



HAETAE
HEON
CRYPTOLAB

격자기반 양자내성 전자서명 알고리즘 & 참조 구현 투어

KpqC Winter Camp 2026

2026.02.25.

(주) 크립토랩

김태경

오늘 발표의 구성

HAETAE: 격자기반 양자내성 전자서명 알고리즘 & 참조 구현 투어

01

전자서명은 무엇이고, 양자내성 전자서명이 필요한 이유는 무엇일까?

02

HAETAE 알고리즘 개괄, ML-DSA와는 어떤 차이가 있을까?

03

참조코드 walk-through: KeyGen, Sign, Verify API

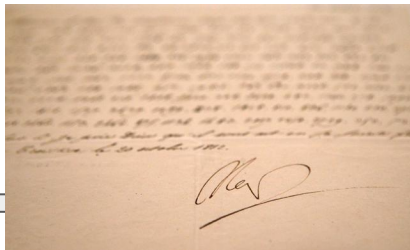
04

파라미터와 실제 사용 가이드라인



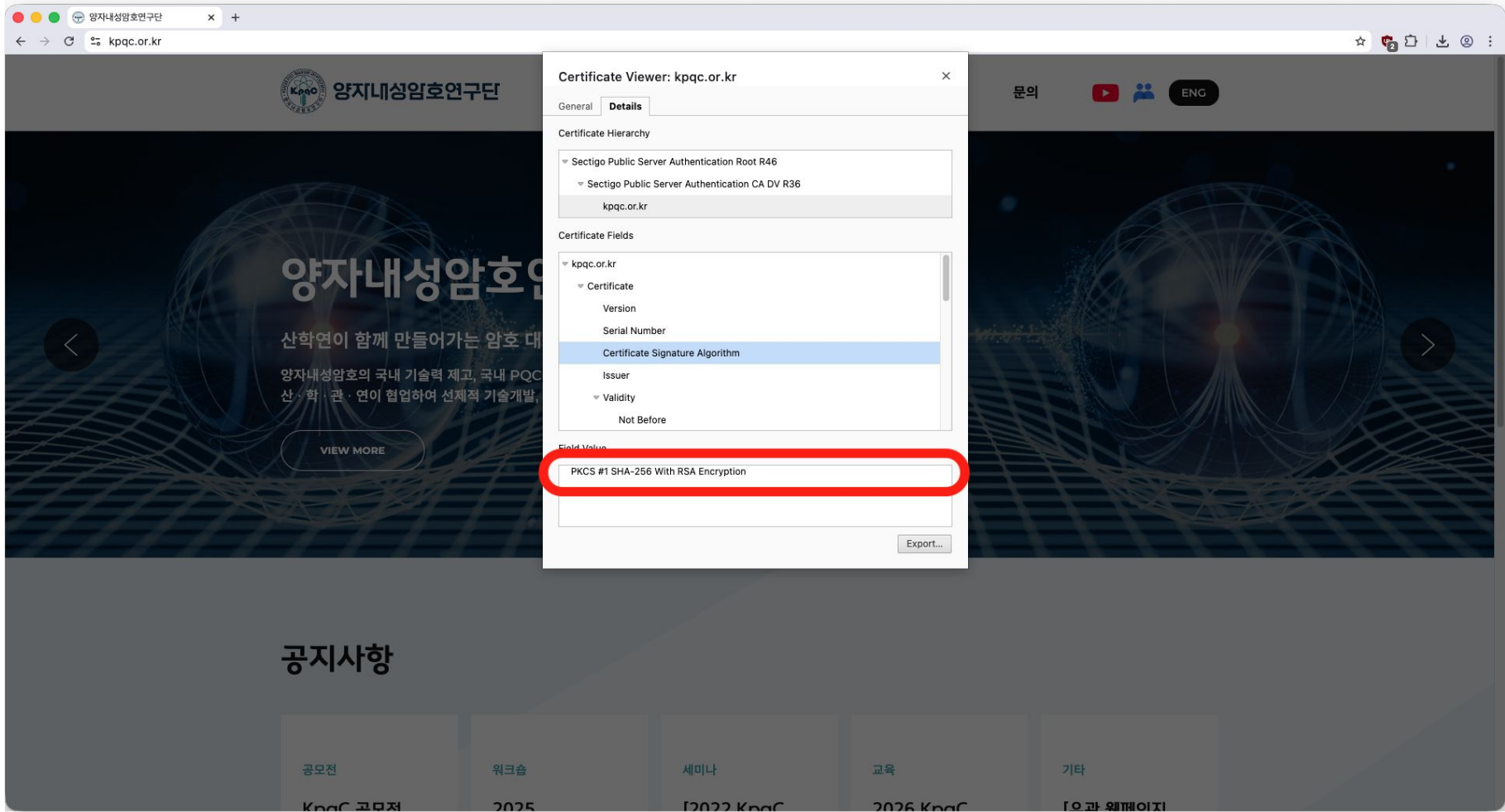
Alice

Message



Bob

인증	서명을 검증함으로써 메시지를 보낸 사람의 신원 (Alice)을 확인.
위조 불가	합법적인 서명자 (Alice) 만이 이 서명을 만들어 낼 수 있음.
부인 방지	Alice는 추후에 본인이 서명한 것이 아니라고 발뺌할 수 없음.
무결성 보장	메시지는 서명 당시와 단 한 글자의 차이도 없음.
재사용 불가	서명은 이 메시지만을 위해 만들어진 것임.



전자서명, 지금은 안전할까?

전자서명이 쓰이는 곳

- TLS 인증서
- 코드서명 / 펌웨어 업데이트
- 전자문서 / 공인인증서
- JWT / OAuth 토큰
- 블록체인 트랜잭션

현재, 위의 모든 곳에서 대부분
Elliptic Curve / RSA 기반 전자서명 사용 중
→ 양자컴퓨터로 위조 서명 생성 가능!

01

서명 위조 가능

Shor 알고리즘으로 서명 비밀 키 복구
→ 임의 문서에 유효한 서명 생성

02

인증서 신뢰 붕괴

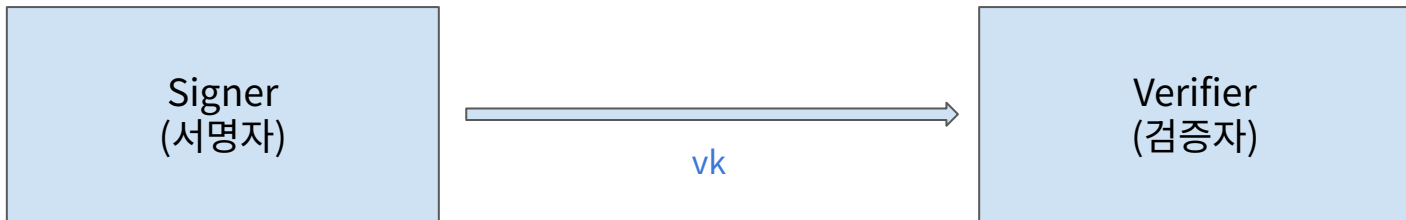
CA 서명 위조 → 가짜 인증서 발급
→ HTTPS, 코드 서명 체계 전체 붕괴

03

장기 무결성 위협

현재 서명된 계약·문서도
양자컴퓨터 등장 시 재검증 불가!

전자서명 알고리즘과 HAETAE



$(sk, vk) \leftarrow \text{KeyGen}()$
 $s \leftarrow \text{Sign}(sk, \text{message})$

$T/F \leftarrow \text{Verify}(vk, \text{message}, s)$

ML-DSA
(Crystals-Dilithium)

NIST 표준 (FIPS 204)

HAETAE

KpqC 한국 표준 후보,
국내 공공기관·금융·의료
섹터 등에 최적화 (인증 등)

주요 공통점과 차이점

공통점

- 동일한 격자 수학 난제 기반

차이점

- (HAETAE) 다양한 고급 최적화를 통해 키/서명 사이즈 감소



HAETAE
HEAAN
CRYPTO LAB

크립토헤 & 서울대학교 & 프랑스 리옹 고등사범학교 & 독일 보훔 루르 대학교 & 독일 AI 연구센터 (DFKI)

서울대학교
SEOUL NATIONAL UNIVERSITY



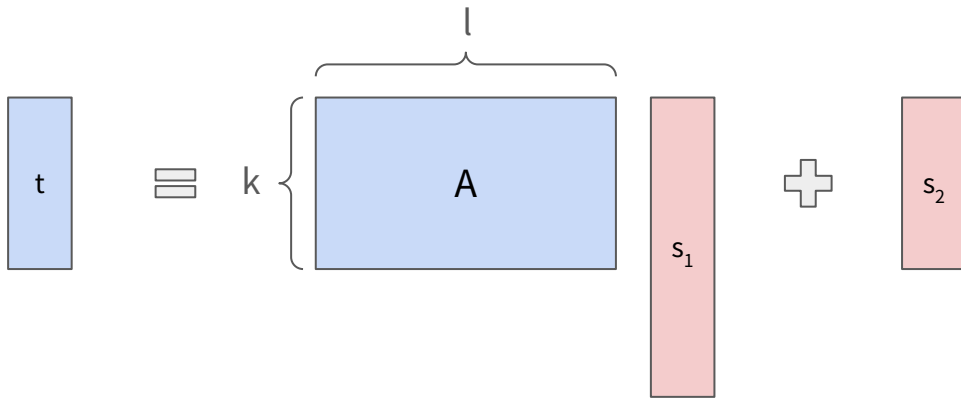
HAETAE (& ML-DSA) 의 수학적 기반

Module-LWE (MLWE) 문제

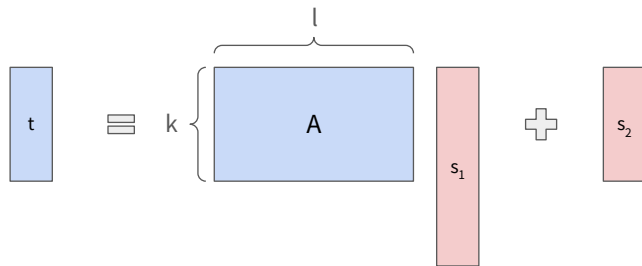
$$R_q = \mathbb{Z}_q[X] / (X^N + 1)$$

- 행렬 $A \in R_q^{k \times l}$
- 벡터 $s_1 \in R_q^l$ (계수가 작은 다항식들)
- 벡터 $s_2 \in R_q^k$ (계수가 작은 다항식들)

이 주어져 있을 때, $t = A \cdot s_1 + s_2$ 와 A 를 공개해도 s_1 과 s_2 를 알아내기는 어려움.



HAETAE 전자서명 알고리즘: 1 키 생성



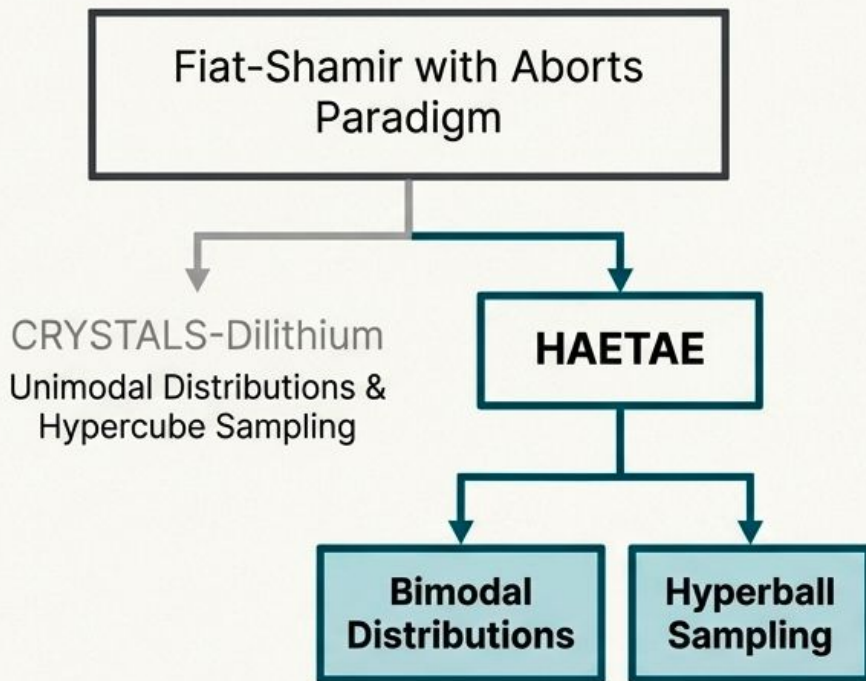
$sk = (s_1, s_2, A) \rightarrow$ 서명 생성에 사용

$vk = (A, t) \rightarrow$ 서명 검증에 사용

HAETAE 전자서명 알고리즘: 2 서명 생성

입력값	m	서명할 메시지
	$sk = (s_1, s_2, A)$	서명키
중간 계산	$y \in \mathbb{R}_q^l$	마스킹 벡터: 무작위로 생성된 작은 벡터
	$c = H(w_1, m)$	챌린지 다항식, $w_1 = A \cdot y$ 의 상위 비트
	$z = y + s_1 \cdot c$	서명 벡터 (후보)
기각 샘플링	서명 벡터 z 가 특정 조건 (충분히 작은 지 등) 만족하는지 확인! 만족 못하면 다시 처음부터...	
최종 출력값	(z, c)	최종 서명

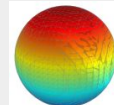
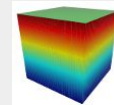
HAETAE vs ML-DSA



Hyperball Sampling

마스킹 벡터 y 를 어디서 뽑아 오는가?

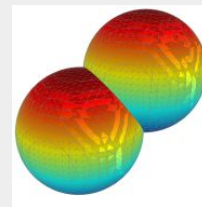
- ML-DSA: hypercube
- HAETAE: hyperball



Bimodal Distribution

서명 벡터 z 의 중심은 몇 개?

- ML-DSA: 1개, $z = y + s_1 \cdot c$
- HAETAE: 2개, $z = y + (-1)^r \cdot s_1 \cdot c$



HAETAE 전자서명 알고리즘: ③ 서명 검증

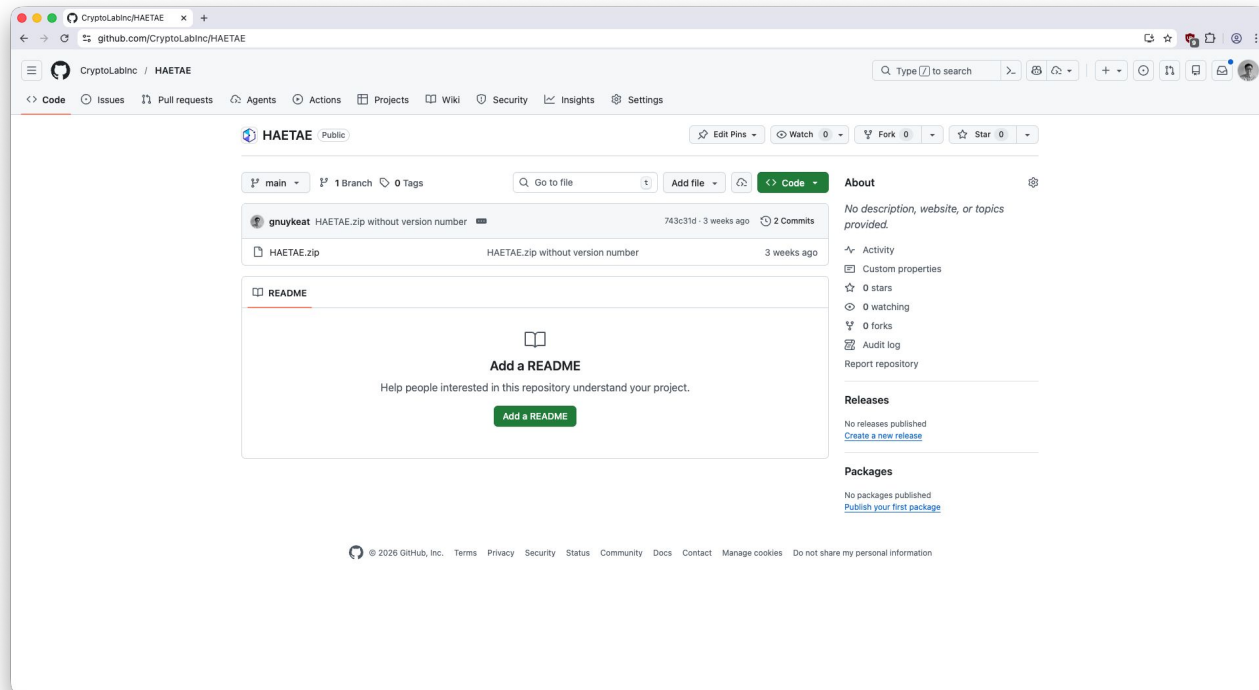
입력값	(z, c)	서명
	m	메시지
	$vk = (A, t)$	검증키
중간 계산	$w' = A \cdot z - t \cdot c$	재구성
	w_1'	복원, w' 의 상위 비트.
검증 & 출력값	$(z \text{가 충분히 작은가?}) \ \&\& \ (c == H(w_1', m))$	

검증이 가능한 이유

$w' = A \cdot z - t \cdot c = A \cdot (y + s_1 \cdot c) - (A \cdot s_1 + s_2) \cdot c = A \cdot y - s_2 \cdot c$
 $\rightarrow w'$ 과 w_1 은 상위 비트에서 정확히 동일!

HAETAE 참조 코드 walk-through

<https://github.com/CryptoLabInc/HAETAE>



핵심 API

KEYGEN

```
int crypto_sign_keypair(uint8_t* vk, uint8_t* sk)
```

검증키/서명키 쌍 생성

SIGN

```
int crypto_sign(...); int crypto_sign_signature(...);
```

서명 생성. `crypto_sign`은 서명과 메시지를 병합, `crypto_sign_signature`는 서명만 생성.

VERIFY

```
int crypto_sign_verify(...)
```

서명 검증.


```
int sign_verify_test(void) {
    if (print_flag) {
        printf(">> sign and verify test\n");
        print_flag = 0;
    }

    uint8_t pk[CRYPTO_PUBLICKEYBYTES] = {0};
    uint8_t sk[HAETAECRYPTO_SECRETKEYBYTES] = {0};

    crypto_sign_keypair(vk: pk, sk);

    size_t siglen = 0;
    uint8_t sig[CRYPTO_BYTES] = {0};
    uint8_t msg[HAETAECRYPTO_SEEDBYTES] = {0};
    randombytes(out: msg, outlen: HAETAECRYPTO_SEEDBYTES);
    crypto_sign_signature(sig, siglen: &siglen, m: msg, mlen: HAETAECRYPTO_SEEDBYTES, ctx: NULL, ctxlen: 0, sk);

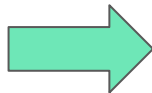
    if (crypto_sign_verify(sig, siglen, m: msg, mlen: HAETAECRYPTO_SEEDBYTES, ctx: NULL, ctxlen: 0, vk: pk)) {
        return 1;
    }

    return 0;
}
```

기존 ECDSA → HAETAE 교체 예시

BEFORE (ECDSA, OpenSSL)

```
// ECDSA 서명 (OpenSSL)
EC_KEY *key =
EC_KEY_new_by_curve_name(
    NID_X9_62_prime256v1);
EC_KEY_generate_key(key);
ECDSA_sign(0, hash, hashlen,
    sig, &siglen, key);
ECDSA_verify(0, hash, hashlen,
    sig, siglen, key);
// 양자 취약! ⚠
```



AFTER (HAETAE)

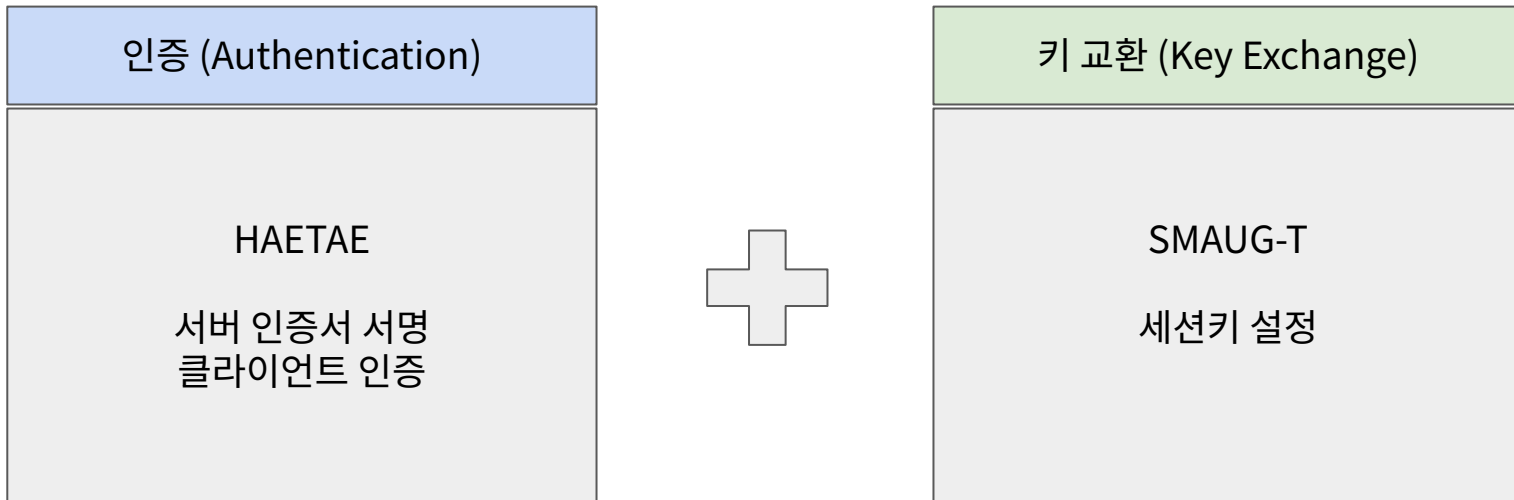
```
// HAETAE 서명 교체
crypto_sign_keypair(vk, sk);

crypto_sign_signature(
    sig, &siglen,
    msg, mlen, NULL, 0, sk
);
int r = crypto_sign_verify(
    sig, siglen,
    msg, mlen, NULL, 0, vk
);
// 양자 안전 ✓
```

파라미터, 실제 사용 가이드라인

HAETAE + SMAUG-T: 완전한 PQC PKI

PQC 완전 전환 TLS 구성



결과: 양자 내성 TLS — 인증도 안전, 키 교환도 안전

KpqC 표준으로 국내 공공·금융 규제 대응 완비!

HAETAE Parameters

HAETAE-2

Targets NIST Security Level 2
(~128-bit classical security)

HAETAE-3

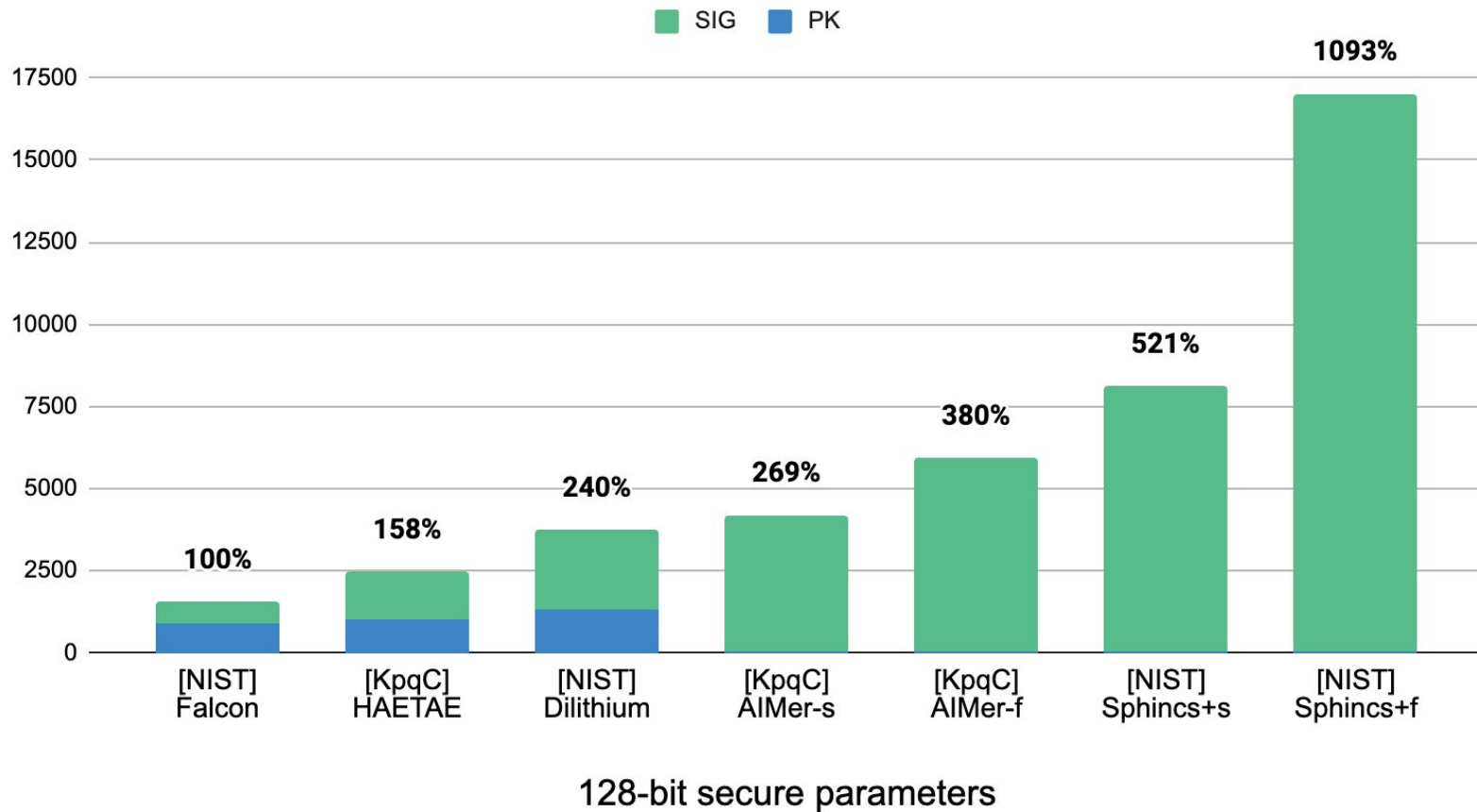
Targets NIST Security Level 3
(~192-bit classical security)

HAETAE-5

Targets NIST Security Level 5
(~256-bit classical security)

Parameters	MODE = 2	MODE = 3	MODE = 5
Polynomial Dimension (N)	256		
Polynomial Modulus (Q)	64,513		
K	2	3	4
L	4	6	7
Verification Key Size (Bytes)	992	1,472	2,080
Signature Key Size (Bytes)	1,408	2,112	2,752
Signature Size (AT MAX) (Bytes)	1,474	2,349	2,948

Size in bytes



감사합니다!

Contact: 김태경
(taekyung.kim@cryptolab.co.kr)

