**
—bnorm/fpr add	14 (0.02%)	184 (0.14%)	**13.1x**	35 (0.02%)	424 (0.13%)	**12.1x**
—compute public key	383 (0.49%)	383 (0.30%)	1.00x	887 (0.50%)	887 (0.27%)	1.00x
—solve ntru:	40,337 (52%)	68,764 (54%)	**1.70x**	122,696 (66%)	188,438 (56%)	**1.54x**
encode priv key	26 (0.03%)	26 (0.02%)	1.00x	52 (0.03%)	52 (0.02%)	1.00x
recomp sk and encode	384 (0.50%)	385 (0.3%)	1.00x	815 (0.44%)	815 (0.24%)	1.00x
Signing Dynamic	**512-FPU**	**512-EMU**	**Vs.**	**1024-FPU**	**1024-EMU**	**Vs.**
sign start	4 (0.08%)	4 (0.01%)	1.00x	4 (0.04%)	4 (0.01%)	1.00x
decode/comp priv key	488 (11%)	489 (1.69%)	1.00x	1,040 (11%)	1,040 (2%)	1.00x
hash mess to point	<1 (0.01%)	<1 (0.00%)	0.10x	<1 (0.00%)	<1 (0.00%)	1.00x
signature encode	11 (0.26%)	11 (0.04%)	1.00x	22 (0.24%)	22 (0.03%)	1.00x
convert basis to fft	241 (6%)	3,885 (13%)	**16.1x**	549 (6%)	8,751 (14%)	**15.9x**
comp gram matrix	67 (2%)	628 (2%)	**9.37x**	167 (2%)	1,290 (2%)	**7.72x**
apply lattice basis	89 (2%)	1,250 (4%)	**14.0x**	207 (2%)	2,756 (4%)	**13.3x**
ffsampling	2,814 (66%)	16,190 (56%)	**5.75x**	6,009 (65%)	35,324 (56%)	**5.88x**
recomp matrix basis	258 (6%)	3,900 (14%)	**15.1x**	586 (6%)	8,787 (14%)	**15.0x**
get lattice point	314 (7%)	2,527 (9%)	**8.05x**	706 (8%)	5,564 (8%)	**7.88x**
Verifying	**512-FPU**	**512-EMU**	**Vs.**	**1024-FPU**	**1024-EMU**	**Vs.**
verf start	<1 (0.06%)	<1 (0.06%)	1.00x	<1 (0.03%)	<1 (0.00%)	1.00x
get degree via pk	<1 (0.01%)	<1 (0.01%)	1.00x	<1 (0.00%)	<1 (0.00%)	1.00x
decode pub key	9 (1.6%)	9 (2%)	1.00x	18 (2%)	18 (2%)	1.00x
decode sign	12 (2%)	12 (2%)	1.00x	24 (2%)	24 (2%)	1.00x
hash mess to point	312 (55%)	311 (55%)	1.00x	595 (52%)	595 (52%)	1.00x
verify sign	231 (41%)	231 (41%)	1.00x	501 (44%)	501 (44%)	1.00x

1) Howe, James, and Bas Westerbaan. "Benchmarking and analysing the nist pqc finalist lattice-based signature schemes on the arm cortex m7." *Cryptology ePrint Archive* (2022).
2) Pornin, Thomas. "New efficient, constant-time implementations of falcon." *Cryptology ePrint Archive* (2019).

2 경량환경 KpqC 구현 고려사항

2.3 외부 의존성 라이브러리 제거 및 공통 암호모듈 사용

- **외부 의존성 라이브러리 제거**
  - KpqC 알고리즘 Reference 구현 코드들은 대부분 OpenSSL 기반의 AES, SHA를 사용
  - 또는, 프로세서에서 제공하는 AES-NI 명령어를 사용한 구현을 활용 → ARM에서 동작하지 않음
- **공통 암호모듈 사용**
  - KpqC 암호 알고리즘들이 공통적으로 사용하는 AES/SHA 구현은 통일하여 정확한 성능 평가를 수행해야함
  - 이를 위해, PQM4에서는 Cortex-M4에 최적화된 AES/SHA 구현을 제공하고 있으며, 이를 KpqC에 적용 가능

※ PQM4 AES 구현 참고논문 : Adomnicai, Alexandre, and Thomas Peyrin. "Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V." Cryptology ePrint Archive (2020).

※ PQM4 SHA256/512 구현 참고 : http://bench.cr.yp.to/supercop.html (supercop의 SHA 256/512 레퍼런스 구현 활용)

```
80  /** single, by-the-book AES encryption with AES-NI */
81  static inline void aesni_encrypt1(unsigned char *out, unsigned char *n, __m128i
    rkeys[16]) {
82    __m128i nv = _mm_load_si128((const __m128i *)n);
83    int i;
84    __m128i temp = _mm_xor_si128(nv, rkeys[0]);
85  #pragma unroll(13)
86    for (i = 1 ; i < 14 ; i++) {
87      temp = _mm_aesenc_si128(temp, rkeys[i]);
88    }
89    temp = _mm_aesenclast_si128(temp, rkeys[14]);
90    _mm_store_si128((__m128i*)(out), temp);
91  }
```

NTRU+ Reference 코드에서 사용중인 AES 코드 일부 (AES-NI 명령어 사용)

```
542 ********************************************************************************
543 @ void aes256_encrypt_ffs(u8* ctext, u8* ctext_bis, const u8* ptext,
544 @                            const u8* ptext_bis, const u32* rkey);
545 .global aes256_encrypt_ffs
546 .type   aes256_encrypt_ffs,%function
547 .align 2
548 aes256_encrypt_ffs:
549     push    {r0-r12,r14}
550     sub.w   sp, #56                 // allow space on the stack for tmp var
551     ldr.w   r4, [r2]                // load the 1st 128-bit blocks in r4-r7
552     ldr     r5, [r2, #4]
553     ldr     r6, [r2, #8]
554     ldr     r7, [r2, #12]
555     ldr.w   r8, [r3]                // load the 2nd 128-bit blocks in r8-r11
556     ldr     r9, [r3, #4]
557     ldr     r10,[r3, #8]
558     ldr     r11,[r3, #12]
559     ldr.w   r1, [sp, #112]          // load 'rkey' argument from the stack
560     str.w   r1, [sp, #48]           // store it there for 'add_round_key'
561     bl      packing                 // pack the 2 input blocks
562     bl      ark_sbox                // ark + sbox (round 0)
```

PQM4에서 제공하는 AES 어셈블리 최적화 구현 코드

2 경량환경 KpqC 구현 고려사항

2.4 메모리 사용량 관리

- 경량 환경인 Cortex-M4 보드(STM32F407G-DISC1)는 작은 메모리 사이즈를 가짐
  - 전체 메모리 크기는 192KB이지만, 64KB는 CCM(Core Coupled Memory)로 사용됨
  - KpqC 구현에서 실질적으로 사용할 수 있는 메모리의 크기는 128KB임
- KpqC 알고리즘에 사용되는 Parameter의 Lifecycle을 고려하여 변수 선언 필요
  - Parameter의 Lifecycle에 따라 지역/전역 변수, 동적 할당 등을 사용하여 메모리(Stack, Heap, Data) 관리 필요
- Classic McEliece의 경우 공개키 사이즈 문제로 PQM4 성능평가 대상에서 제외됨[1)]
  - 공개키 사이즈가 최소 260KB로 메모리만 사용할 경우 Cortex-M4 환경에서 성능평가 불가능
  - 이를 해결하기 위해, Flash Memory를 활용하여 Cortex-M4에서 평가한 사례가 있음[2)](McEliece 저자 포함)

구분	Public Key	Private Key
mceliece348864	261,120	6,492
mceliece460896	524,160	13,608
mceliece6688128	1,044,992	13,932
mceliece6960119	1,047,319	13,948
mceliece8192128	1,357,824	14,120

Classic Mceliece Parameter 크기(단위 : byte)

```
#if 0 == _PK_ROW_64BIT_ALIGNED_
        if( (0==l) ) {
            flashpk_start[row/2] = pkmat[ row*len ];
        } else
#endif  // 0 == _PK_ROW_64BIT_ALIGNED_
        {
            uint64_t dd = flashpk_remain[row];
            dd |= ((uint64_t)pkmat[row*len])<<32;
            flashpk_program_2words(dest_address-4,dd);
        }
        dest_address += 4;
        row_idx += 1;
        row_len --;
    }
    flashpk_program_n_2words( dest_address , &pkmat[row*len+row_idx] , row_len>>1 );
    if( 1 == (row_len&1) ) flashpk_remain[row] = pkmat[row*len+len-1];
}
```

Classic McEliece의 큰 공개키를 Flash Memory에 저장하는 경우 Cortex-M4 상에 구현이 가능

Flash Memory를 활용한 Classic Mceliece 구현 코드 일부

1) Kannwischer, Matthias J., et al. "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4." (2019).
2) Ming-Shing Chen and Tung Chou, "Classic McEliece on the ARM Cortex-M4." IACR Transactions on Cryptographic Hardware and Embedded Systems Vo. 2021, No.3 pp.125-148

2 경량환경 KpqC 구현 고려사항

2.4 메모리 사용량 관리

- KpqC 알고리즘의 메모리 개선 방향 - HAETAE 사례
  - HAETAE는 Encoder, Decoder 등에 사용되는 전역변수의 크기가 큼
    (encoding.c 파일 내부의 emsys_h, dsyms_h, symbol_h, esysms_hb_z1, dsyms_hb_z1 변수)
  - HAETAE3, HAETAE5의 경우 Cortex-M4 환경에서 동작하기 위해서는 메모리 최적화가 필요해보임
    (컴파일 단계에서 오버플로우 오류는 발생하지 않으나, 실행단계에서 스택오버플로우 발생)

```
10   #if HAETAE_MODE == 2
11
12   #define M_H 252
13   #define M_HB_Z1 37
14 > static RansEncSymbol esyms_h[M_H] = {…
268 > static RansDecSymbol dsyms_h[M_H] = {…
321 > static uint16_t symbol_h[SCALE] = {…
4693 > static RansEncSymbol esyms_hb_z1[M_HB_Z1] = {…
4732 > static RansDecSymbol dsyms_hb_z1[M_HB_Z1] = {…
4742 > static uint16_t symbol_hb_z1[SCALE] = {…
8194
8195 #elif HAETAE_MODE == 3
8196
8197 #define M_H 252
8198 #define M_HB_Z1 71
8199 > static RansEncSymbol esyms_h[M_H] = {…
8453 > static RansDecSymbol dsyms_h[M_H] = {…
```

HAETAE2에서 많은 크기를 차지하는 static 변수

static 제거

```
10   #if HAETAE_MODE == 2
11
12   #define M_H 252
13   #define M_HB_Z1 37
14 > static RansEncSymbol esyms_h[M_H] = {…
268 > static RansDecSymbol dsyms_h[M_H] = {…
321 > uint16_t symbol_h[SCALE] = {…
93  > static RansEncSymbol esyms_hb_z1[M_HB_Z1] = {…
32  > static RansDecSymbol dsyms_hb_z1[M_HB_Z1] = {…
4742 > uint16_t symbol_hb_z1[SCALE] = {…
8194
```

Overflow 발생

```
/Applications/ArmGNUToolchain/12.2.mpacbti-rel1/arm-none-eabi/b
e-eabi/bin/ld: region `ram' overflowed by 155412 bytes
collect2: error: ld returned 1 exit status
collect2: error: ld returned 1 exit status
collect2: error: ld returned 1 exit status
make: *** [elf/crypto_sign_haetae2_clean_speed.elf] Error 1
make: *** Waiting for unfinished jobs....
```

컴파일 단계에서 HAETAE2의 Overflow 발생

2 경량환경 KpqC 구현 고려사항

2.4 메모리 사용량 관리

- KpqC 알고리즘의 메모리 개선 방향 - SOLMAE 사례
  - 내부적으로 사용하는 Secret Key 구조체 사이즈가 57KB 이상, 입출력에 사용되는 Secret Key 배열은 16KB 정도의 크기 차지(사용 가능한 메모리는 128KB로 Secret Key가 상당 부분을 차지할것임)
  - 경량 환경에서 동작하기 위해서는 Secret Key 내부적으로 사용되는 Secret Key 구조체의 최적화가 필요함

```
// Assuming No Encoding/Decoding in Solmae-(512 or 1024)
#define CRYPTO_SECRETKEYBYTES 16385 // 16384 bytes + header 1 byte
#define CRYPTO_PUBLICKEYBYTES 4097  // 4096
#define CRYPTO_BYTES 8273           // nonce                    bytes
#define CRYPTO_ALGNAME "Solmae-512"

int crypto_sign_keypair(unsigned char *pk, unsigned char *sk);

int crypto_sign(unsigned char *sm, unsigned long long *smlen,
                const unsigned char *m, unsigned long long mlen,
                const unsigned char *sk);

int crypto_sign_open(unsigned char *m, unsigned long long *mlen,
                     const unsigned char *sm, unsigned long long smlen,
                     const unsigned char *pk);
```

입출력으로 사용되는 Secret Key는 16385 Bytes의 크기를 가짐 (api.h)

SOLMAE 512 의 최상위 API 에서 입출력으로 사용되는 Secret Key 배열 길이 선언

```
typedef struct{
  fpr coeffs[SOLMAE_D];
} poly;


typedef struct{
  int8_t f[SOLMAE_D];
  int8_t g[SOLMAE_D];
  int8_t F[SOLMAE_D];
  int8_t G[SOLMAE_D];
  poly b10;   // 4096 byte
  poly b11;
  poly b20;
  poly b21;
  poly GSO_b10; //~b1[0]/<~b1, ~b1>
  poly GSO_b11; //~b1[1]/<~b1, ~b1>
  poly GSO_b20; //~b2[0]/<~b2, ~b2>
  poly GSO_b21; //~b2[1]/<~b2, ~b2>
  poly beta10;
  poly beta11;
  poly beta20;
  poly beta21;
  poly sigma1;
  poly sigma2;
} secret_key;  // @SmartM2M : 512*4 + 8*512*14 = 57344 byte
```

SOLMAE 512에서 내부적으로 사용되는 Secret Key 구조체 선언

2 경량환경 KpqC 구현 고려사항

2.5 경량환경에서 지원하지 않는 자료형 사용

- Cortex-M4는 128 비트 크기의 자료형을 지원하지 않으며, uint32_t 등의 자료형으로 대체 구현 필요
  - (NCC-Sign 레퍼런스 코드) Kratsuba, Toom Cook 4way 구현에 __uint128_t 자료형 사용
  - (GCKSign 레퍼런스 코드) Polynomial 곱셈, NTT 등에 __int128_t 자료형 사용

```
karatsuba_simple(aw1, bw1, w1);
karatsuba_simple(aw2, bw2, w2);
karatsuba_simple(aw3, bw3, w3);
karatsuba_simple(aw4, bw4, w4);
karatsuba_simple(aw5, bw5, w5);
karatsuba_simple(aw6, bw6, w6);
karatsuba_simple(aw7, bw7, w7);

uint64_t mask_coef = 0x00000000FFFFFFFF, si2 = 0;
__uint128_t  mt, t, b;

__uint128_t Q128 = ((__uint128_t)Q << 64) ^ Q, ttmp;
```

NCC-Sign __uint128_t 자료형 사용 코드 일부

```
int64_t mont_mul(int64_t a, int64_t b)
{
    __int128_t t = (__int128_t)a * (__int128_t)b;

    return mont_ivt(t);
}
```

```
int64_t mont_ivt(__int128_t in)
{
    int64_t m = (in * (__int128_t)_MONT_QInvmodbeta);

    __int128_t t = ((__int128_t)m * Q);
    t = in - t;
    t >>= 64;
    t = caddp(t);
    return  csubp((int64_t) t);
}
```

GCK-Sign __int128_t 자료형 사용 코드 일부

2 경량환경 KpqC 구현 고려사항

2.6 KpqC 응용 애플리케이션의 성능/메모리 부하 고려

- TLS, SSH, X.509 등 KpqC가 적용될 다양한 보안 프로토콜, 애플리케이션의 성능/메모리 부하 고려
  - 경량환경에 KpqC 암호 기반 애플리케이션 개발/적용 시, 알고리즘 구현 특징(공개키 길이, 전자서명 크기, Sign/Verify 성능 차이 등)을 고려하여 최적화(Speed/Stack) 구현 필요
- (예시) Falcon의 경우 Key Generation, Sign에 많은 성능 부하가 발생하지만, Verify 성능이 뛰어남
  - 클라이언트(경량 환경)에서 서버/CA의 서명을 검증하는데 효율적 (성능 부하가 적음 → Stack 최적화 구현 적용)

Table 2. Connection characteristics according to 3GPP [1]

Name	Abbrev.	Bandwidth	RTT time
Broadband	BB	1 Mbit	26 ms
LTE machine type communication	LTE-M	1 Mbit	120 ms
Narrowband-IoT	NB-IoT	46 kbit	3 s

PQC 적용 TLS 성능평가 네트워크 환경

	KEX	Auth.	CA	PQC code (%)	CA size (%)	Memory
PQTLS	Kyber	Dilithium	Dilithium	29.0 kB (20.1%)	4.0 kB (2.8%)	58.0 kB
	Kyber	Falcon	Dilithium	34.4 kB (23.0%)	4.0 kB (2.7%)	60.7 kB
	Kyber	Falcon	Falcon	25.8 kB (18.6%)	1.8 kB (1.3%)	56.2 kB
	NTRU	Dilithium	Dilithium	203.4 kB (63.8%)	4.0 kB (1.3%)	56.6 kB
	NTRU	Falcon	Dilithium	208.7 kB (64.4%)	4.0 kB (1.2%)	59.3 kB
	NTRU	Falcon	Falcon	200.1 kB (63.9%)	1.8 kB (0.6%)	54.8 kB
	SABER	Dilithium	Dilithium	31.5 kB (21.5%)	4.0 kB (2.7%)	58.0 kB
	SABER	Falcon	Dilithium	36.8 kB (24.2%)	4.0 kB (2.6%)	60.7 kB
	SABER	Falcon	Falcon	28.2 kB (20.0%)	1.8 kB (1.3%)	56.2 kB

PQC 적용 TLS 수행에 필요한 코드 사이즈, 인증서 크기, 메모리

	bytes transmitted			stored	computation (≈Kcycles)		
Signatures	pubkey	sig	sum	secret	keygen	sign	verify
Dilithium ★	1 312	2 420	3 732	2 528	1 597	4 095	1 572
Falcon ★	897	690	1 587	1 281	163 994	39 014	473
Rainbow †	161 600	66	161 666	103 648	94	907	238
KEMs	pubkey	ciph	sum	secret	keygen	encaps	decaps
Kyber ★	800	768	1 568	1 632	440	539	490
NTRU †	699	699	1 398	953	2 867	565	538
SABER †	672	736	1 408	1 568	352	481	453

★: Scheme was selected for standardization.
†: Scheme was eliminated from the NIST standardization project.

핸드쉐이크에 전송되는 Parameter 크기 및 연산 성능

	KEX	Auth.	CA	Handshake traffic	Handshake time in Mcycles (% of crypto) BB (%)	LTE-M (%)	NB-IoT (%)
PQTLS	Kyber	Dilithium	Dilithium	8.4 kB	19.9 (35.9%)	36.8 (19.5%)	818.1 (0.9%)
	Kyber	Falcon	Dilithium	6.3 kB	15.5 (33.0%)	29.0 (17.6%)	586.4 (0.9%)
	Kyber	Falcon	Falcon	4.5 kB	10.9 (30.1%)	21.0 (15.6%)	464.6 (0.7%)
	NTRU	Dilithium	Dilithium	8.3 kB	24.3 (47.6%)	41.1 (28.1%)	821.3 (1.4%)
	NTRU	Falcon	Dilithium	6.1 kB	19.9 (47.8%)	33.4 (28.5%)	590.6 (1.6%)
	NTRU	Falcon	Falcon	4.3 kB	15.2 (50.3%)	25.4 (30.2%)	468.0 (1.6%)
	SABER	Dilithium	Dilithium	8.3 kB	19.7 (35.2%)	36.6 (19.0%)	817.3 (0.8%)
	SABER	Falcon	Dilithium	6.1 kB	15.3 (32.0%)	28.8 (17.0%)	586.2 (0.8%)
	SABER	Falcon	Falcon	4.3 kB	10.7 (28.5%)	20.9 (14.6%)	464.0 (0.7%)

PQC 적용 TLS 성능평가

Gonzalez, Ruben, and Thom Wiggers. "KEMTLS vs. Post-quantum TLS: Performance on Embedded Systems." *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Cham: Springer Nature Switzerland, 2022.

III 경량환경 KpqC 성능평가

SmartM2M
스마트엠투엠

3 경량환경에서의 성능 평가

3.1 KpqC Round 1 KEM 알고리즘 평가 - 성능

Scheme	Parameter (byte size)	Implementation	Key Generation [cycles]	Encapsulation [cycles]	Decapsulation [cycles]
NTRU+576	sk : 1,728 pk : 864 ct : 864	ntru+_ref	321,405	110,754	163,277
		ntru+_avx2	17,440	14,307	12,445
		m4clean	**800,457**	**578,011**	**764,190**
NTRU+768	sk : 2,304 pk : 1,152 ct : 1,152	ntru_ref	313,669	145,658	227,028
		ntru+_avx2	16,032	17,514	15,848
		m4clean	**899,443**	**697,521**	**947,030**
NTRU+864	sk : 2,592 pk : 1,296 ct : 1,296	ntru_ref	339,912	169,634	262,017
		ntru+_avx2	14,068	19,293	17,671
		m4clean	**1,041,375**	**797,486**	**1,076,691**
NTRU+1152	sk : 3,456 pk : 1,728 ct : 1,728	ref	905,131	230,448	348,076
		ntru+_avx2	42,993	25,592	24,063
		m4clean	**1,364,431**	**1,072,541**	**1,457,290**
SMAUG128	sk : 174 pk : 672 ct : 768	smaug_ref	**73,584**	**81,684**	**88,920**
		m4clean	**1,041,572**	**1,263,455**	**407,661**
SMAUG192	sk : 185 pk : 992 ct : 1024	smaug_ref	**106,956**	**115,128**	**124,812**
		m4clean	**1,629,668**	**1,895,641**	**467,935**
SMAUG256	sk : 182 pk : 1,632 ct : 1,536	smaug_ref	**191,268**	**200,520**	**210,240**
		m4clean	**2,844,016**	**3,062,469**	**539,045**

- ntru+_ref : Intel Core i8-8900K(3700HMz)
- ntru+_avx2 : Intel Core i8-8900K(3700HMz) with AVX2
- smaug_ref : AMD Ryzen 3700X(3,589MHz) with TurboBoost and hyperthreading disabled
- m4clean : 저자들이 제공한 Code를 일부 수정하여 Cortex-M4에서 실행

3 경량환경에서의 성능 평가

3.2 KpqC Round 1 KEM 알고리즘 평가 - 메모리 사용량

Scheme	Parameter (byte size)	Implementation	Key Generation [bytes]	Encapsulation [bytes]	Decapsulation [bytes]
NTRU+576	sk : 1,728 pk : 864 ct : 864	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	6,948	6,604	12,704
NTRU+768	sk : 2,304 pk : 1,152 ct : 1,152	ntru_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	8,964	8,500	16,664
NTRU+864	sk : 2,592 pk : 1,296 ct : 1,296	ntru_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	9,972	9,452	18,648
NTRU+1152	sk : 3,456 pk : 1,728 ct : 1,728	ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	13,004	12,300	24,592
SMAUG128	sk : 174 pk : 672 ct : 768	smaug_ref	-	-	-
		m4clean	5,156	6,348	4,472
SMAUG192	sk : 185 pk : 992 ct : 1024	smaug_ref	-	-	-
		m4clean	8,244	10,460	5,520
SMAUG256	sk : 182 pk : 1,632 ct : 1,536	smaug_ref	-	-	-
		m4clean	18,272	21,688	7,516

- ntru+_ref : Intel Core i8-8900K(3700MHz)
- ntru+_avx2 : Intel Core i8-8900K(3700MHz) with AVX2
- smaug_ref : AMD Ryzen 3700X(3589MHz) with TurboBoost and hyperthreading disabled
- m4clean : 저자들이 제공한 Code를 일부 수정하여 Cortex-M4에서 실행

- 로 표시된 부분은 저자들이 제시한 메모리 사용량 측정 결과가 없음을 뜻함

3 경량환경에서의 성능 평가

3.3 KpqC Round 1 Signature 알고리즘 평가 - 성능

Scheme	Parameter (byte size)		Implementation	Key Generation [cycles]	Sign [cycles]	Verify [cycles]
GCKSIGN2	sk : pk : sig :	288 1,760 1,952	gcksign_ref	184,000	1,062,000	237,000
			m4clean	3,185,429	13,473,226	3,601,819
GCKSIGN3	sk : pk : sig :	288 1,952 2,080	gcksign_ref	202,000	1,240,000	253,000
			m4clean	3,180,656	29,208,406	3,585,449
GCKSIGN5	sk : pk : sig :	544 3,040 2,080	gcksign_ref	265,000	1,421,000	373,000
			m4clean	4,809,045	21,571,844	5,786,047
HAETAE120	sk : pk : sig :	1,376 992 1,463	haetae_ref	1,832,973	8,903,852	388,377
			m4clean	7,528,583	35,536,313	1,473,756
HAETAE180	sk : pk : sig :	2,080 1,472 2,337	haetae_ref	3,464,004	11,763,246	719,400
			m4clean	-	-	-
HAETAE260	sk : pk : sig :	2,720 2,080 2,908	haetae_ref	2,129,737	12,459,046	914,336
			m4clean	-	-	-
SOLMAE512	sk : pk : sig :	16,385 896 666	solmae-ref	27,000,000	387,000	40,000
			m4clean	1,228,875,057	-	-
SOLMAE1024	sk : pk : sig :	32,769 1,792 1,375	solmae-ref	65,000,000	775,000	84,000
			m4clean	-	-	-
NCCSIGN1 (Concrete Param)	sk : pk : sig :	2,400 1,400 2,529	nccsign_ref	1,257,562	16,174,808	2,444,616
			m4clean	-	-	-
NCCSIGN3 (Concrete Param)	sk : pk : sig :	3,377 2,037 3,678	nccsign_ref	2,386,408	28,184,328	4,765,774
			m4clean	-	-	-
NCCSIGN5 (Concrete Param)	sk : pk : sig :	4,713 2,462 5,135	nccsign_ref	4,202,722	49,062,056	8,342,102
			m4clean	-	-	-
NCCSIGN1 (Conservative Param)	sk : pk : sig :	2,800 1,984 3,186	nccsign_ref	1,727,508	11,768,076	3,400,702
			m4clean	-	-	-
NCCSIGN3 (Conservative Param)	sk : pk : sig :	3,914 2,443 4,251	nccsign_ref	2,965,942	20,816,964	5,876,246
			m4clean	-	-	-
NCCSIGN5 (Conservative Param)	sk : pk : sig :	4,940 3,091 5,385	nccsign_ref	4,700,228	42,227,652	9,324,876
			m4clean	-	-	-

- 로 표시된 부분은 성능평가 진행중임을 뜻함

- gcksign_ref : Intel i7-8700K
- haetae_ref : Intel i7-10800K with TurboBoost
- solmae_ref : Intel i7-8550U (1.80GHz)
- nccsign_ref : Intel i7-12700K (3.60GHz)
- m4clean : Cortex-M4에서 실행

3 경량환경에서의 성능 평가

3.4 KpqC Round 1 Signature 알고리즘 평가 - 메모리 사용량 (Cortex-M4 환경)

Scheme	Parameter (byte size)		Implementation	Key Generation [bytes]	Sign [bytes]	Verify [bytes]
GCKSIGN2	sk : pk : sig :	288 1,760 1,952	m4clean	19,892	40,132	-
GCKSIGN3	sk : pk : sig :	288 1,952 2,080	m4clean	-	-	-
GCKSIGN5	sk : pk : sig :	544 3,040 2,080	m4clean	-	-	-
HAETAE120	sk : pk : sig :	1,376 992 1,463	m4clean	26,116	78,084	30,780
HAETAE180	sk : pk : sig :	2,080 1,472 2,337	m4clean	-	-	-
HAETAE260	sk : pk : sig :	2,720 2,080 2,908	m4clean	-	-	-
SOLMAE512	sk : pk : sig :	16,385 896 666	m4clean	-	-	-
SOLMAE1024	sk : pk : sig :	32,769 1,792 1,375	m4clean	-	-	-
NCCSIGN1 (Concrete Param)	sk : pk : sig :	2,400 1,400 2,529	m4clean	-	-	-
NCCSIGN3 (Concrete Param)	sk : pk : sig :	3,377 2,037 3,678	m4clean	-	-	-
NCCSIGN5 (Concrete Param)	sk : pk : sig :	4,713 2,462 5,135	m4clean	-	-	-
NCCSIGN1 (Conservative Param)	sk : pk : sig :	2,800 1,984 3,186	m4clean	-	-	-
NCCSIGN3 (Conservative Param)	sk : pk : sig :	3,914 2,443 4,251	m4clean	-	-	-
NCCSIGN5 (Conservative Param)	sk : pk : sig :	4,940 3,091 5,385	m4clean	-	-	-

- 로 표시된 부분은 메모리 사용량 평가 진행중임을 뜻함

감사합니다

부록

PQM4에 적용된 Kyber 최적화 기술 동향

- 현재 PQM4에 구현된 Kyber는 Improved Plantard Arithmetic[1]이 적용됨('22년10월)
  - '22년 2월 : Faster Kyber and Dilithium on the Cotex-M4[2] 적용
  - '20년 10월 : CortexM4 optimization for {R, M} LWE schemes[3] 적용
- Improved Plantard Arithmetic은 메모리 사용량에 큰 차이 없이 성능 향상 가능
  - Kyber의 Speed/Stack 구현에 공통적으로 적용된 상태

업데이트 날짜	적용 논문	최적화 분류	Key Generation		Encapsulation		Decapsulation	
			Speed[cycles]	Stack[bytes]	Speed[cycles]	Stack[bytes]	Speed[cycles]	Stack[bytes]
'22년 10월	Improved Plantard Arithmetic for lattice-based cryptography[1]	Speed	434,438	**4,320**	530,469	**5,424**	476,712	**5,432**
		Stack	433,718	**2,248**	531,676	**2,336**	478,166	**2,352**
'22년 2월	Faster Kyber and Dilithium on the Cotex-M4[2]	Speed	441,489	**4,272**	540,354	**5,376**	491,034	**5,384**
		Stack	441,712	**2,212**	542,251	**2,300**	493,434	**2,316**
'20년 10월	Cotex-M4 Optimization for {R, M} LWE sceh	-	468,578	**2,404**	594,543	**2,484**	552,657	**2,508**

1) [IPA22] - Huang, Junhao, et al. "Improved Plantard arithmetic for lattice-based cryptography." IACR Transactions on Cryptographic Hardware and Embedded Systems 2022.4 (2022)
2) [AHKS22] - Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In Giuseppe Ateniese and Daniele Venturi, editors, Applied Cryptography and Network Security - 20th International Conference, ACNS 2022
3) [ABCG20] - Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R, M} LWE schemes. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(3):336-357, 2020.

부록

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- 최적화 관련 논문 - Improved Plantard Arithmetic for lattice-based cryptography[1)]
  - 기존의 Plantard Arithmetic[2)]을 개선하여 PQC에 적용한 것으로, Singed integers 입력 허용, Lazy Reduction 효율 향상 등에서 개선됨

Table 2: Cycle counts (cc) and stack usage (Bytes) for KeyGen, Encaps, and Decaps on Cortex-M4. k is the dimension of the underlying Module-LWE problem for Kyber. The first row of each entry indicates the cycle count and the second row refers to stack usage.

Scheme	Implementation	KeyGen			Encaps			Decaps		
		$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$
Kyber	[ABCG20][3)]	454k	741k	1 177k	548k	893k	1 367k	506k	832k	1 287k
		2 464	2 696	3 584	2 168	2 640	3 208	2 184	2 656	3 224
	This work[a]	446k	729k	1 162k	542k	885k	1 357k	497k	818k	1 270k
		2 464	2 696	3 584	2 168	2 640	3 208	2 184	2 656	3 224
	Stack[AHKS22][4)]	439k	717k	1 139k	534k	871k	1 329k	484k	797k	1 233k
		2 608	3 056	3 576	2 160	2 660	3 236	2 176	2 676	3 252
	Speed[AHKS22]	438k	711k	1 129k	531k	864k	1 316k	479k	787k	1 217k
		4 268	6 732	7 748	5 252	6 284	7 292	5 260	6 308	7 300
	This work[b]	430k	702k	1 119k	526k	861k	1 314k	472k	780k	1 211k
		2 608	3 056	3 576	2 160	2 660	3 236	2 176	2 676	3 252

[a] Implementation based on [ABCG20], [b] Implementation based on the stack-friendly code of [AHKS22].

Improved Plantard Arithmetic 적용 Kyber 성능 (Cortex-M4 환경)

1) Huang, Junhao, et al. "Improved Plantard arithmetic for lattice-based cryptography." IACR Transactions on Cryptographic Hardware and Embedded Systems 2022.4 (2022): 614-636.
2) Plantard, Thomas. "Efficient word size modular arithmetic." IEEE Transactions on Emerging Topics in Computing 9.3 (2021): 1506-1518.
3) Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. CortexM4 optimizations for {R, M} LWE schemes. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(3):336-357, 2020.
4) Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In Giuseppe Ateniese and Daniele Venturi, editors, Applied Cryptography and Network Security - 20th International Conference, ACNS 2022

부록

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- **Plantard Arithmetic 개요**
  - 1-word 크기 이하의 Modulus를 사용할때의 Multiplication 최적화 기법
    → 기존의 Barret, Montgomery 등의 기법은 Modulus의 크기가 1-word 작을때 생각보다 성능이 좋지 않음
  - NTT 적용 관점에서 Barret, Montgomery Modular Multiplication 보다 좋은 성능을 보임

TABLE 4. Comparison between our and Barrett's modular arithmetic.

	$\log_2 P$	New (8) (cycles)	Barrett (2)
EXP (Modular Exponentiation)	30	128.9	168.0 (+30%)
	31	134.5	171.0 (+27%)
	32	137.2	n/a
EVL (Polynomial Evaluation)	30	330.2	473.9 (+44%)
	31	330.1	461.6 (+40%)
	32	329.5	n/a
NTT	30	189.8	229.4 (+21%)
	31	188.3	228.6 (+22%)
	32	189.0	n/a
RNS (Residue Number System)	30	944.9	1723 (+82%)
	31	946.9	1753 (+85%)
	32	954.4	n/a

Plantard, Barrett 성능 비교 결과(intel i7 환경)

TABLE 2. Comparison between our and Montgomery's modular arithmetic.

	$\log_2 P$	New (8) (cycles)	Montgomery (1)	
			w corr.	wo corr.
EXP	30	128.9	178.3 (+38%)	126.5 (−1.9%)
	31	134.5	189.8 (+41%)	n/a
	32	137.2	169.6 (+24%)	n/a
EVL	30	330.2	450.4 (+36%)	357.4 (+8.2%)
	31	330.1	450.1 (+36%)	n/a
	32	329.5	450.0 (+36%)	n/a
NTT	30	189.8	204.8 (+7.9%)	202.2 (+6.6%)
	31	188.3	210.9 (+12%)	n/a
	32	189.0	210.9 (+12%)	n/a
RNS	30	944.9	1366 (+44%)	1187 (+26%)
	31	946.9	1390 (+47%)	n/a
	32	954.4	1419 (+48%)	n/a

Plantard, Montgomery 성능 비교 결과(intel i7환경)

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- **Plantard Arithmetic 알고리즘**
  - Notation : $[X]_k := X \bmod 2^k$, $[X]^k := \left\lfloor \frac{X}{2^k} \right\rfloor$
  - Statement : $\phi = \frac{1+\sqrt{5}}{2}$, $P < \frac{2^n}{\phi}$

Algorithm 8. New Modular Multiplication

Input: A, B, P, R, n with $0 \le A, B \le P$ and $R = P^{-1} \bmod 2^{2n}$
Output: C with $0 \le C < P$ and $C = AB(-2^{-2n}) \bmod P$
begin
  $C \leftarrow [([[ABR]_{2n}]^n + 1)P]^n$
  if $C = P$ **then**
    return 0
  end
  return C
end

$$\left\lfloor \frac{\left(\left\lfloor \frac{(ABR \bmod 2^{2n})}{2^n} \right\rfloor + 1\right)P}{2^n} \right\rfloor$$

ii) Second, we check that $C \equiv AB(-2^{-n}) \bmod P$. As P is odd, we know that there exist a $Q = ABP^{-1} \bmod 2^{2n}$. Consequently, we have

$$\begin{aligned} QP - AB &\equiv (ABP^{-1})P - AB \bmod 2^{2n} \\ &\equiv AB - AB \bmod 2^{2n} \\ &\equiv 0 \bmod 2^{2n}. \end{aligned}$$

We obtain that $QP - AB$ is divisible by 2^{2n} and therefore that

$$AB(-2^{-2n}) \bmod P = \frac{QP - AB}{2^{2n}}.$$

We will note $Q_1 = \lfloor \frac{Q}{2^n} \rfloor$ and $Q_0 = Q - Q_1 2^n$ with $0 \le Q_0 < 2^n$.

Now, we analyse $P2^n - Q_0P + AB$, knowing that $0 \le A, B \le P$, then we obtain $P2^n - Q_0P + AB \le P2^n + P^2$. As by Theorem statement, we have $P < \frac{2^n}{\phi}$ with $\phi = \frac{1+\sqrt{5}}{2}$. Consequently, we obtain

$$\begin{aligned} P2^n - Q_0P + AB &< 2^n \frac{2^n}{\phi} + \left(\frac{2^n}{\phi}\right)^2 \\ &= \frac{2^{2n}}{\phi} + \frac{2^{2n}}{\phi^2} \\ &= 2^{2n} \frac{\phi + 1}{\phi^2} = 2^{2n} \end{aligned}$$

Furthermore, as $Q_0 < 2^n$ and $A, B \ge 0$, we obtain as well that $P2^n - Q_0P + AB > 0$, we, consequently, obtain that

$$0 < \frac{P2^n - Q_0P + AB}{2^{2n}} < 1.$$

Therefore and because $QP - AB$ is divisible by 2^{2n},

$$\begin{aligned} lI\frac{QP - AB}{2^{2n}} &= \left\lfloor \frac{QP - AB}{2^{2n}} + \frac{P2^n - Q_0P + AB}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{QP - AB + P2^n - Q_0P + AB}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q - Q_0 + 2^n)P}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q_1 2^n + 2^n)P}{2^{2n}} \right\rfloor \\ &= \left\lfloor \frac{(Q_1 + 1)P}{2^n} \right\rfloor. \end{aligned}$$

$$Q_1 = \left\lfloor \frac{Q}{2^n} \right\rfloor = \left\lfloor \frac{(ABP^{-1} \bmod 2^{2n})}{2^n} \right\rfloor$$

$$\begin{aligned} lIAB(-2^{-2n}) \bmod P &\equiv \frac{QP - AB}{2^{2n}} \\ &\equiv \left\lfloor \frac{(Q_1 + 1)P}{2^n} \right\rfloor \\ &\equiv \left\lfloor \frac{\left(\left\lfloor \frac{(ABP^{-1} \bmod 2^{2n})}{2^n} \right\rfloor + 1\right)P}{2^n} \right\rfloor \end{aligned}$$

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

▪ Plantard Arithmetic 알고리즘과 기존 알고리즘의 비교

지표	설명	지표	설명
$\log_2(P)$	the size of possible moduli	IF	the number of time an instruction if is used(includes comparison)
$\log_2(\#P)$	the number of possible moduli	SFT	the number of shift
REP	the need of a specific representation indicated by True, or False	MSK	the number of mask used
MUL	the number of word size multiplication	CNT	the number of constant used by each algorithm
ADD	the number of word size addition/subtraction	REG	the number of variable used by each algorithm
TOT	the total number of operations required to perform a modular multiplication		

Algorithm	$\log_2(P)$	$\log_2(\#P)$	REP	MUL	ADD	IF	SFT	MSK	TOT	CNT	REG	TOT
New method (8) w/wo corr.	n-0.694	n-1.694	T	3	1	1/0	2	0	7/6	2	1	3
New (8) for a constant w/wo corr.	n-0.694	n-1.694	T	2	1	1/0	2	0	6/5	1	1	2
Montgomery (1)	n-0.694	n-1.694	T	3	2	1	1	1	8	2	2	4
Montgomery (1) wo corr.	n-2	n-3	T	3	1	0	1	1	6	2	2	4
Barrett (2)	n-1	n-1	F	3	2	2	2	0	9	2	2	4
NFLlib (6)	n-1	n-3	F	3	3	1	3	1	11	2	2	4
Pseudo-Mersenne (4)	n	$\frac{n}{2}$	F	3	3	1	2	2	11	2	2	4
Pseudo-Mersenne (4) wo corr.	n-1	$\frac{n-4}{2}$	F	3	2	0	2	2	9	2	2	4
Montgomery-Friendly (7)	n	$\frac{n}{2}$	T	3	3	1	2	2	11	2	2	4
Montgomery-Friendly (7) wo corr.	n-1	$\frac{n-3}{2}$	T	3	2	0	2	2	9	2	2	4
Generalized Mersenne (5)	n	$\frac{\log_2(n)-2}{2}$	F	1	5	1	4	2	11	1	2	3
Generalized Mersenne (5) wo corr	n-1	$\frac{\log_2(n)-6}{2}$	F	1	4	0	4	2	9	1	2	3
Mersenne (3)	n	0	F	1	2	1	1	1	6	0	2	2
Forced Mersenne (3)	n	0	F	1	2	0	2	2	7	0	2	2

부록

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- **Improved Plantard Arithmetic for lattice-based cryptography**
  - 기존 Plantard Arithmetic은 Unsigned integers [0, q]만 허용하였으나, Singed Integer $(-\frac{q}{2}, \frac{q}{2})$ 를 허용함
    → Signed Integer를 지원하면, Butterfly Computation 수행 시 연산 결과의 양수 보장을 위해 필요했던 modulus q의 배수를 더하는 연산이 필요없으며, 이를 통해 연산 효율을 향상 가능
  - NTT(CT butterflies), INTT(GS butterflies) 과정에서 Lazy Reduction 성능 개선
  - Lazy Reduction 성능은 입력 범위가 크고 출력 coefficient가 작을수록 향상됨(Modular Reduction 횟수 감소)
    → 기존의 Montgomery Reduction 보다 제안 기법은 입력 범위가 크고, 출력 coefficient 크기는 절반정도임
  - Kyber의 경우 기존 NTT/INTT 구현[1,2] 대비 10% 이상의 성능이 향상됨

Table 1: Cycle counts for the core polynomial arithmetic in Kyber and NTTRU on Cortex-M4, i.e., NTT, INTT, base multiplication, and base inversion.

Scheme	Implementation	NTT	INTT	Base Mult	Base Inv
Kyber	[ABCG20]	6 822	6 951	2 291	-
	This work[a]	5 441	5 775	2 421	-
	Speed-up	20.24%	16.92%	-5.67%	-
	Stack[AHKS22]	5 967	5 917	2 293	-
	Speed[AHKS22]	5 967	5 471	1 202	-
	This work[b]	4 474	4 684/4 819/4 854	2 422	-
	Speed-up (stack)	25.02%	20.84%/18.56%/17.97%	-5.58%	-
	Speed-up (speed)	25.02%	14.38%/11.92%/11.28%	-101.41%	-
NTTRU	[LS19]	102 881	97 986	44 703	100 249
	This work	17 274	20 931	10 550	40 763
	Speed-up	83.21%	78.64%	76.40%	59.34%

[a] Implementation based on [ABCG20], [b] Implementation based on the stack-friendly code of [AHKS22].

1) Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. CortexM4 optimizations for {R, M} LWE schemes. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(3):336-357, 2020.
2) Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In Giuseppe Ateniese and Daniele Venturi, editors, Applied Cryptography and Network Security - 20th International Conference, ACNS 2022

부록

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- **Improved Plantard Multiplication 제안 기법**

Algorithm 10 Improved Plantard multiplication

Input: Two signed integers $a, b \in [-q2^{\alpha}, q2^{\alpha}], q < 2^{l-\alpha-1}, q' = q^{-1} \bmod^{\pm} 2^{2l}$
Output: $r = ab(-2^{-2l}) \bmod^{\pm} q$ where $r \in (-\frac{q}{2}, \frac{q}{2})$

1: $r = \left[\left([[abq']_{2l}]^{l} + 2^{\alpha}\right) q\right]^{l}$
2: **return** r

a, b : Integer
ab : a와 b의 곱셈 연산
q : Odd modulus value
l' : $q < 2^{l-\alpha-1}$을 만족하는 최소 워드 크기(α는 작은 정수)
$2l$: 2^{2l}로 modulo 연산을 하기 위해 사용되는 2의 거듭제곱
r : 곱셈 연산의 최종 결과
α : 최소 워드 크기 l'을 계산하기 위해 사용되는 작은 정수

- Improved Plantard multiplication는 격자기반 양자내성암호에서 Constant time을 만족하기 위한 곱셈 연산 기법임
- 나눗셈 연산 대신 Constant time 곱셈 및 Shift Operation을 사용하여 곱셈 연산을 수행

- l은 $q<2^{(l-\alpha-a)}$ 을 만족하는 최소 워드 크기, α는 작은 정수이므로 특정 입력 크기에 대해 빠른 고속 연산이 가능하며 Cortex-M4 SIMD 명령어를 통해 2 cycle로 구현이 가능함

부록

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- The 2-cycle Improved Plantard Multiplication by a constant on Cortex-M4

Algorithm 11 The 2-cycle improved Plantard multiplication by a constant on Cortex-M4

Input: An l-bit signed integer $a \in [-2^{l-1}, 2^{l-1})$, a precomputed $2l$-bit integer bq' where b is a constant and $q' = q^{-1} \bmod^{\pm} 2^{2l}$

Output: $r_{top} = ab(-2^{-2l}) \bmod^{\pm} q, r_{top} \in (-\frac{q}{2}, \frac{q}{2})$

1: $bq' \leftarrow bq^{-1} \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **smulwb** r, bq', a ▷ $r \leftarrow [[abq']_{2l}]^l$
3: **smlabb** $r, r, q, q2^{\alpha}$ ▷ $r_{\text{top}} \leftarrow [q[r]_l + q2^{\alpha}]^l$
4: **return** r_{top}

- Cortex M4 SIMD 구현 방법으로 smulwb, smlabb 명령어는 1cycle을 가짐
- 효율적인 연산을 위해 벡터 및 행렬연산에 효율적인 SIMD 명령어를 활용한 Improved Plantard Multiplication 구현 기법을 제안함
  - SMULWB r, bq', a : bq'과 a를 곱한 후 하위 16bit를 확장하여 32bit 결과를 r 레지스터에 저장
  - SMLABB $r, r, q, q^{2\alpha}$: r 레지스터에 저장된 값을 가져와 q와 $q^{2\alpha}$를 곱한 뒤 누적

		Cycle counts	
CLASS	INSTRUCTION	CORTEX-M3	Cortex-M4
Multiplication	MUL, MLA	1 - 2	1
	MULS, MLAS	1 - 2	1
	SMULL, UMULL, SMLAL, UMLAL	5 - 7	1
	SMULBB, SMULBT, SMULTB, SMULTT	n/a	1
	SMLABB, SMLBT, SMLATB, SMLATT	n/a	1
	SMULWB, SMULWT, SMLAWB, SMLAWT	n/a	1

다항식 곱셈과 Modular reduction 알고리즘 연산 최적화

- The 2-cycle improved Plantard reduction on Cortex-M4

Algorithm 13 The 2-cycle improved Plantard reduction on Cortex-M4

Input: A $2l$-bit signed integer $c \in [-q^2 2^{2\alpha}, q^2 2^{2\alpha}]$
Output: $r_{top} = c(-2^{-2l}) \bmod^{\pm} q, r_{top} \in (-\frac{q}{2}, \frac{q}{2})$

1: $q' \leftarrow q^{-1} \bmod^{\pm} 2^{2l}$ ▷ precomputed
2: **mul** r, c, q'
3: **smlatb** $r, r, q, q2^{\alpha}$
4: **return** r_{top}

- 제안 기법은 Cortex-M4 상에서 16bit 모듈러 상의 다항식 역원을 계산하기 위한 기법
- Improved Plantard Reduction은 SIMD 명령어를 통해 Inverse 및 Multiplication 연산을 수행하여 임베디드 환경 상에서 효율적인 연산이 가능함
  - mul r, c, q' : $r = c * q$
  - smlatb $r, r, q, q^{2^{\alpha}}$: $r = r + (q * q^{2^{\alpha}})$

		Cycle counts	
CLASS	INSTRUCTION	CORTEX-M3	Cortex-M4
Multiplication	MUL, MLA	1 - 2	1
	MULS, MLAS	1 - 2	1
	SMULL, UMULL, SMLAL, UMLAL	5 - 7	1
	SMULBB, SMULBT, SMULTB, SMULTT	n/a	1
	SMLABB, SMLBT, SMLATB, SMLATT	n/a	1
	SMULWB, SMULWT, SMLAWB, SMLAWT	n/a	1