SmartM2M
스마트엠투엠

KpqC 알고리즘 구현특성 분석

- KpqC 7차 워크숍 -

(주)스마트엠투엠
김호원

2023.11.14

CONTENTS

KpqC 암호 경량 프로세서 구현 특성 분석

Ⅰ 경량환경 PQC 성능평가

SmartM2M
스마트엠투엠

1 경량 환경에서의 KpqC 구현 및 성능평가 - Kpqm4

1.1 KpqC 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
6	KEM	Lattice	SMAUG	Junghee Cheon
7	KEM	Lattice	TiGER	Seunghwan Park
8	전자서명	MPC-in-the-Head	AIMer	Jooyoung Lee
9	전자서명	Code	Enhanced pqsigRM	Jongseon No
10	전자서명	Isogeny	FIBS	Suhri Kim
11	전자서명	Lattice	GCKSign	Jonghwan Park
12	전자서명	Lattice	HAETAE	Junghee Cheon
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 (실제 가용 용량 128 KB)

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)

SmartM2M 스마트엠투엠

1 경량 환경에서의 KpqC 구현 및 성능평가 - Kpqm4

1.1 KpqC 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

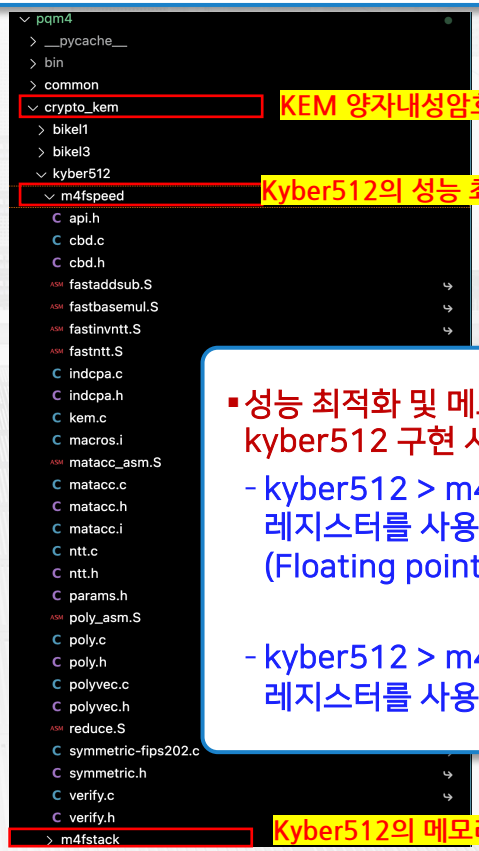
구현 수준	설명
clean	NIST pqC : 제출 코드에서 외부 라이브러리 의존성을 삭제하고 Cortex-M4 에서 동작가능하도록 수정한 코드
opt	NIST pqC : NIST에 제출된 reference 코드의 C언어 레벨 최적 구현 버전
m4	NIST pqC : Cortex-M4 대상 어셈블리 레벨 최적 구현 버전
m4f	NIST pqC : Cortex-M4 대상 어셈블리 레벨 (floating-point 레지스터 활용) 최적 구현 버전

〈 사례: PQM4에서 제공하는 다섯가지 NIST PQC 구현 수준 〉

PQM4에 대응하는 KPQM4 구현

구현 수준	설명
clean	KpqC : 제출 코드에서 외부 라이브러리 의존성을 삭제하고 Cortex-M4 에서 동작가능하도록 수정한 코드

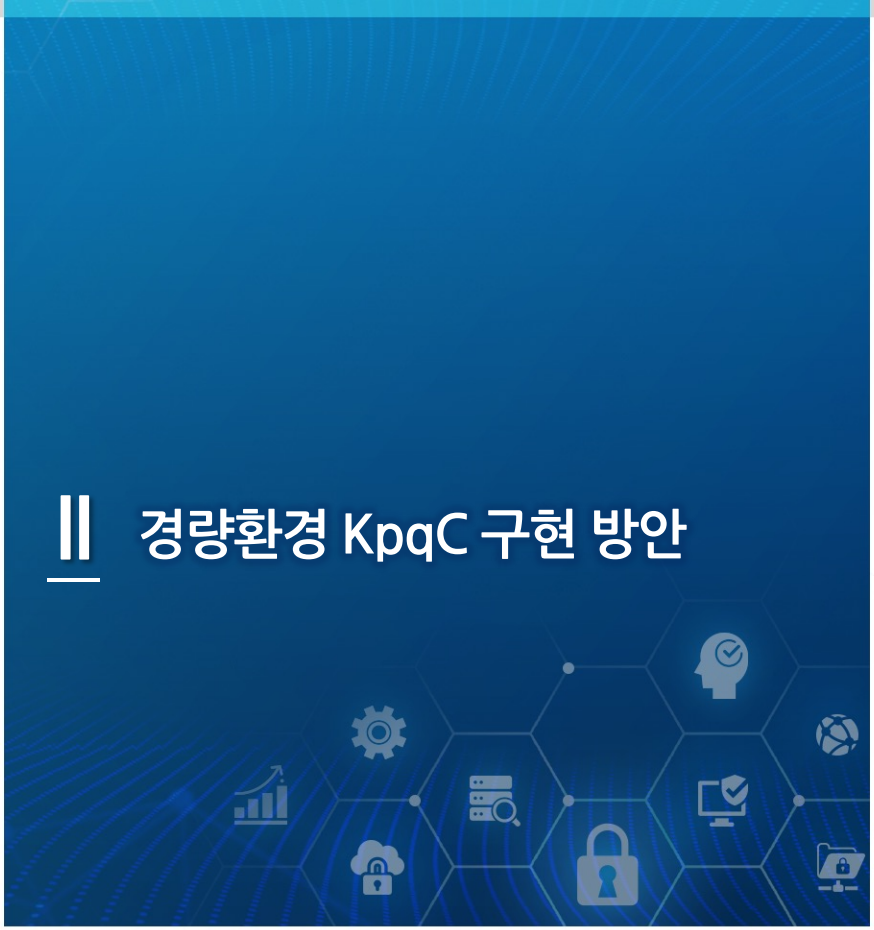
〈 본 연구결과의 최종 결과물 : KPQM4〉



- 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

SmartM2M 스마트엠투엠

Ⅱ 경량환경 KpqC 구현 방안

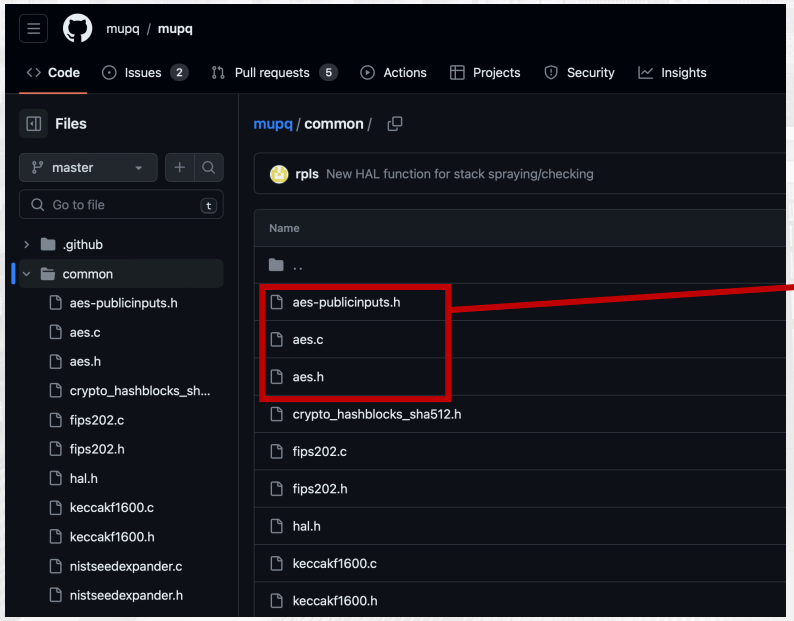


SmartM2M
스마트엠투엠

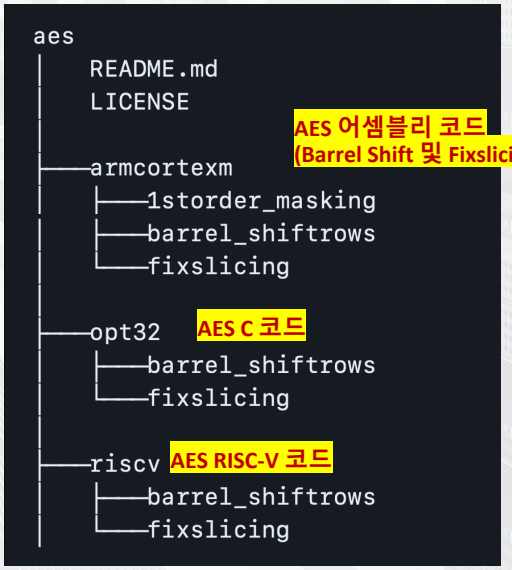
1 mupq 오픈소스 적용

1.1 pqm4 오픈소스 프로젝트와 동일한 시험 환경 구축

- KpqC 공모 제출 코드의 Cortex M4 동작을 위한 외부 의존성 제거(mupq 오픈소스 프로젝트 적용)
  - mupq 오픈소스 프로젝트는 pqclean 코드를 기반으로 만들어진 오픈소스 프로젝트
  - OpenSSL 종속성이 없는 Cortex M4 환경에 적합한 AES 및 SHA-2, SHA-3 코드를 제공함
  - KpqC 1 Round 제출물에서는 AVX2/AES-NI 명령어의 사용, OpenSSL의 종속 함수 등 Cortex-M4에서 지원하지 않는 코드가 다수 존재 → mupq 라이브러리로 대체
  - mupq 라이브러리 적용으로 Cotex-M4 상에서의 동작 및 객관적 성능 평가 가능
  - 어셈블리 및 bitslicing, t-table 최적화 구현 AES 코드를 KpqC에 적용하여 Cortex-M4 환경에서의 성능향상 수행



Github mupq 오픈소스 프로젝트



mupq에서 사용하고 있는 AES 최적화 코드
(https://github.com/aadomn/aes)

1 효율적인 메모리 사용 방안

1.2 동적 할당을 통한 메모리 사용량 최적화 (1/2)- 사례#1. SOLMAE512

- 저전력, 저사양 디바이스에서는 작은 크기의 메모리를 가지고 있음
  - 많은 메모리를 사용하는 경우 임베디드 보드에서는 정상 동작이 불가할 수 있음
  - 이에, 사용이 완료된 변수는 메모리에서 제거할 수 있도록 동적 할당을 하는 것이 필요함
  - 동적할당을 사용할 경우 Memory Leak에 주의해야 하며, 일부 성능상의 오버헤드도 발생할 수 있음
  - Heap 영역과 Stack 영역의 적절한 Tradeoff를 고려하지 않는 경우, Heap 영역이 Stack 영역 침범할 수 있음

암호 알고리즘 연산에서 사용되는 중간값

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

암호 알고리즘 연산에서 사용되는 중간값

Secret Key 구조체 자료형의 크기 = 58KB
-> 일부 변수는 동적할당하여 약 40KB 크기 사용

```
void sign(const uint8_t *m, const secret_key *sk, signature *s)
{
  int trials = 0;
  // @SmartM2M [kj] poly, m_r 동적할당으로 변경

  // poly c1, c2, v0, v1;

  poly c1, c2;
  poly* v0, * v1;

  v0 = (poly*)malloc(sizeof(poly));
  v1 = (poly*)malloc(sizeof(poly));

  // uint8_t m_r[MSG_BYTES + SOLMAE_K / 8];
  uint8_t* m_r = (uint8_t*)malloc(sizeof(uint8_t) * (MSG_BYTES + SOLMAE_K / 8));
```

일부 변수만 동적할당 적용

```
  // @SmartM2M [kj] 중간 계산에 쓰이는 메모리 할당
  skstruct.GSO_b10 = (poly*)malloc(sizeof(poly));
  skstruct.GSO_b11 = (poly*)malloc(sizeof(poly));
  skstruct.GSO_b20 = (poly*)malloc(sizeof(poly));
  skstruct.GSO_b21 = (poly*)malloc(sizeof(poly));

  precomp_GSO(&skstruct);
  precomp_sigma(&skstruct);
  precomp_beta_hat(&skstruct);

  free(skstruct.GSO_b10); skstruct.GSO_b10 = NULL;
```

사용 완료 후, pointer 변수에
NULL을 할당하여, dangling pointer 가능성 없앰

SmartM2M 스마트엠투엠

1 효율적인 메모리 사용 방안

1.2 동적 할당을 통한 메모리 사용량 최적화 (2/3) - 사례#2. NCCSIGN 1

- poly 구조체 1개는 약 9KB의 크기를 가짐
  - NCCSIGN에서는 다수의 polynomial 연산이 수행되며, 이를 위해 필요한 구조체 변수가 다수 사용됨
  - poly 구조체 변수 1개는 약 9KB의 크기를 가짐에 따라 경량 프로세서 환경에서는 부담이 됨
  - 이에, 사용이 완료된 poly 구조체에 대해서는 최적 동적할당해제를 통해, 최대한 메모리 공간을 확보해야함

```
typedef struct {
  int32_t coeffs[NR<<1];
} poly;
```

NCCSIGN1에서 poly 구조체 크기는 약 9KB

```
    w0 = malloc(sizeof(poly));
    w1 = malloc(sizeof(poly));
    mw1 = w1;
    z = malloc(sizeof(poly));
    mz  = z;
rej:
    y = malloc(sizeof(poly));
    poly_uniform_gamma1(y, rho, nonce++);

    poly_tomont(mz, y);
    poly_mul(mw1, mz, &mat);
    poly_frommont(w1, mw1);

    poly_decompose(w1, w0, w1);

    polyw1_pack(sig, w1);

    shake256_init(&state);
    shake256_absorb(&state, mu, CRHBYTES);
    shake256_absorb(&state, sig, POLYW1_PACKEDBYTES);
    shake256_finalize(&state);
    shake256_squeeze(sig, SEEDBYTES, &state);
    poly_challenge(&cp, sig);

    poly_tomont(mcp, &cp);
    poly_mul(mz, mcp, ms1);
    poly_frommont(z, mz);

    poly_add(z, z, y);
    free(y);
```

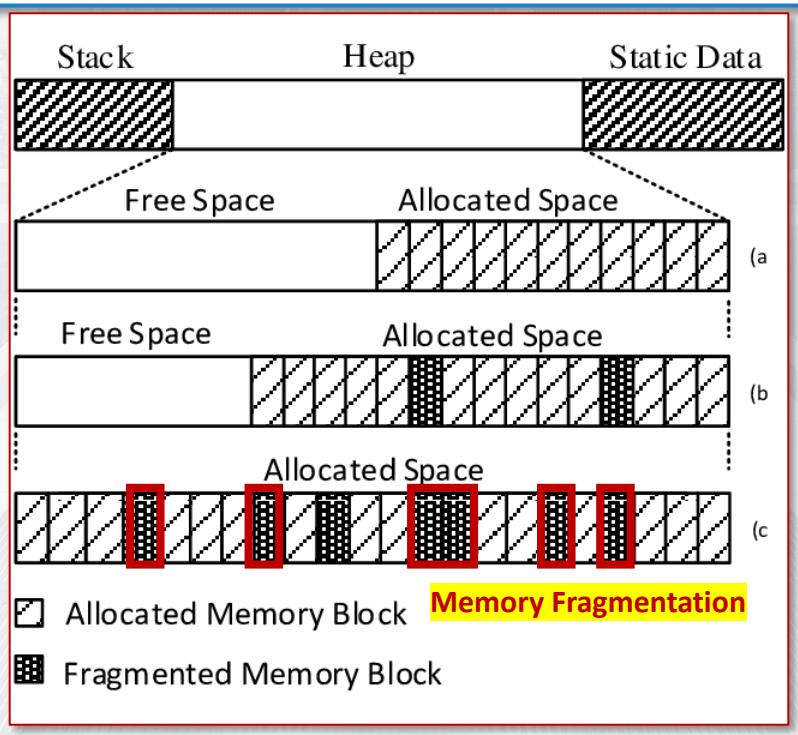
```
        goto rej;

    free(w0); free(w1);
    pack_sig(sig, sig, z, &h);
    free(z);
    *siglen = CRYPTO_BYTES;
    return 0;
}
```


SmartM2M 스마트엠투엠

1 효율적인 메모리 사용 방안

1.2 동적 할당을 통한 메모리 사용량 최적화 (3/3)

- 하지만, 동적할당은 필연적으로 Memory Fragmentation 문제를 유발함
  - 잦은 동적할당과 동적할당 해제는 가용 메모리 공간이 조각화를 유발시킴 (external memory fragementation)
  - 이러한 Memory Fragmentation은 메모리의 할당 실패할 수 있으며 예상치 못한 오류가 발생할 수 있음
  - 기존 구현 제출물 중에서는 동적할당과 해제를 반복하여 실행되는 코드에서는 시간이 지남에 따라, 메모리 할당 오류가 발생하는 경우가 다수 존재함 ← Memory Fragmentation으로 인한 문제
  - Cortex-M4 환경에서 이러한 문제를 해결할 수 있는 효율적인 구현 방안을 도출할 필요가 있음
    - ✓ Cortex-M4 에서는 memory management 기능 제공안함 (OS 역할) → customized memory allocator 구현 필요



SmartM2M
스마트엠투엠

2 적절한 변수 자료형 사용

2.1 32bit 임베디드 보드에서 지원하는 자료형 사용 - 사례#1. NCCSIGN

- 일부 Lattice based PQC에서는 128bit 자료형이 사용되고 있음
  - 128bit 자료형 변수는 64bit 자료형 변수 2개(high, low)로 분할하여 연산하는 것이 필요함
  - 64bit 자료형 2개로 분할하는 경우 각 변수의 캐리, 오버플로우를 고려한 기능 구현

```
// t = (__uint128_t)acc1*(__uint128_t)acc5;
// ttmp = (t & 0xffffffff) << 64;
// t = t << 64;
t_high = (uint64_t)acc1 * (uint64_t)acc5;
ttmp_high = t_high & mask_coef;

// imn = (uint64_t)acc2*(uint64_t)acc6;
// t = (__uint128_t)(t ^ imn);
// ttmp = ttmp ^ (imn & 0xffffffff);
t_low = (uint64_t)acc2 * (uint64_t)acc6;
ttmp_low = t_low & mask_coef;

// mt = ttmp * QINV;
// mt = mt & ((__uint128_t)(mask_coef) << 64 | (__uint128_t)(mask_coef));
mt_high = (ttmp_high * QINV) & mask_coef;
mt_low  = (ttmp_low  * QINV) & mask_coef;

// mt *= Q;
// mt = t + mt;
// b = mt >> 32;
mt_high = mt_high * (uint64_t)Q;
mt_low  = mt_low  * (uint64_t)Q;
add_128_128(&mt_high, &mt_low, mt_high, mt_low, t_high, t_low);
shift_right32_128(&b_high, &b_low, mt_high, mt_low);

// t1 = (uint64_t)(b & 0xFFFFFFFF);
t1 = b_low & mask_coef;
t1 -= ((Q - t1) >> 63) & Q;

// t2 = (b) >> 64;
t2 = b_high;
t2 -= ((Q - t2) >> 63) & Q;
```

Karatsuba 연산 코드의 일부

128bit -> 64bit 변환 예시

128bit	ttmp	
64bit	ttmp_high	ttmp_low
digit	63 0	63 0

- karatsuba 연산에 사용되는 128bit 자료형은 곱셈에 대한 중간값 및 캐리 저장, 모듈러 연산을 위한 용도로 사용됨
- 변수의 사용목적과 알고리즘 분석 결과
  64bit 변수 ttmp_high, ttmp_low를 사용하더라도 알고리즘의 동작에 문제가 없음

- 128bit 변수 mt 와 t 를 더하는 경우, high와 low 각자 덧셈 수행
- 단, low 에서 캐리 발생시 high의 결과에 1을 더함

```
void add_128_128(
    uint64_t* result_high, uint64_t* result_low,
    uint64_t a_high, uint64_t a_low,
    uint64_t b_high, uint64_t b_low)
{
    *result_low = a_low + b_low;

    if (*result_low < a_low)
        *result_high = a_high + b_high + 1;
    else
        *result_high = a_high + b_high;
}
```


SmartM2M 스마트엠투엠

2 적절한 변수 자료형 사용

2.2 size_t 자료형으로 인한 오버플로우 발생 - 사례#2. SMAUG

- size_t 자료형
  - size_t 자료형은 부호 없는 정수 자료형(unsigned)이며, 시스템 아키텍처 및 컴파일러에 따라 다르게 정의됨
  - 32bit processor → 4byte 자료형, 64bit processor → 8byte 자료형으로 정의됨
  - 이에, size_t 자료형을 사용하는 일부 코드에서는 오버플로우가 발생하는 경우 있음 (32bit 환경이므로)

```
#define SHARED_SECRETE_BYTES 32
#define CRYPTO_BYTES SHARED_SECRETE_BYTES
```

Reference 코드

```
uint8_t genSx(uint8_t *s, const uint8_t *seed)
{
    uint8_t xof_res[CRYPTO_BYTES] = {0};

    shake128(xof_res, CRYPTO_BYTES, seed, CRYPTO_BYTES + sizeof(size_t));

    uint8_t s_temp[LWE_N] = {0};
    hwt(s_temp, xof_res, CRYPTO_BYTES, HS);

    return convToIdx(s, HS, s_temp, LWE_N, 0);
}
```

CRYPTO_BYTES = 32

64BIT 환경에서는 CRYPTO_BYTES + sizeof(size_t) = (32 + 8) byte를 의미
32BIT 환경에서는 CRYPTO_BYTES + sizeof(size_t) = (32 + 4) byte를 의미

```
uint8_t genSx(uint8_t *s, const uint8_t *seed)
{
    uint8_t xof_res[CRYPTO_BYTES] = {0};

    shake128(xof_res, CRYPTO_BYTES, seed, CRYPTO_BYTES + 8);

    uint8_t s_temp[LWE_N] = {0};
    hwt(s_temp, xof_res, CRYPTO_BYTES, HS);

    return convToIdx(s, HS, s_temp, LWE_N, 0);
}
```

32bit 환경의 Cortex-M4 환경에서는 sizeof(size_t)를 4 byte로 인식하므로 정확한 값을 입력해야 함

변경된 코드

SmartM2M 스마트엠투엠

III 경량환경 KpqC 성능평가

SmartM2M
스마트엠투엠

1 경량환경에서의 성능 평가 방법

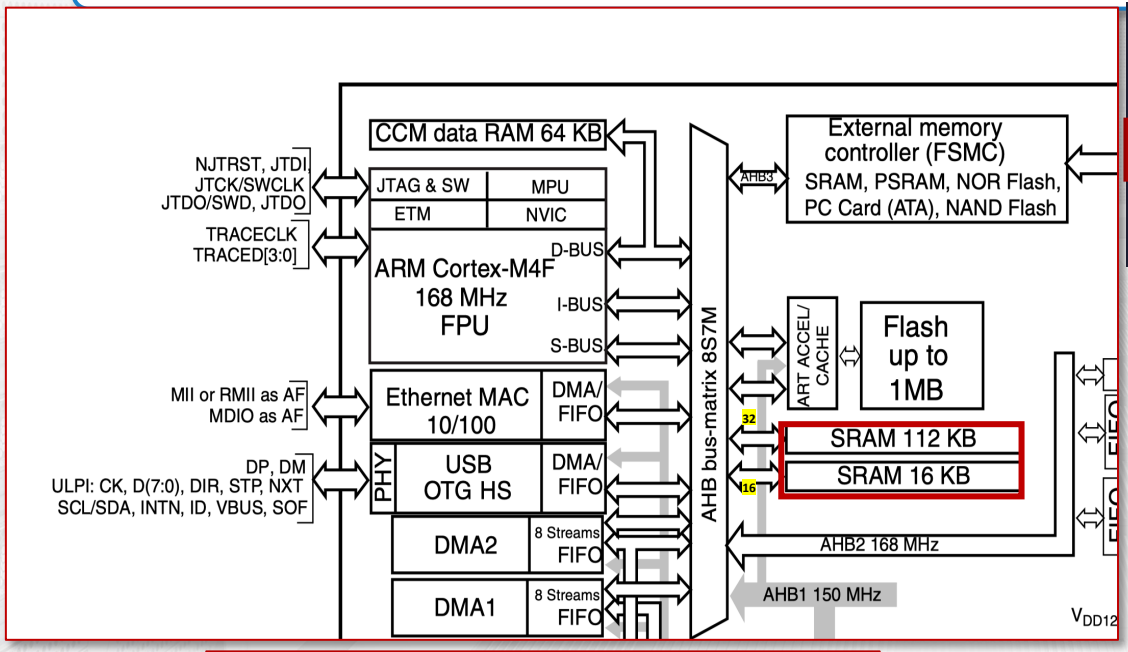
1.1 STM32F407G-DISC1 RAM 설정

- 128KB의 SRAM 활용

CCM : Core Coupled Memory

  - 192KB의 RAM 중 CCM 메모리 64KB를 제외하면 최대 128KB의 RAM을 사용할 수 있음
  - Cortex M4 프로세서 내부에는 SRAM이 두개 있음 - SRAM1(112KB), SRAM2(16KB)
    - ✓ 일반적으로 정적/동적 메모리 공간 할당 목적으로 각각 사용됨. memory protection 수준 등이 다름
    - ✓ 112KB SRAM : 1~2 clock cycles access time, 32bits bus width, 16KB SRAM: 2~3 clock cycles access time, 16bits bus width
  - 한편, 사용자 정의 Linker Script를 사용하여, 128KB의 SRAM 1개로 merge 가능(상위 수준의 merge)

  → 상위 수준에서 메모리 확보 차원에서 두개의 SRMA을 merge하여 사용함



STM32F407 Block Diagram

```
EXTERN(vector_table)
ENTRY(reset_handler)
MEMORY
{
 ram (rwx) : ORIGIN = 0x20000000, LENGTH = 112K
 ram2 (rwx) : ORIGIN = 0x2001c000, LENGTH = 16K
 rom (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
 ccm (rwx) : ORIGIN = 0x10000000, LENGTH = 64K
}
```

2개의 ram으로 분할

stm32f4discovery.ld (Default 메모리 설정)

```
EXTERN(vector_table)
ENTRY(reset_handler)
MEMORY
{
 ram (rwx) : ORIGIN = 0x20000000, LENGTH = 128K
 rom (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
 ccm (rwx) : ORIGIN = 0x10000000, LENGTH = 64K
}
```

1개의 RAM

stm32f4discovery_fullram.ld
(변경된 RAM 공간)

FrodoKEM, Round5, Dilithium이 128KB RAM 환경에서 동작

1.2 컴파일 최적화 옵션

- 본 연구에서는 최적화 옵션으로 -O3만을 고려하여 측정
  - KpqC 1 Round 제출물 대부분은 -O3로 컴파일을 수행하고 있으므로 본 연구에서도 동일하게 -O3 옵션으로 컴파일을 수행
  - -O3로 최적화 옵션 설정으로 Inlining 및 Loop Unrolling, Constant Propagation 최적화가 수행됨
  - 코드의 크기는 증가하지만 성능 최적화를 위한 옵션으로 컴파일함

Optimization Level	설명
-O0	• 모든 최적화 기능 비활성화 • 소스코드와 매우 유사한 흐름이 되도록 빌드 됨 • 컴파일 및 빌드 속도는 빠르나, 동작시 스택 사용량이 많고 느림
-O1	• 일부 코드 최적화가 이루어짐. 비교적 빠른 컴파일 속도 • 코드 크기 증가는 거의 없음
-O2	• O1보다 더 높은 수준의 최적화. 일부 Inlining 및 Loop Unrolling 수행 • 단순 반복문 및 독립적인 스칼라 연산들에 대한 벡터화 및 벡터 연산 명령어 추가될 수 있음
-O3	• 가장 높은 수준의 최적화 • 성능향상이 크지만 이로 인해 코드 크기가 증가 및 컴파일 시간 오래 걸림 • Loop Unrolling 및 Constant Propagation, Instruction Scheduling 등의 기법이 적용되어 성능을 높임

```
DEBUG ?=
OPT_SIZE ?=
LTO ?=
AIO ?= 1
MUPQ_ITERATIONS ?= 1

RETAINED_VARS += DEBUG OPT_SIZE LTO AIO MUPQ_ITERATIONS

ifeq ($(DEBUG),1)
CFLAGS += -O -g3
else ifeq ($(OPT_SIZE),1)
CFLAGS += -Os -g3
else
CFLAGS += -O3 -g3
endif
```

-O3 옵션 사용 시 디버깅이 어렵기 때문에 최적화 하지 않음

-O3 옵션으로 컴파일 수행

Kpqm4 makefile 일부

SmartM2M 스마트엠투엠

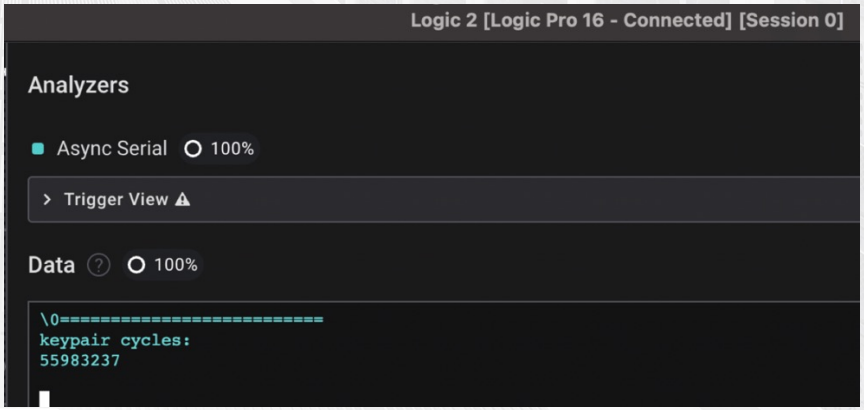
1 경량환경에서의 성능 평가 방법

1.3 KpqC Round 1 속도 측정 방법

```c
int main(void)
{
  unsigned char sk[MUPQ_CRYPTO_SECRETKEYBYTES];
  unsigned char pk[MUPQ_CRYPTO_PUBLICKEYBYTES];
  unsigned char sm[MLEN+MUPQ_CRYPTO_BYTES];
  size_t smlen;
  unsigned long long t0, t1;
  int i;

  hal_setup(CLOCK_BENCHMARK);

  hal_send_str("==========================");

  for(i=0;i<MUPQ_ITERATIONS; i++)
  {
    // Key-pair generation
    t0 = hal_get_time();
    MUPQ_crypto_sign_keypair(pk, sk);
    t1 = hal_get_time();
    printcycles("keypair cycles:", t1-t0);

    // Signing
    t0 = hal_get_time();
    MUPQ_crypto_sign(sm, &smlen, sm, MLEN, sk);
    t1 = hal_get_time();
    printcycles("sign cycles:", t1-t0);

    // Verification
    t0 = hal_get_time();
    MUPQ_crypto_sign_open(sm, &smlen, sm, smlen, pk);
    t1 = hal_get_time();
    printcycles("verify cycles:", t1-t0);

    hal_send_str("+");
  }
  hal_send_str("#");
  return 0;
}
```

```c
uint64_t hal_get_time()
{
  while (1) {
    unsigned long long before = overflowcnt;
    unsigned long long result = (before + 1) * 16777216llu - SysTick->VAL;
    if (overflowcnt == before) {
      return result;
    }
  }
}
```

- 성능 측정 시, Cortex M4의 SysTick(24bit System Timer) 사용. Downcounter임 (2^24 → 16,777,216)
- Timer 값이 down되어, 0이되면(정확히는 underflow임), 다시 처음(2^24-1)부터 시작함. 즉, 16,777,216부터 다시 시작함(overflowcnt 값 증가함)
- 16,....에서 SysTick->VAL 값을 빼면, 경과된 시간을 의미함
- If 조건 : SysTick timer가 overflow(실제는 underflow)된 경우, overflowcnt 값은 before와 달려졌을 것임 → 증가된 overflowcnt 값을 다시 반영(before ← overflowcnt)하여, 경과된 시간을 계산함



임베디드 보드 상의 출력 확인

SmartM2M 스마트엠투엠

1 경량환경에서의 성능 평가 방법

1.4 KpqC Round 1 Stack Usage 측정 방법

- **STM32F407G-DISC1 스택 사용량 측정 방법**
  - 정확한 스택 사용량 측정을 위해서는 암호 알고리즘 구현 시 Heap 영역을 사용하지 않아야 함
    ex) 동적할당
  - 본 연구결과에서는 메모리 오버 플로우를 방지하기 위해 동적할당을 다수 사용하고 있으며, 이에 정확한 스택 사용량 측정이 제한됨
  - NCCSIGN 및 SOLMAE의 경우 스택을 초과한 사용으로 인해 정확한 스택 사용량은 측정이 불가능함

```c
static int test_sign(void) {
  volatile unsigned char a;
  // Alice generates a public key
  FILL_STACK()
  MUPQ_crypto_sign_keypair(pk, sk);
  CHECK_STACK()
  if(c >= canary_size) return -1;
  stack_key_gen = c;

  // Bob derives a secret key and creates a response
  randombytes(m, MLEN);
  FILL_STACK()
  MUPQ_crypto_sign(sm, &smlen, m, MLEN, sk);
  CHECK_STACK()
  if(c >= canary_size) return -1;
  stack_sign = c;

  // Alice uses Bobs response to get her secret key
  FILL_STACK()
  rc = MUPQ_crypto_sign_open(sm, &mlen, sm, smlen, pk);
  CHECK_STACK()
  if(c >= canary_size) return -1;
  stack_verify = c;
```

```c
#define FILL_STACK()
  p = &a;
  while (p > &a - canary_size)
    *(p--) = canary;
#define CHECK_STACK()
  c = canary_size;
  p = &a - canary_size + 1;
  while (*p == canary && p < &a) {
    p++;
    c--;
  }
```

스택 사용량 측정을 위해 스택의 특정 공간을 canary_size만큼 canary로 채운 후 그 값이 변경된 부분을 확인하여 최종 스택 사용량을 측정함

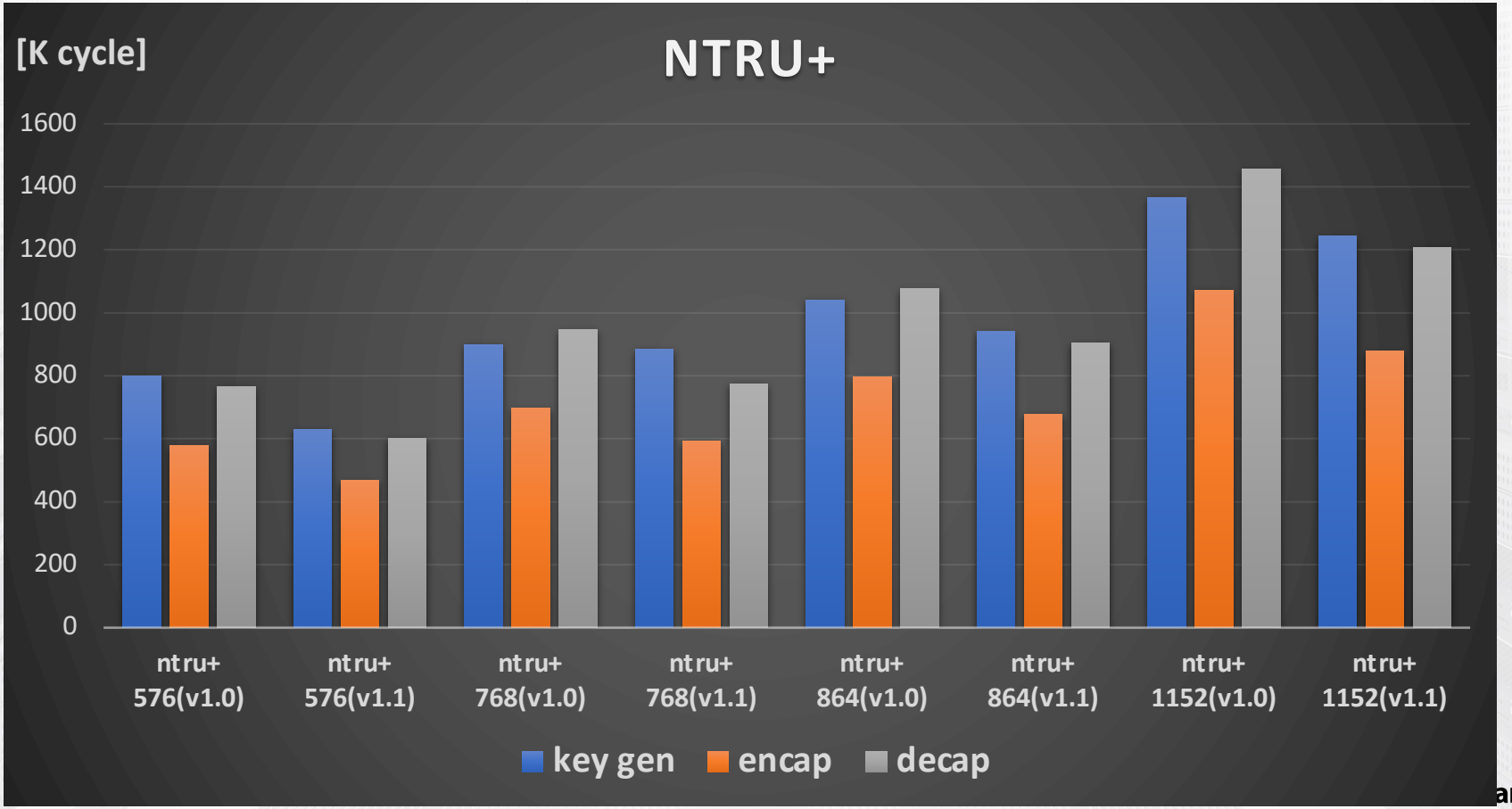
이에, 동적할당 등 Heap 영역을 사용할 경우에는 메모리 오버플로우가 발생할 가능성이 매우 높음

SmartM2M 스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.1 KpqC Round 1 KEM 알고리즘 성능 평가 - NTRU+

- NTRU+ v1.0 및 v1.1 성능 측정 결과
  - 각 파라미터별 v1.1에서의 성능 개선을 확인할 수 있음

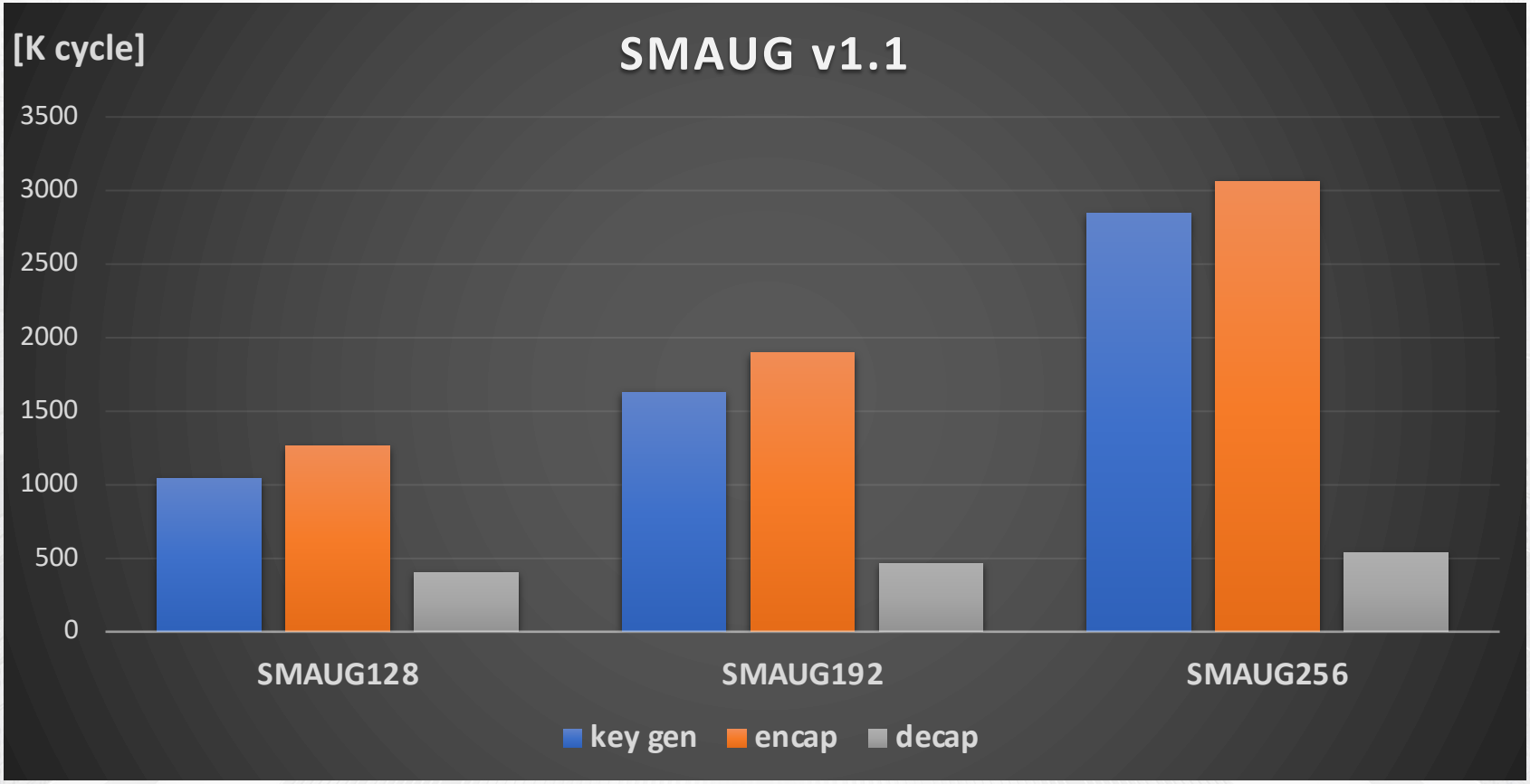


artM2M
스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.2 KpqC Round 1 KEM 알고리즘 성능 평가 - SMAUG(v1.1)

- SMAUG v1.1 성능 측정 결과
  - Decapsulation에서의 성능이 우수

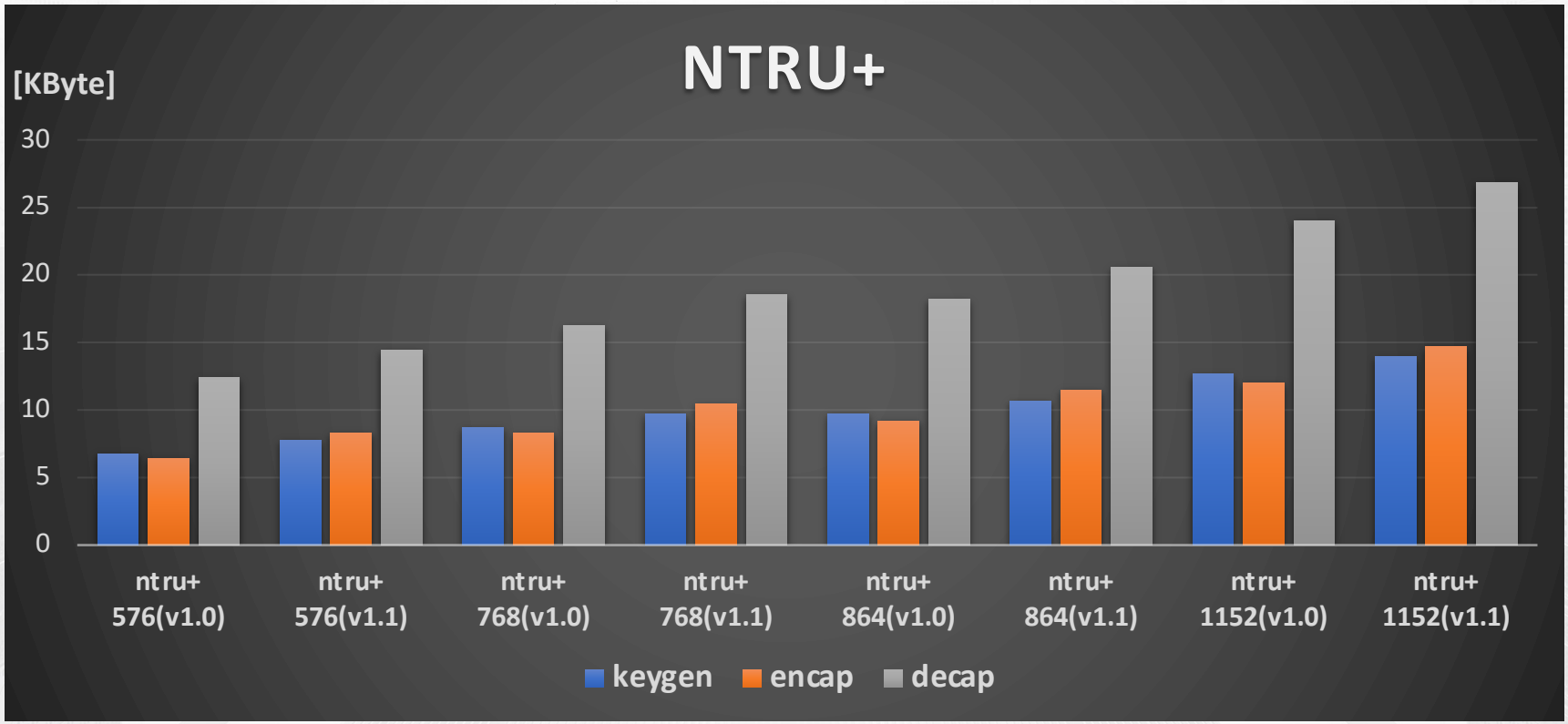


SmartM2M
스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.3 KpqC Round 1 KEM 알고리즘 Stack Usage 평가 - NTRU+

- NTRU+ Stack Usage 측정 결과
  - v1.1에서 성능의 개선은 이루어졌으나, Stack 사용량은 소폭 증가

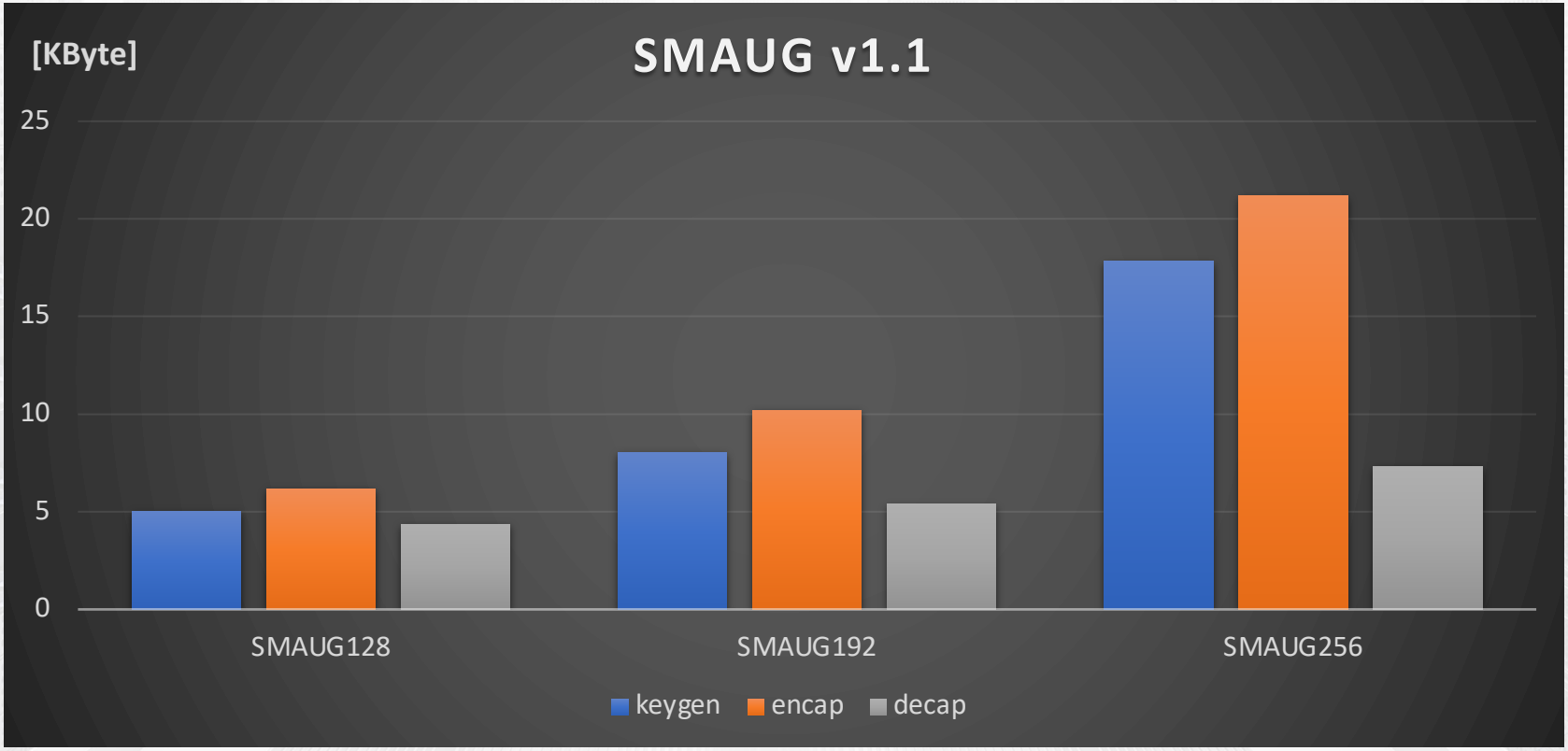


SmartM2M 스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.3 KpqC Round 1 KEM 알고리즘 Stack Usage 평가 - SMAUG(v1.1)

- SMAUG Stack Usage 측정 결과
  - SMAUG 256에서 Stack 사용량이 다소 증가
  - 내부적으로 동적할당을 사용하고 있으므로 실제 Stack 사용량은 더 클 것으로 예상됨

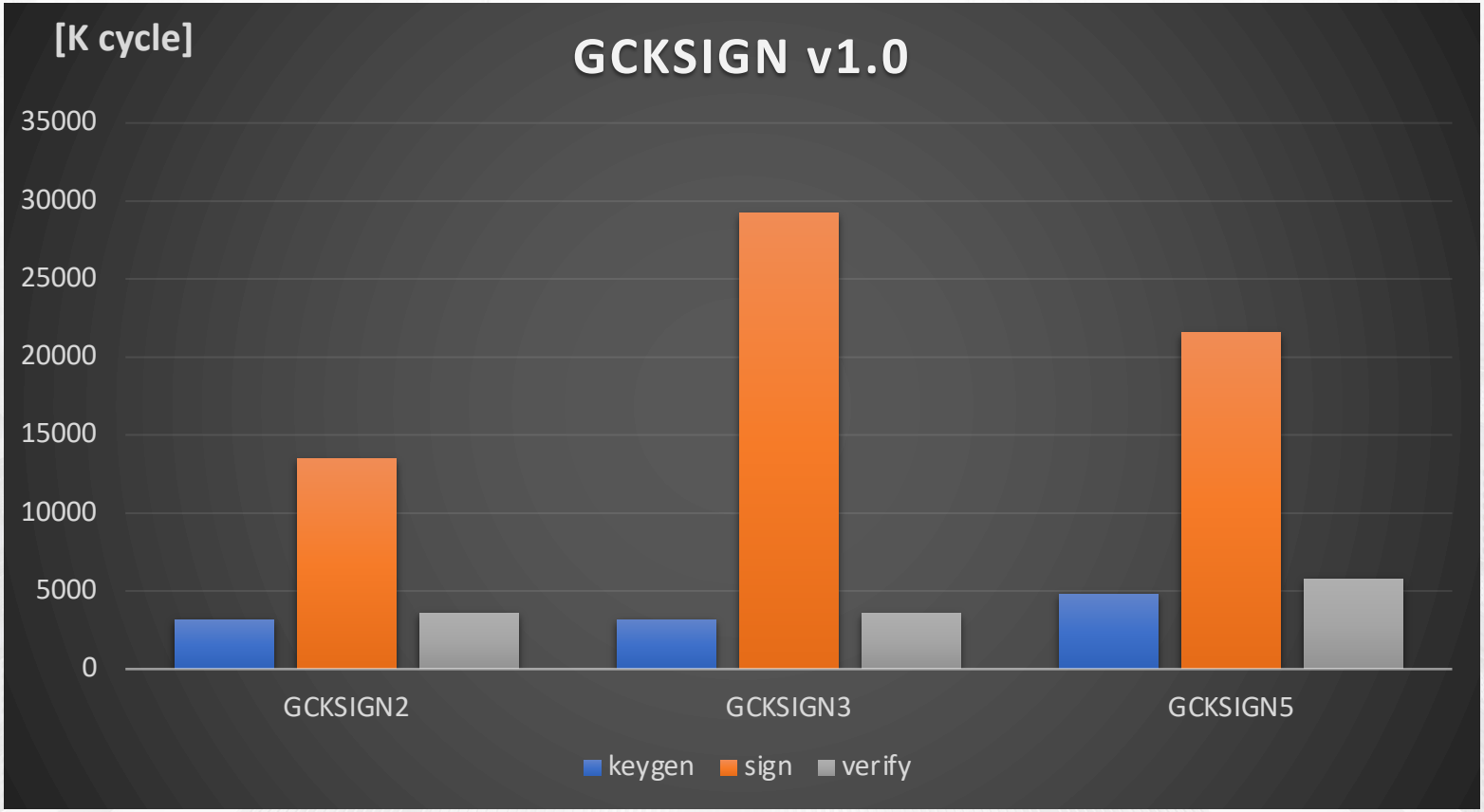


SmartM2M
스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.4 KpqC Round 1 Signature 알고리즘 성능 평가 - GCKSIGN (v1.0)

- GCKSIGN 성능 측정 결과
  - 키 생성 및 서명 검증 성능은 우수하며, 서명 생성에서 비교적 많은 시간이 소요

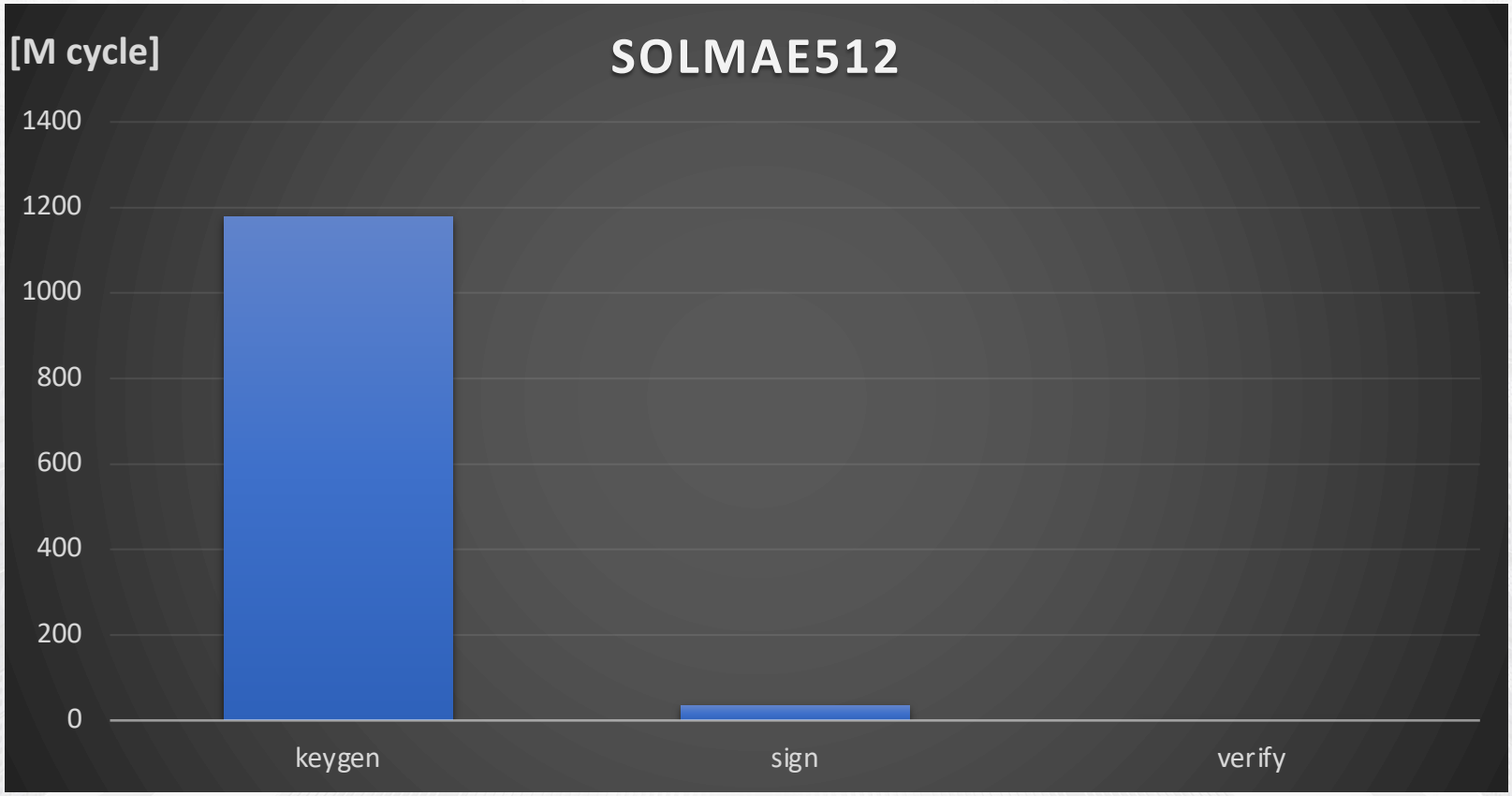


SmartM2M 스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.5 KpqC Round 1 Signature 알고리즘 성능 평가 - SOLMAE 512 (v1.0)

- SOLMAE 512 성능 측정 결과
  - 키 생성 시간은 다소 소요되지만 서명, 서명 검증에는 비교적 짧은 시간이 소요

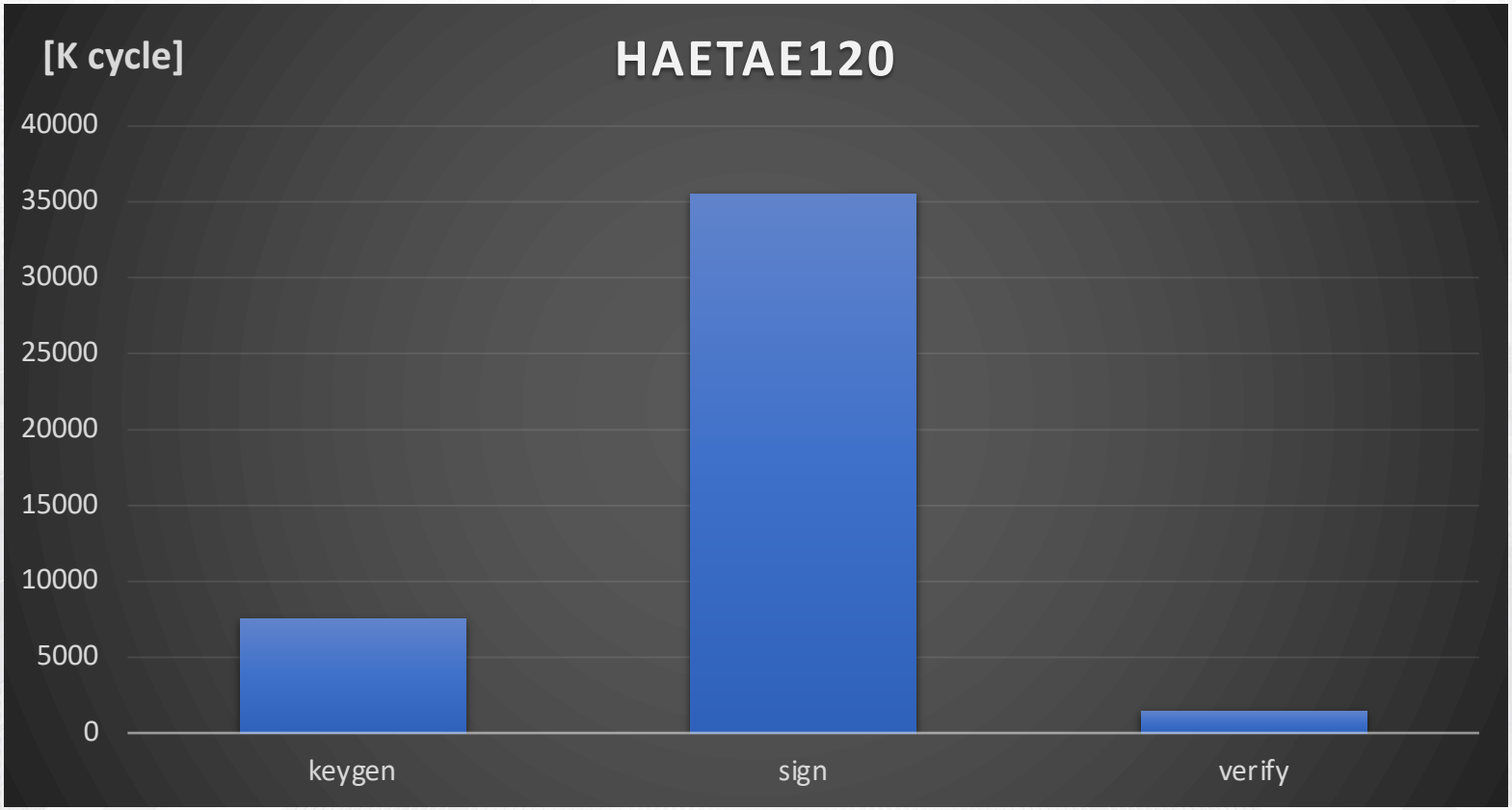


SmartM2M
스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.6 KpqC Round 1 Signature 알고리즘 성능 평가 - HAETAE 120 (v1.1)

- HAETAE 120 성능 측정 결과
  - 키 생성 및 서명 검증 성능은 우수하며, 서명 생성에서 비교적 많은 시간이 소요

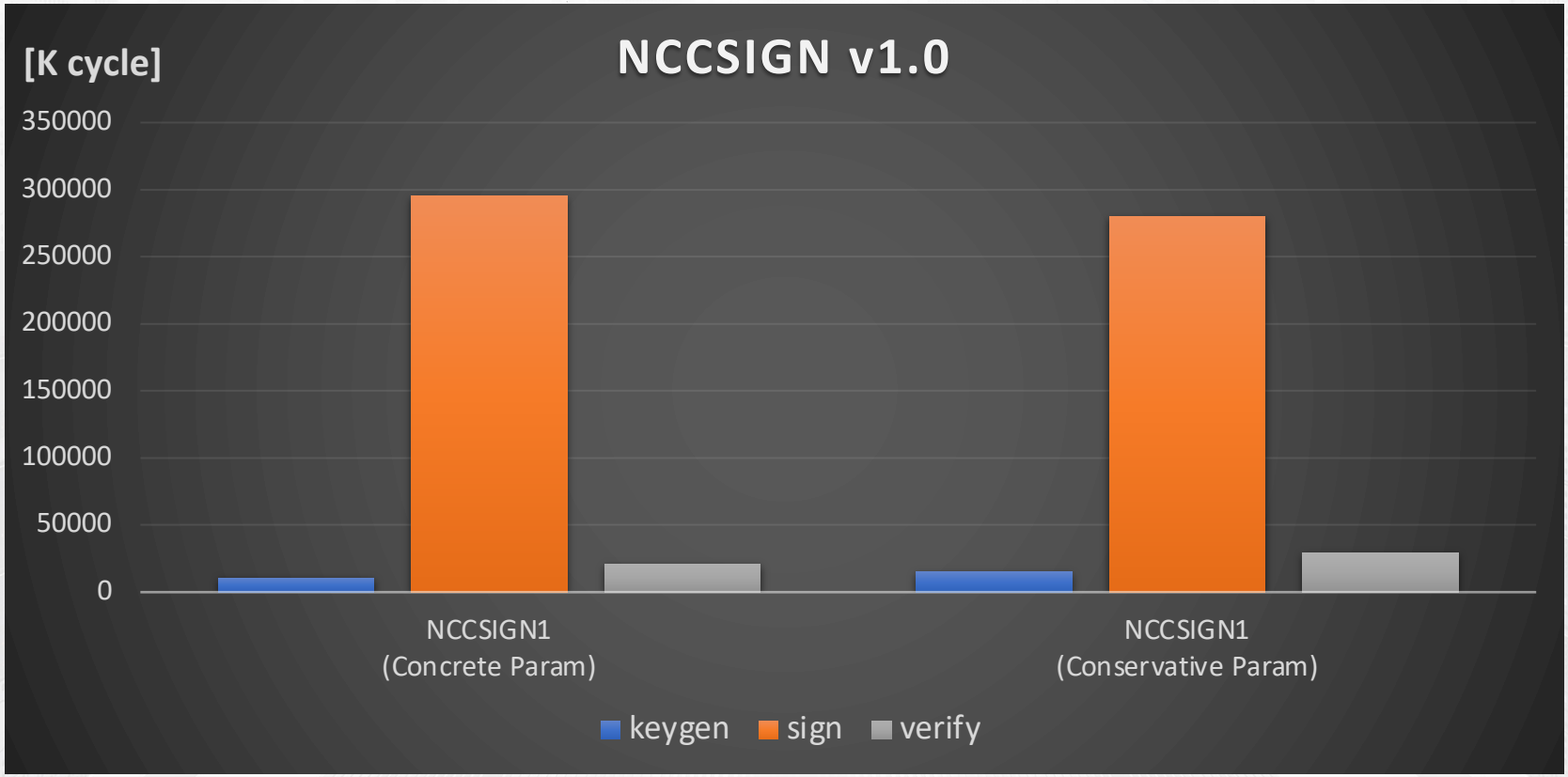


SmartM2M 스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.7 KpqC Round 1 Signature 알고리즘 성능 평가 - NCCSIGN (v1.0)

- NCCSIGN 성능 측정 결과
  - 키 생성 및 서명 검증 성능은 우수하며, 서명 생성에서 비교적 많은 시간이 소요

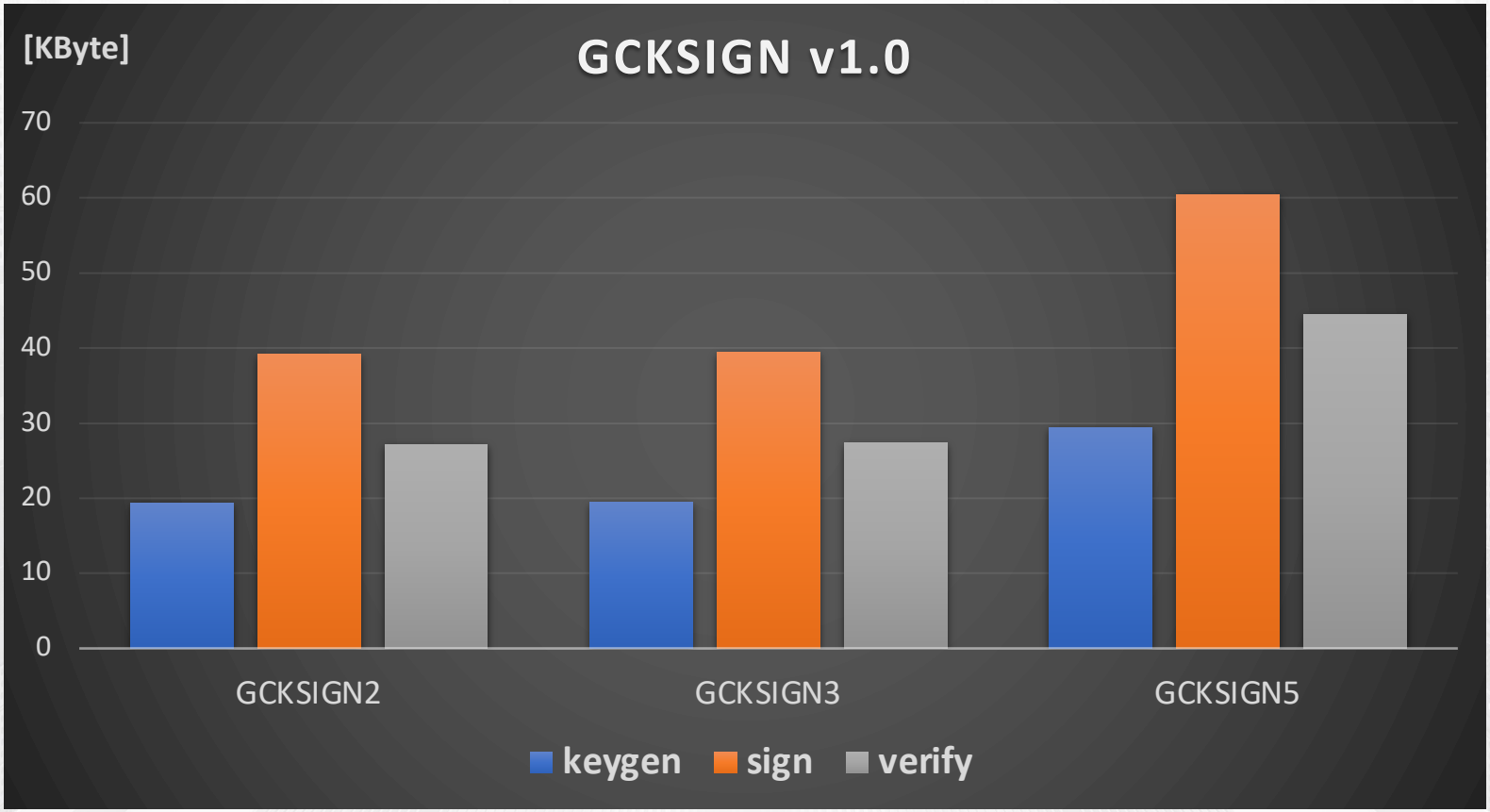


SmartM2M
스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.8 KpqC Round 1 Signature 알고리즘 Stack Usage 평가 - GCKSIGN (v1.0)

- GCKSIGN Stack Usage 평가 결과
  - GCKSIGN2,3의 Stack 사용량 변화는 미미하나, GCKSIGN5에서 Stack 사용량이 증가

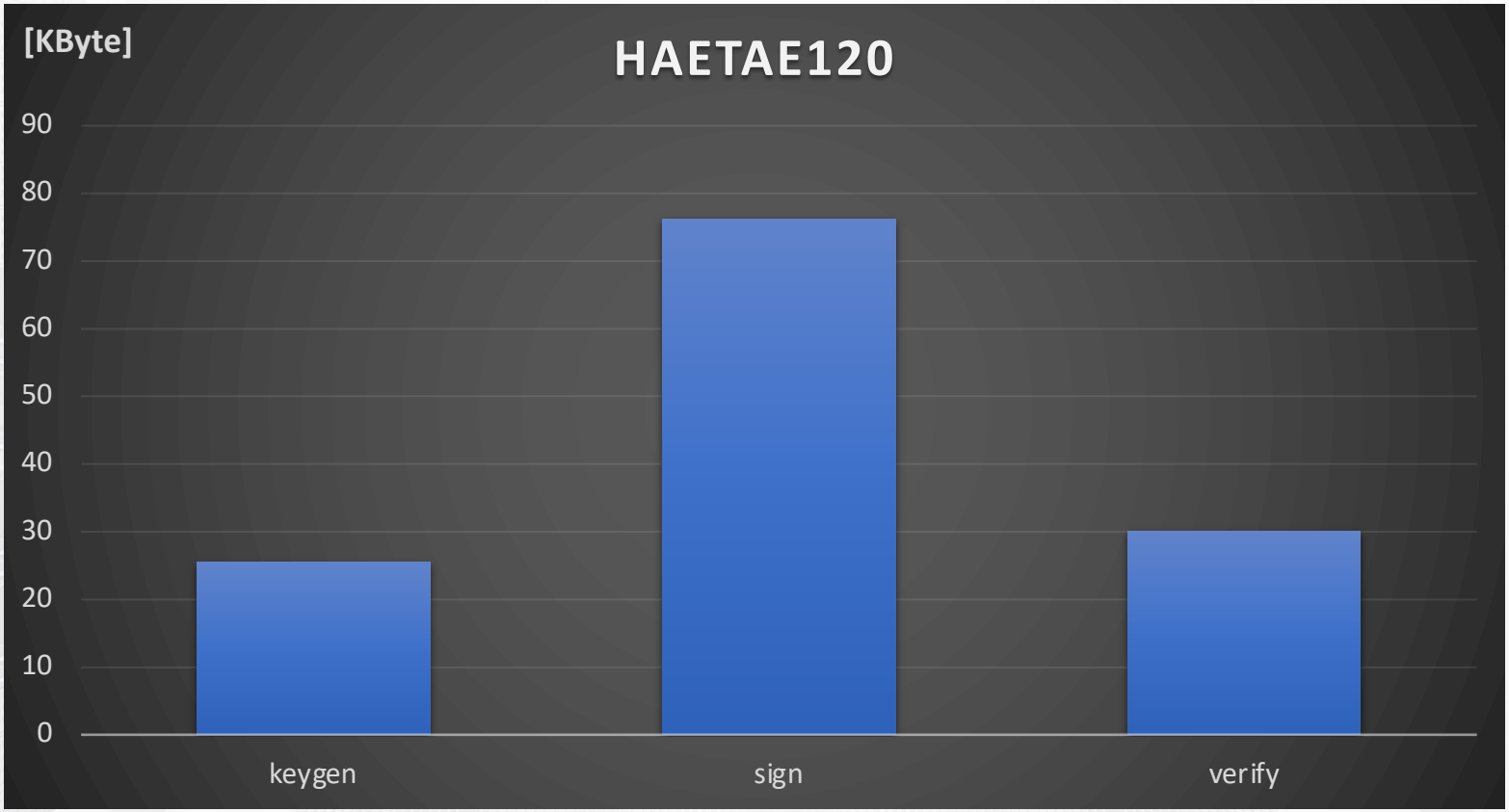


SmartM2M 스마트엠투엠

2 경량환경에서의 성능 평가 결과

2.9 KpqC Round 1 Signature 알고리즘 Stack Usage 평가 - HAETAE (v1.1)

- HAETAE Stack Usage 평가 결과
  - 서명 생성 시 Stack 사용량이 다소 증가



SmartM2M 스마트엠투엠

감사합니다

부록 - Kpqm4 Benchmark

NTRU+ 성능 측정 결과표

Scheme	Parameter (byte size)	Implementation	Key Generation [cycles]	Encapsulation [cycles]	Decapsulation [cycles]
NTRU+576 (v1.0)	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 (v1.0)	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 (v1.0)	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 (v1.0)	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**
NTRU+576 (v1.1)	sk : 1,760 pk : 864 ct : 864	ntru+_ref	**285,000**	**106,000**	**135,000**
		ntru+_avx2	**20,000**	**20,000**	**12,000**
		m4clean	**630,293**	**467,932**	**602,201**
NTRU+768 (v1.1)	sk : 2,336 pk : 1,152 ct : 1,152	ntru+_ref	**325,000**	**137,000**	**177,000**
		ntru+_avx2	**24,000**	**26,000**	**17,000**
		m4clean	**884,715**	**592,063**	**774,854**
NTRU+864 (v1.1)	sk : 2,624 pk : 1,296 ct : 1,296	ntru+_ref	**324,000**	**162,000**	**217,000**
		ntru+_avx2	**23,000**	**28,000**	**18,000**
		m4clean	**939,662**	**678,338**	**904,570**
NTRU+1152 (v1.1)	sk : 3,488 pk : 1,728 ct : 1,728	ntru+_ref	**770,000**	**204,000**	**288,000**
		ntru+_avx2	**45,000**	**36,000**	**24,000**
		m4clean	**1,244,108**	**877,784**	**1,208,407**

부록 - Kpqm4 Benchmark

NTRU+ 스택사용량 측정 결과표

Scheme	Parameter (byte size)	Implementation	Key Generation [byte]	Encapsulation [byte]	Decapsulation [byte]
NTRU+576 (v1.0)	sk : 1,728 pk : 864 ct : 864	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	6,948	6,604	12,704
NTRU+768 (v1.0)	sk : 2,304 pk : 1,152 ct : 1,152	ntru_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	8,964	8,500	16,664
NTRU+864 (v1.0)	sk : 2,592 pk : 1,296 ct : 1,296	ntru_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	9,972	9,452	18,648
NTRU+1152 (v1.0)	sk : 3,456 pk : 1,728 ct : 1,728	ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	13,004	12,300	24,592
NTRU+576 (v1.1)	sk : 1,760 pk : 864 ct : 864	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	7,976	8,552	14,760
NTRU+768 (v1.1)	sk : 2,336 pk : 1,152 ct : 1,152	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	9,944	10,696	18,960
NTRU+864 (v1.1)	sk : 2,624 pk : 1,296 ct : 1,296	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	10,952	11,792	21,088
NTRU+1152 (v1.1)	sk : 3,488 pk : 1,728 ct : 1,728	ntru+_ref	-	-	-
		ntru+_avx2	-	-	-
		m4clean	14,276	15,072	27,464

부록 - Kpqm4 Benchmark

SMAUG 성능 측정 결과표

Scheme	Parameter (byte size)	Implementation	Key Generation [cycles]	Encapsulation [cycles]	Decapsulation [cycles]
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

SMAUG 스택사용량 측정 결과표

Scheme	Parameter (byte size)	Implementation	Key Generation [byte]	Encapsulation [byte]	Decapsulation [byte]
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

SmartM2M 스마트엠투엠

부록 - Kpqm4 Benchmark

GCKSIGN, HAETAE, SOLMAE, NCCSIGN 성능 측정 결과표

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**
SOLMAE512	sk : pk : sig :	16,385 896 666	solmae-ref	**27,000,000**	**387,000**	**40,000**
			m4clean	**1,178,774,338**	**35,088,447**	**2,640,319**
NCCSIGN1 (Concrete Param)	sk : pk : sig :	2,400 1,400 2,529	nccsign_ref	**1,257,562**	**16,174,808**	**2,444,616**
			m4clean	**11,297,649**	**295,016,934**	**21,586,176**
NCCSIGN1 (Conservative Param)	sk : pk : sig :	2,800 1,984 3,186	nccsign_ref	**1,727,508**	**11,768,076**	**3,400,702**
			m4clean	**15,606,311**	**280,078,650**	**29,796,191**

SmartM2M 스마트엠투엠

부록 - Kpqm4 Benchmark

GCKSIGN, HAETAE, SOLMAE, NCCSIGN 스택 사용량 측정 결과표

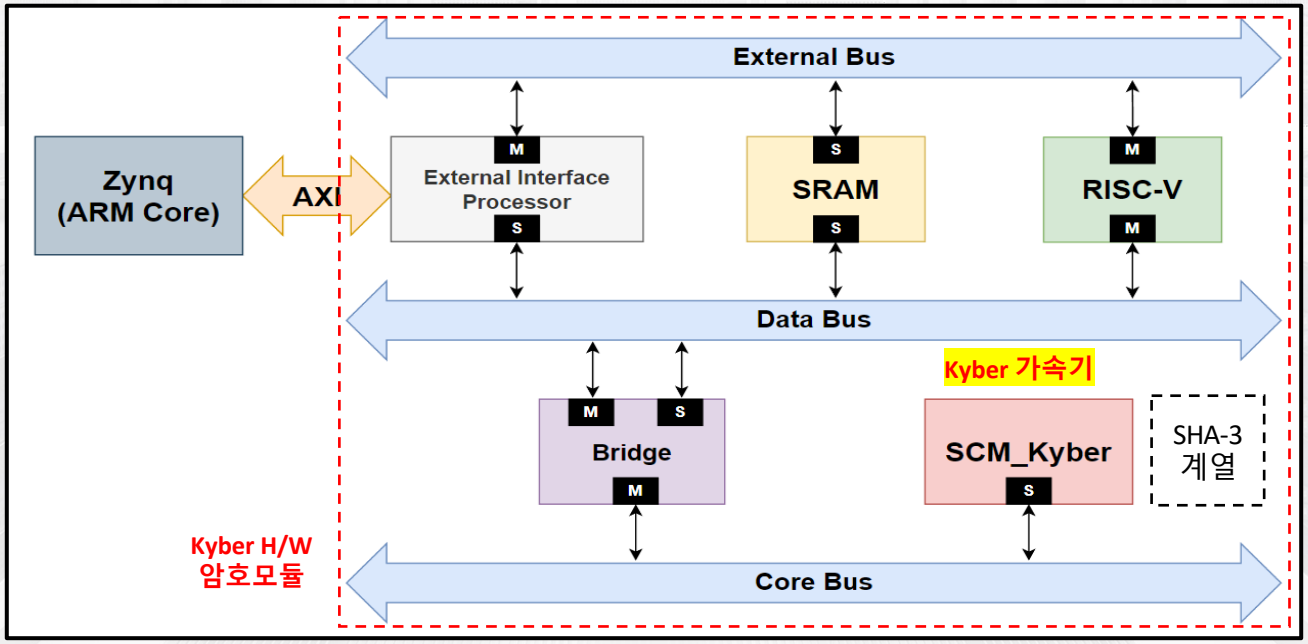
Scheme	Parameter (byte size)		Implementation	Key Generation [bytes]	Sign [bytes]	Verify [bytes]
GCKSIGN2	sk : pk : sig :	288 1,760 1,952	m4clean	19,892	40,132	27,860
GCKSIGN3	sk : pk : sig :	288 1,952 2,080	m4clean	19,932	40,348	28,044
GCKSIGN5	sk : pk : sig :	544 3,040 2,080	m4clean	30,172	61,916	45,524
HAETAE120	sk : pk : sig :	1,376 992 1,463	m4clean	26,116	78,084	30,780
SOLMAE512	sk : pk : sig :	16,385 896 666	m4clean	131,072 이상	131,072 이상	131,072 이상
NCCSIGN1 (Concrete Param)	sk : pk : sig :	2,400 1,400 2,529	m4clean	131,072 이상	131,072 이상	131,072 이상
NCCSIGN1 (Conservative Param)	sk : pk : sig :	2,800 1,984 3,186	m4clean	131,072 이상	131,072 이상	131,072 이상

SmartM2M 스마트엠투엠

부록

스마트엠투엠 Kyber-512 하드웨어(FPGA) 특징 - 유사 KpqC 알고리즘 SMAUG

- 주요 구현 특징
  - RISC-V 프로세서를 이용한 Kyber (Lattice) 하드웨어 가속기 형태로 구현 (SHA-3 계열 연산은 RISC-V 활용)
    - RISC-V 프로세서는 암호 모듈에 사용되는 데이터 이동 / 키 관리 등을 수행
    - FPGA 기준 자원 소모량 최적화 구현 및 SW대비 성능 향상
  - SHA-3 계열 HASH 함수 연산은 RISC-V 프로세서를 이용한 SW+HW 하이브리드 형식 구현
    - PKE/KEM 모두 지원, 추후 SHA-3 하드웨어 모듈 업데이트 예정 (성능 향상 가능)



SCM_Kyber 아키텍처

SmartM2M
스마트엠투엠

부록

스마트엠투엠 Kyber-512 하드웨어(FPGA) 구현 사례 (정상 동작 확인)

Scheme	Parameter (byte size)		Implementation	LUT(개)	Flip Flop(개)	BRAM(개)	DSP(개)	Frequency [MHz]	Latency(cycle)	
Kyber-512(PKE/KEM)	sk : pk : c :	768/1632 800/800 768/768	FPGA (ZCU3EG, 100MHz 타겟 합성) UltraScale+ MPSoC	10,719	8,538	41.5	22	134	Keygen: Enc: Dec:	3,196 5,314 2,710

```
[INFO] Keygen Start

d: 0x00FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1

[INFO] Keygen send done
[INFO] Keygen receive done
========================= Public Key =========================
7116424441593b5349376781940 4aa5899b485738b22306beeeab0e75aac8539
05bc122e987925dc331f10cacaa045c48438b33d90c157f1915862bc819c5e19
4a53a5c2c543932c0ac326a4d110b2801ad3fabc34c050cf77697b6387723c44
68d580b854cc62c38c91557f850a128e223f796a12a46c662b1762a39bb1ebb8
19db523e5c9c25ba4b64dbe118e1583a89422331ea6c9d8703f9460c79d6c19a
1a2ccc664db0308d81d90a86a5b53b884d40277d2c1c346df2956bc638b5b3ba
f8d39ba745bee164aa9ea3a519a060f21c2e3fb7114ba25b4137acf198cb0114
79a625aa                                                        b265
5b5e6c65                                                        0988
65385b3c                                                        8cab
65e09592                                                        0481
e38911a5                                                        8551
80653ae3                                                        59a5
9cf7438e92f854f4511645848f211610b69caf1c1350068c47ecab9cf612432e
e71534eccfcd43a417b91101b3b37ae5c7850b48f7c9459096349967c4f82ac6
cd49a16dd05140fb51d738440709ae47738c89837a0ec398a0a447c1bc74cd0c
5f74c08f2ab4168a871a012548bb33876f82ac41054e68443b02c702e8dab32a
e5583aec62946225bb6822e42386e8b501b3f70b58156a02f8124e1c92307370
63c01c90a5a1700709675abe66f7b4a95b021903170378 7f04a7b53338418e66
327f017f6e88065c4a4961960f5f982996401c7db01fdc2388c4db3dbac2c459
575e1c9a9f30b20f60b888630246c4165ce33a1a4479043128890522bd2112ad
ba36a076352b92eb3c4b955a9c573f81e44ac3179e282a2f297155b10b6e6c7c
77d1240d17093778a51fc68c9b22e064b9453d15ec07b7a2cbf0998e0a892cc5
010d30e63432c962b310b2ac46c11dd47e4a400705c622cfb51401a17c8431b0
==============================================================
```

공개키 생성 검증

```
========================= Private Key =========================
237289ea2235e3581ca733a110448522800824e021763b52789b49174b1ad609
b5a590c0a0e0350af1cfc1d39f5414b224e838b8bb5e26479eb79b34dfb6733e
3cb88da0735dea5a15707990d52f22fa8ab745376fc30577513af4e73ba987ac
ab0bc247704d6b70c26ee34393d750384cce91784a94da0bdab77a3a8b4027f6
6d8dd8ace1e5bcbb8a95317b3f1d20c0aa65341665670d395ebfe18a719c0b09
cc8c2cb65a5b899350c10890670e17c4ae1e5c12918991a787919c89a472a996
d54b665246cd63fca3c220ac49a1a845a449b9484ce58019945a2918b15acbb8
b043c1755e878153575e2f5782eed742ffc79ded538c4a521494c19a1042280b
f7b483cc5c965126c3e68c40909162813e42fc20cb2a20b44821e9ea45cb8796
68cc0a8c9b9adf108e60408c4c364780ac2bb2163d0914c48a579d18aac06a5c
18b279098df8261a9c58fe6a291bc50691fcacb84805c450caf6b846389a23bd
2c1b00c8aa6e988                                                 2707d21af
b0ca34c0aa42314                                                 c394c3d52
3483c96dfbaa495f                                                0918ba3df
33b58398937a0bbf                                                046f7119a
5379cd29b062c9885                                               178622f6b5
10db633bc72017d2b170868701abd55d77c5894892940151762107ba3dac6a61
60ac602a15b287b8a0972bd4f6a7f93b256762b79123c0146863be41c3506839
65704e5469bccc1a0170e993082acdf1910a7f84ab985506164a6be37c5c9d40
85dd11561a6a44c1567933b763aa280aaef81a3264ac3db96294b320f2574338
b90f39bc10b4e8001f1c2467777c09479b27c55fe8882319b449b365a6475116
53673dc82206b894700698865eec5f05fc8fb570c22d3a0b5f086c46a26c7697
5908a83ecde0911127624 8d01956a3cc9d233f6f8b1f30424bbdb88224411685
b86b3ff9b84aa4cbd22853a29a12f14a4879616c960a0681663d19440d8205a8
==============================================================
[INFO] Keygen End
```

개인키 생성 검증

```
[INFO] Encryption Start

Random coins: 0x00FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1
Msg: 0x534D4152544D324D53434D2D4B5942455235313253434D2D4B59424552353132

[INFO] send done
[INFO] receive done
========================= Chiper text =========================
c03185182befe7ae571fe2dcb67206969a33f5f52a6687a530d3b87a406ab1e
5a32cd5dc6293c851a81158b9043b0b4151ceb7cfd04d475187dc12111083ba8
9fb8673a6f43be053c445c000f48997dfa250561e7a8e0e25c88193009acfdc6
10368a40e303e2f66fce15accdcf819e6c42e2234c667d711d0bc710982cd935
cbb8d3fe4d176ef42ccfee75e95a616da014c081dde21f24ed5e4c9c7b872455
e481f2d1726e7d1bd1b83e4bc1b87da16eea9fc19a1f1ab05726f6cbfa212ec9
ee38d7a2bf4f5a380c55639757d268eb2f9858a4d3a68fce470be9f5ca62921f
1d2024edfa2d14a                                                 5
558256a6df8c5ce                                                 e
8a30dfddc0c1a37                                                 9
f3e25eaa00cb18c                                                 0
40be2fcdf046969                                                 8
d27673b9ad94e5e                                                 b
57e9e20008f8f3b0ac5cd275f5d3e8a269152ecd1381d78268f2c909eec33ad9
7d017ac832d567efb75eb59e4575403af510f07e729915967227433deae40ed
04023972b96ee43f893678d2d1df7e06473cacf7256e36fa3ffc81dc93af19c
3980352df57f49844b17c219941d926ae373f9cc6bfb6939cd4a951e06a76ffb
ff8b83165607c815b4d3659ee9b8485925da989c7cbfe0e7a58650c9de2842b6
e54ffbe0e94f3ddcd95b3696acd8f6334812eab6e2e2a895a1a8402f9147d30d
1bcc0aff1986a654fd188e056e5884a8945391a7a17352b16a4e3f514d0e4a21
842fe3381abf4ce12cb3f18065a4125cb04e281cb3f66243ec1f99ee7ecb25a6
159459677c922017e1736ee90d349b906e214b5fd4aec2724c72c24dec2a228f
a8e8b916b6d2b3e5fa4711b490d033745c2aea4a278396c6854da72970599491
5605b89c3bfc23b6f580d673720cbd8740424f76e8791961e433bf11f07399f4
==============================================================
[INFO] Encryption End
```

암호화 검증

```
[INFO] Decryption Start
[INFO] send done
[INFO] receive don
========================= Message =========================
534d4152544d324d53434d2d4b59424552353132534d2d4b59424552353132
==============================================================
[INFO] Decryption End
```

복호화 검증

SCM-Kyber PEK 키생성/암호화/복호화 결과 출력 화면

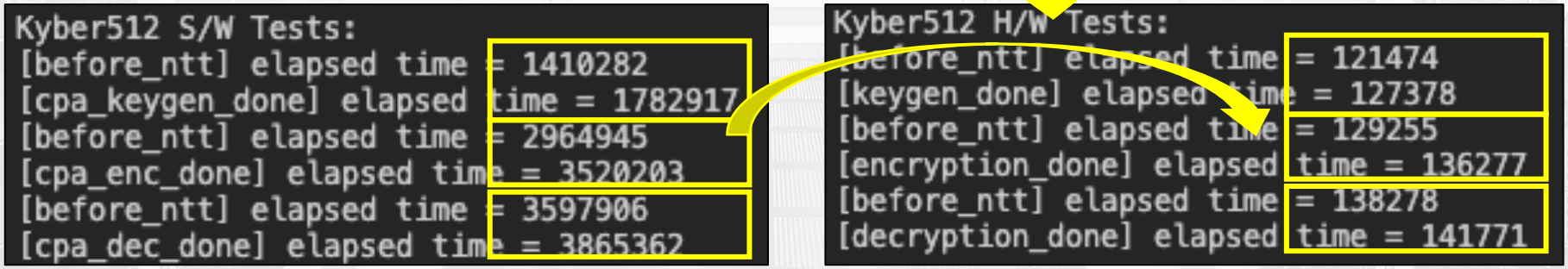
SmartM2M 스마트엠투엠

부록

스마트엠투엠 Kyber-512 하드웨어(FPGA) 구현 사례 (성능 비교)

- **주요 구현 특징**
  - 저사양 디바이스(RISC-V) 소프트웨어 대비 하드웨어 성능 향상 비교
    - 저사양 RISC-V(Ibex) 기준 Kyber SW 및 하드웨어 가속기(HW) 수행 시간 비교
    - 저사양 디바이스에서 Kyber SW 개발 대비 48~79배 성능 향상 (전력 소모량 및 성능 우수)
  - 기능별 SW - HW 성능 비교 (사이클)
    - KEY GEN(약 63배 향상) : SW(372,635) → HW(5,904)
    - CPA_ENC(약 79배 향상) : SW(555,258) → HW(7,022)
    - CPA_DEC(약 48배 향상) : SW(267,456) → HW(5,494)

하드웨어 기반 가속기 사용시
48~79배 성능 향상

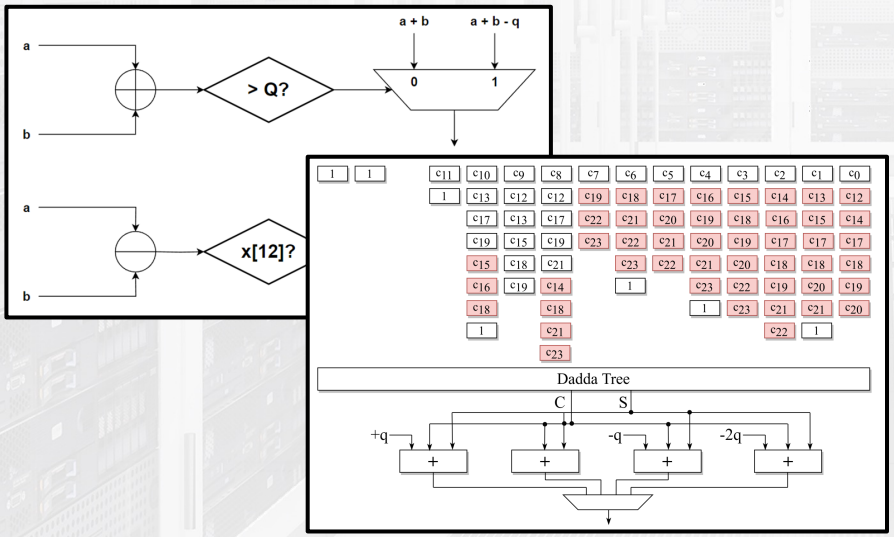


SCM-Kyber 기능별 성능 비교

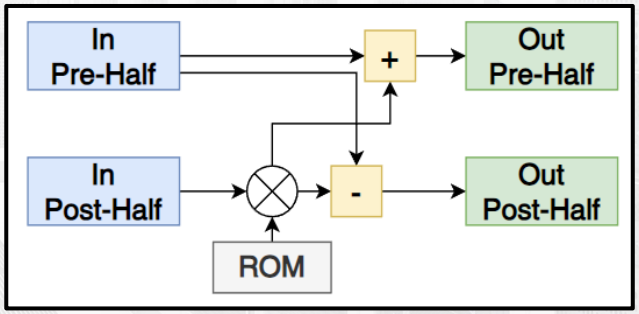
SmartM2M 스마트엠투엠

Kyber 하드웨어 구현 고려사항

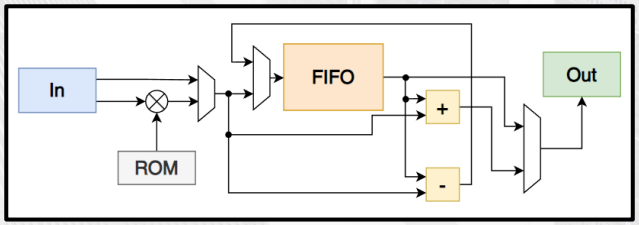
- 하드웨어 최적화 구현 특징
  - 하드웨어 구현 특성을 이용한 Reduction 연산의 경량화 및 최적화
    - 모듈러 ADD / SUB 시 곱셈기를 포함한 Barrett Reduction 대신 비교기를 활용한 Reduction 활용
    - 모듈러 MUL 시 곱셈기를 포함한 Montgomery Reduction 대신 덧셈기만 이용한 Reduction (Dadda Tree Reduction)
  - FIFO를 활용한 NTT 제어로직 경량화
    - NTT 연산에 필요한 데이터 순서 제어를 FIFO를 통해 순서를 정렬하여 제어 로직 경량화



SCM-Kyber 적용 Reduction 연산



NTT 데이터 흐름도



NTT 모듈 구조도

Yaman, F., Mert, A. C., Öztürk, E., & Savaş, E. (2021, February). A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme. In 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1020-1025). IEEE.

SmartM2M 스마트엠투엠

부록 - JavaCard 환경에서의 PQC 구현

JavaCard 구현 특징 및 PQC 구현 고려사항

- JavaCard 개요
  - JavaCard는 Java기반 애플리케이션(Applet)을 SIM 카드, RFID카드 메모리가 제한적인 IoT 디바이스 내에서 구동될 수 있도록 함
  - 이와 같이 JavaCard는 초경량의 컴퓨팅 환경에서 작동되기 때문에 알고리즘 및 코드, 메모리 등의 최적화가 필요함
- JavaCard 특성 및 PQC 구현 고려사항
  - Java의 표준 라이브러리 사용이 제한적이기 때문에 Double, Int, Float과 같은 자료형 사용 불가
    → Byte(1byte), Short(2byte), Boolean 데이터형 사용 가능
  - 경량화된 환경에 최적화 설계된 JavaCard는 병렬 연산 및 스레드 사용이 불가함
  - byte[2][5] Array과 같은 다차원 배열의 사용이 불가함
    → byte[10] Array와 같이 1차원 배열로 변환 필요
  - JavaCard Applet과 통신을 위한 APDU는 한번에 최대 255바이트까지의 제한된 I/O 처리만 가능함
    → 1. 긴 데이터는 작은 Chunk로 분할하여 여러 번에 나누어 전송
    → 2. Extended APDU를 통해 데이터 전송 (카드에서 Extended APDU 지원 및 JavaCard 2.2.2 버전 이상 필요)
  - JavaCard 버전에 따라 제공되는 암호 알고리즘 Native Method가 다르기 때문에 확인 필요
    → Native Method에서 H/W 가속기를 지원하는 경우, 월등히 좋은 연산 수행속도를 보임

Table 14. Kyber 구현에 필요한 Symmetric Primitives와 지원하는 JavaCard 버전

Scheme	Symmetric Primitives	Algorithms	Java Card Version
Kyber	XOF	SHAKE-128	-
	H	SHA-3-256	Since 3.0.5
	G	SHA-3-512	Since 3.0.5
	PRF	SHAKE-256	-
	KDF	SHAKE-256	-
Kyber "90s" Variant	XOF	AES-256-CTR	Since 3.0.5
	H	SHA-256	Since 2.1.2
	G	SHA-512	Since 2.2.2
	PRF	AES-256-CTR	Since 3.0.5
	KDF	SHAKE-256	-

SmartM2M 스마트엠투엠

부록 - JavaCard 환경에서의 PQC 구현

JavaCard 구현 특징 및 PQC 구현 고려사항

- CRYSTALS Kyber
  - Kyber에서 사용되는 SHAKE 해시 알고리즘은 현재 JavaCard 최신버전에서도 지원하지 않음
  - 초경량 환경 특성상, SHAKE 알고리즘 직접 구현 및 사용은 매우 낮은 성능을 보이기 때문에 권장되지 않음
  - JavaCard에 AES, SHA2 암호화 하드웨어 가속기가 있는 경우, Kyber 90's로 대체하여 성능 최적화 기대 가능
    - Kyber 90's는 기존 SHAKE(XOF) 및 SHA3 알고리즘을 AES(XOF) 및 SHA256, SHA512로 대체함
- NTRU+
  - NTRU+는 기본적으로 XOF로써 AES256-CTR 알고리즘과 SHA-256 및 SHA-512을 사용함 (v1.0 버전 기준)
- PQC 구현 및 성능평가 환경
  - PQC 암호 알고리즘 구현 및 성능평가 환경은 다음과 같음

JavaCard 버전	3.0.5
CPU 동작 클럭	약 100Mhz
RAM	약 10KB
EEPROM	약 32KB

  - NTRU+ v1.0에 대한 성능평가 결과

암호 알고리즘 / 버전	NTRU+ / v1.0
키 생성(Key Generation)	약 12 ~ 13 sec
암호화(Encapsulation)	약 14 ~ 15 sec
복호화(Decapsulation)	약 9 ~ 10 sec

SmartM2M 스마트엠투엠

부록 - JavaCard 환경에서의 PQC 구현

JavaCard 구현 특징 및 PQC 구현 고려사항

- JavaCard는 초경량 컴퓨팅 환경이기 때문에 일반적인 환경과는 다른 개발 방법론이 필요
  - 객체 재사용을 통한 메모리 사용 최적화, Class 상속 및 배열 변수 접근 최소화 등

구분	내용
최적화된 Applet 설계	• Java언어의 객체 지향 특성, 클래스, 상속 등을 활용하여 공통 코드의 최소화 및 재사용성 극대화 필요 • 너무 많은 클래스와 메소드를 구현시 메모리 사용량이 많아지기 때문에 Applet을 Compact하게 구현 필요
암호연산 수행시간 최적화	• EEPROM은 RAM과 비교하여 메모리 접근 속도가 약 1,000배까지 느릴 수 있음 • 암호 연산 내부에서 자주 사용하는 변수는 Transient Array(in RAM)를 사용
객체 재사용	• 메소드내에서 지역변수 선언을 최소화할 수 있도록 함 • Key, Message 등과 같이 중복되어 사용되는 객체는 클래스 필드 변수로 선언하여 최대한 재사용 가능하도록 구현
배열 변수 접근 및 복사	• Array에 특정 Element에 반복적으로 접근하는 것보다 해당 Element를 지역 변수에 저장하여 접근하는 것이 더 효율적임 • 원자성 보장이 필요하지 않은 경우, arrayCopyNonAtomic, arrayFillNonAtomic 함수를 사용하여 배열을 복사하도록 함
Switch 문법 사용	• 일반적으로 Switch문이 if-else문보다 적은 메모리를 사용하고 빠름
복합 산술 연산 사용	• 산술연산 시, 하나의 복합문으로 나타내는것이 메모리(스택), 바이트코드 크기 측면에서 효율적임 (예시) x= a+b; x=x-c; → x=a+b-c;

SmartM2M 스마트엠투엠