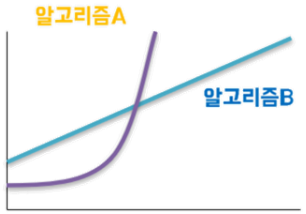


KpqC 알고리즘 성능 분석 및 최적화 동향

심민주, 송경주, 엄시우, 이민우, 서화정

2024-10-23

양자내성암호연구단 KpqC 그리고 KpqClean 소개

최신 업데이트 기준 알고리즘 벤치마크

다양한 측면에서의 성능 비교 분석

KpqC

- **양자 내성 암호 연구단**은 양자컴퓨터에 의한 보안 위협에 대비하기 위해 **양자내성암호 국가공모전 (KpqC 공모전) 진행 중**

- **2라운드 진출 알고리즘**
  - **Digital Signature : AIMer, HAETAE, MQ-Sign, NCC-Sign**
  - **Encryption/Key-establishment : NTRU+, PALOMA, REDOG, SMAUG-T**

KpqCm	
2021.11.25	공모전 공지
2022.02.18	제출마감
2022.03.15	1차 평가
2023.12	1라운드 결과 발표 2라운드 후보 목록 공개
2024.04~11	**2라운드 진행**
2024.11	KpqC 공모전 최종결과 발표 예정

https://www.kpqc.or.kr/competition_02.html

•Digital Signature Algorithm: 4개
•Encryption/Key-establishment Algorithm : 4개

Type	Code-based	Lattice-based	Multivariate Quadratic-based	Hash-based	MPC
Digital Signature	-	HAETAE	MQ-Sign	-	AIMer
		NCC-Sign			
Encryption/ Key-establishment	PALOMA	NTRU+	-	-	-
	REDOG	SMAUG-T	-	-	-

NIST PQC공모전과 유사하게 격자기반이 가장 많이 후보군에 오름

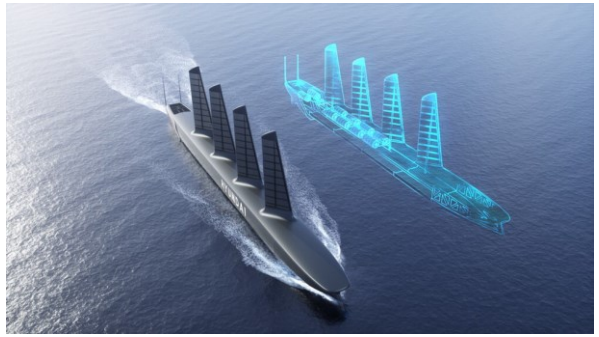
KpqC에 대한 벤치마크 KpqClean

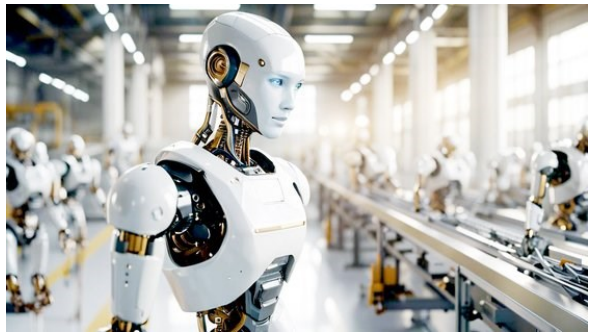
- **KpqC 후보 알고리즘들에 대한 벤치마킹**
  - 현재 각 알고리즘이 제시하는 벤치마크 결과는 **각기 다른 컴퓨터 환경에서 측정**
  - **동일한 환경에서 알고리즘들을 벤치마크**하여 객관성을 확보하는 것이 목표
  - 내부 모듈 (AES, SHA, AVX2)을 통일하여 모듈에 따른 성능 차이를 최소화
- **공정한 벤치마크 결과 제공**
  - **개발 및 운영 환경 차이, 모듈 차이로 인한 성능 격차 최소화**
  - 동일한 환경에서 측정한 성능으로 공정한 성능 비교 분석이 가능하도록 함
- **KpqC 후보 알고리즘의 의존성 제거**
  - NIST 양자내성암호와 관련된 PQClean 프로젝트에서 추구한 방향성
  - **의존성 제거 시 플랫폼에 구애받지 않고 알고리즘 가동 용이**

KpqClean에서 진행한 벤치마크

	Kpqc (KpqClean)	NIST PQC (pqClean)
공모전 기간	2022~2024	2017~2024
참여연구자	한국인 + 제한적인 외국인	전 세계인
벤치마크 플랫폼	고성능	고성능
벤치마크 언어	C, AVX2	C, AVX2

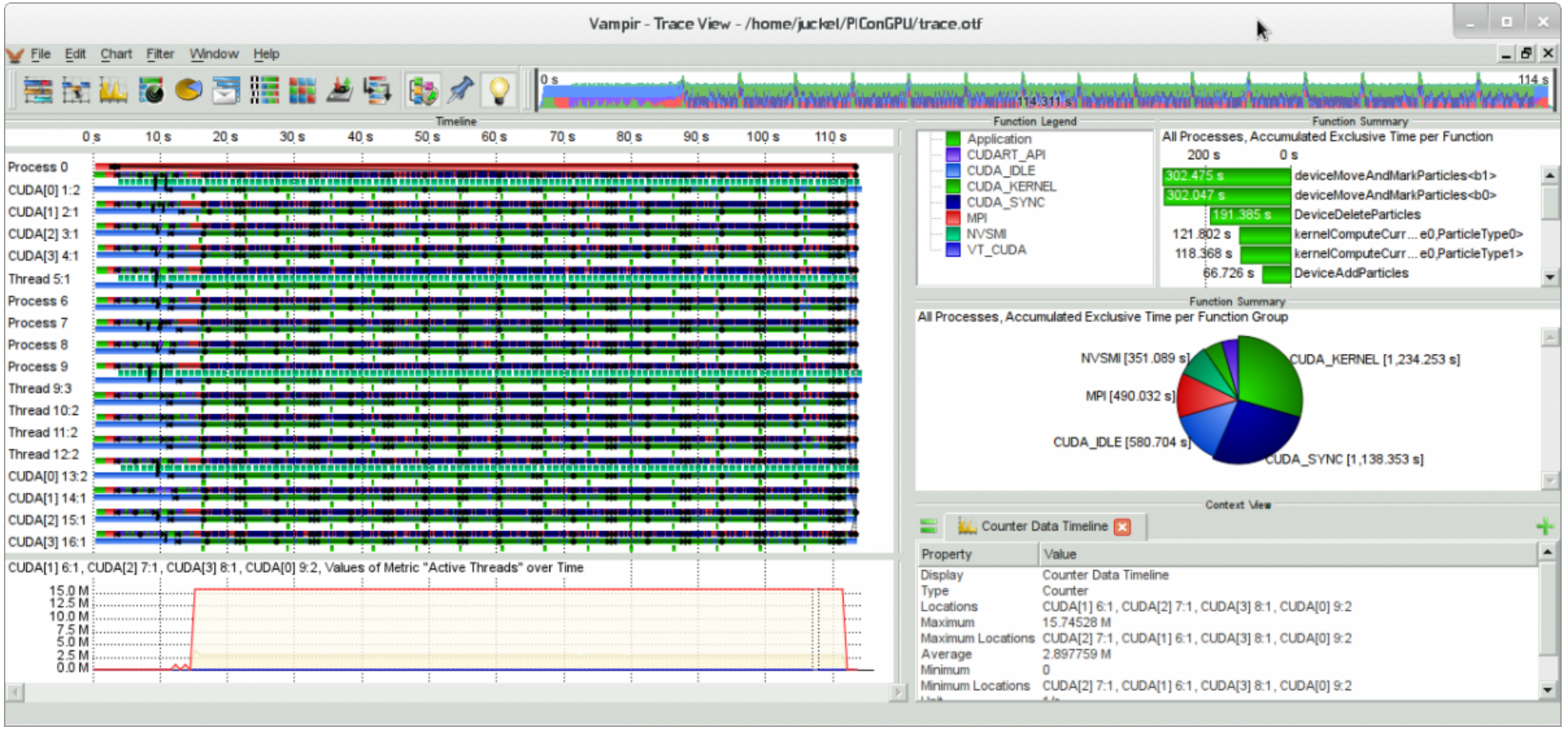
- **KpqClean 프로젝트를 통해 상대적인 성능 차이 도출에 집중**
  - **절대적인 최적화 성능 지표 도출**은 알고리즘 선정 이후에 지속적 추진 필요
  - 이기종의 **컴퓨팅 환경에 맞춘 최적화 구현 기술** 연구 필요



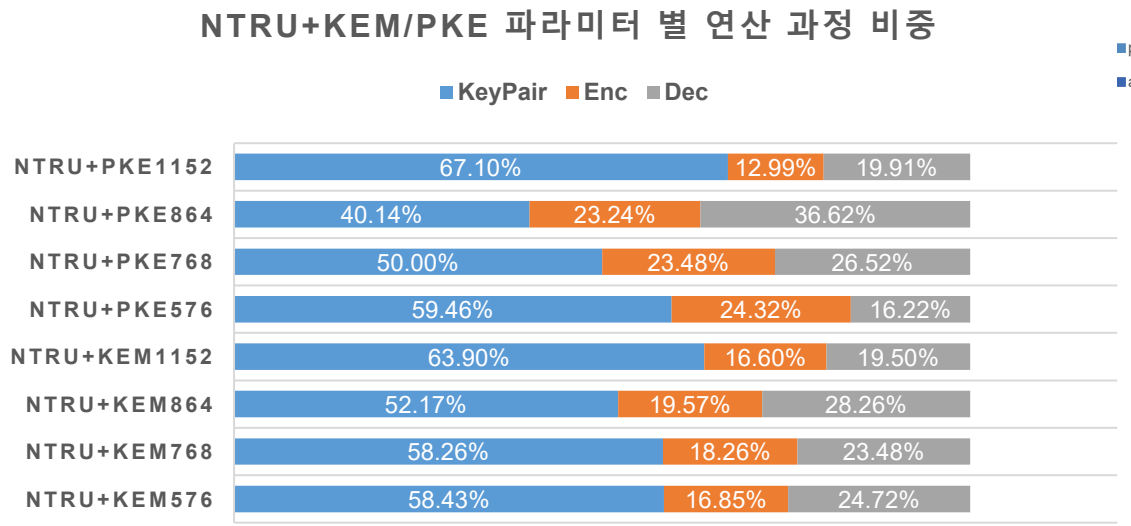
KpqC 2라운드 알고리즘 연산 시간 프로파일링 분석

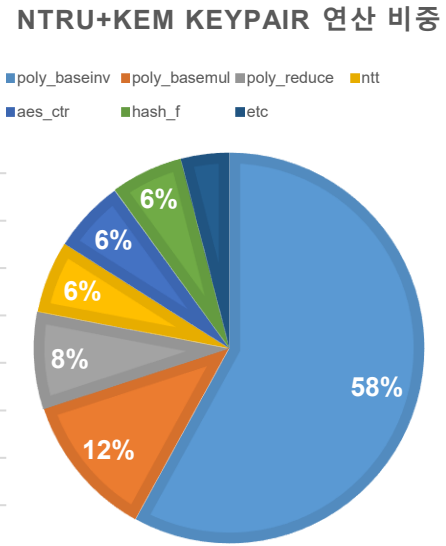
- **KpqC 알고리즘들에 대한 프로파일링 및 최적화 요소 확인**
  - KpqC 알고리즘들마다 **큰 연산 오버헤드가 발생하는 공통적 혹은 상이한 부분 존재**
  - 확인된 오버헤드에 집중함으로써 **효율적으로 알고리즘 성능 개선 가능**

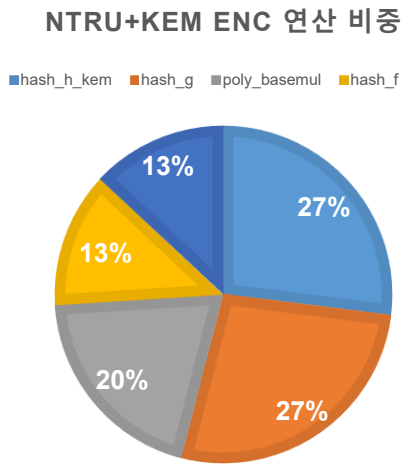


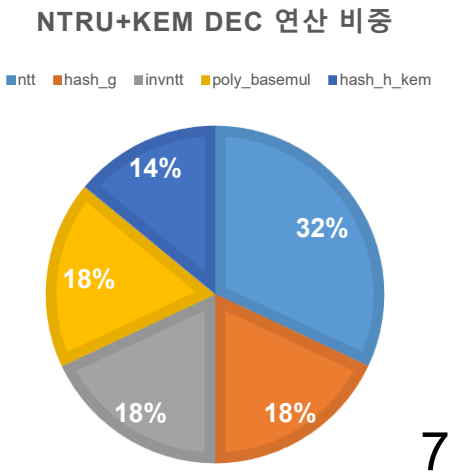
(KEM) NTRU+ 알고리즘 연산 분석 결과

- **NTRU+에서 가장 큰 비중을 차지하는 과정은 키 생성 과정**
  - 키 생성 과정에서는 NTT 도메인 상에서 **역수를 구하는 연산 과정**이 많은 비중 차지
  - 캡슐화 과정에서는 **SHA 연산**이 많은 비중 차지
  - 디캡슐화 과정에서는 **NTT 연산 과정**이 많은 비중 차지
  - 전체적으로 NTRU+ 알고리즘에서 많이 사용되는 연산은 **NTT 관련 연산**
  - 키 생성 과정이 가장 큰 비중을 차지 하기 때문에 **poly_baseinv 연산**의 비중이 큼
    - NTRU+ PKE의 경우 KEM과 비슷한 연산 비중을 보여주고 있음.





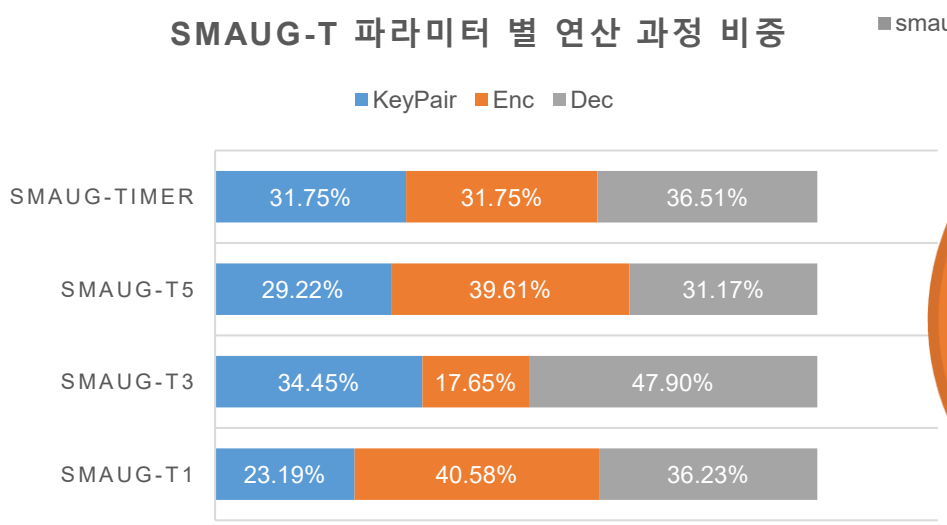


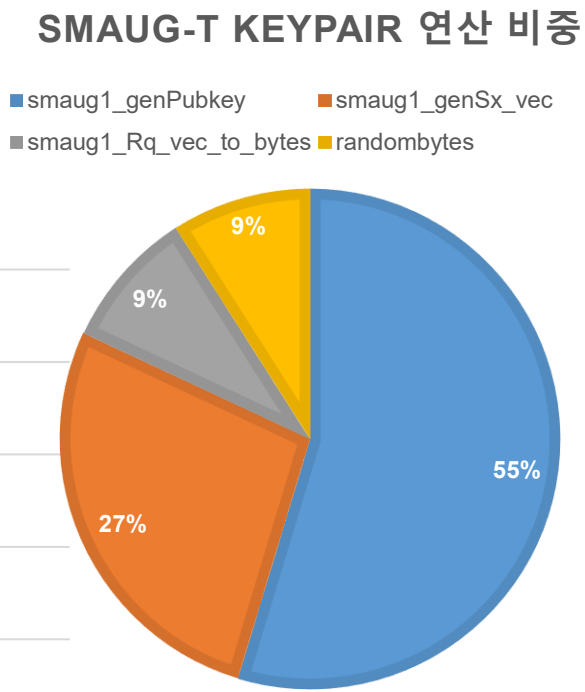


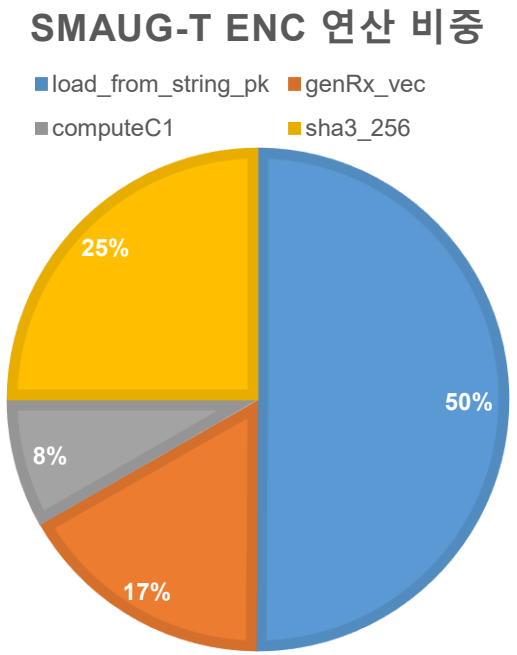
(KEM) SMAUG-T 알고리즘 연산 분석 결과

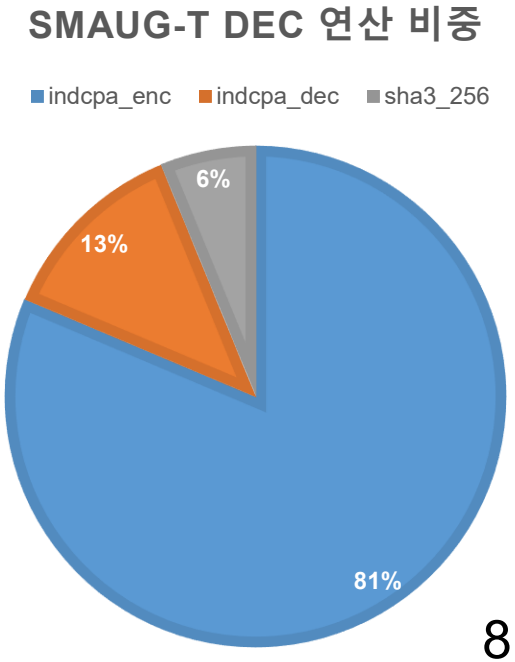
- **SMAUG-T에서 큰 비중을 차지하는 과정은 캡슐화와 디캡슐화 과정**
  - 모든 과정에서 **SHAKE 관련 연산**이 가장 큰 비중을 차지

  - 전체적으로 SMAUG-T 알고리즘에서 많이 사용되는 연산은 **SHAKE 관련 연산**
    - 다만 벡터간의 곱셈 후 덧셈 연산(Vec_mult_add)도 많이 사용되고 있음



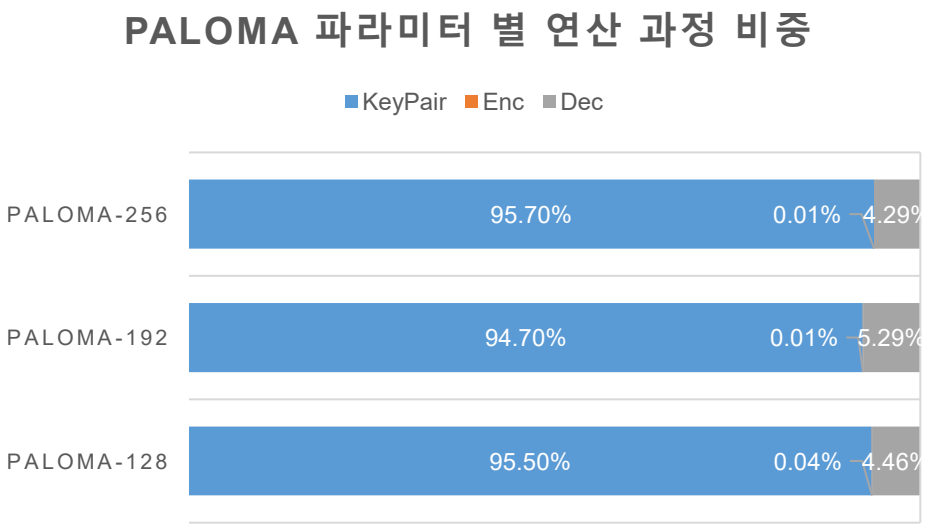


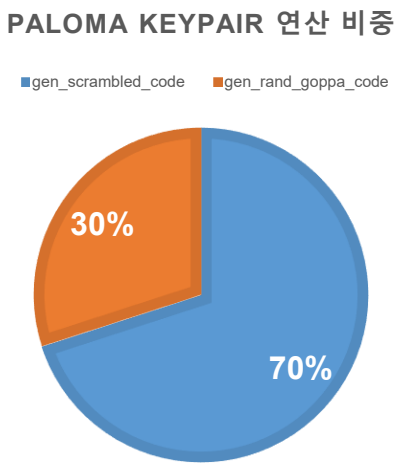


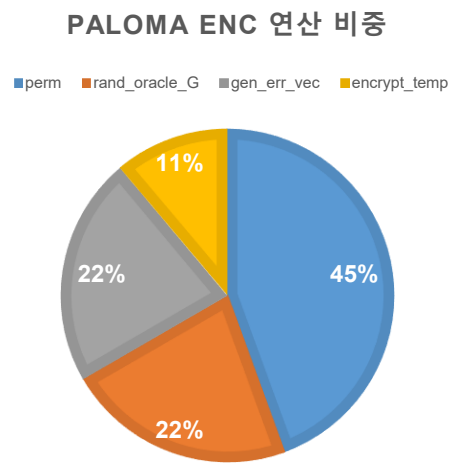


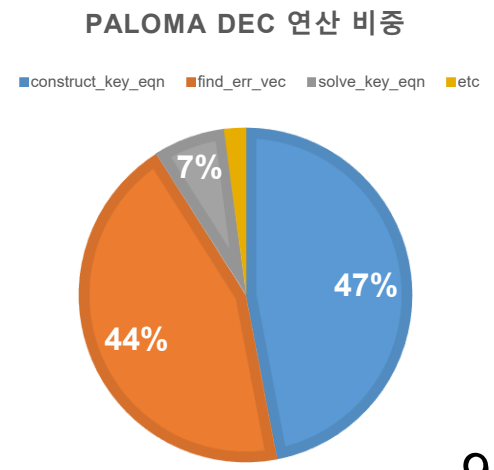
(KEM) PALOMA 알고리즘 연산 분석 결과

- **PALOMA에서 가장 큰 비중을 차지하는 과정은 키 생성 과정**
  - 키 생성 과정에서는 많은 반복 논리 연산을 하는 **Scrambled_code 과정**이 가장 많은 비중을 차지
  - 캡슐화 과정에서는 에러 벡터를 순열하는 **Permutation 연산 과정**이 가장 많은 비중을 차지
    → 에러 벡터 생성, 랜덤 오라클 G 생성 **(LSH 해시 알고리즘 사용)**
  - 디캡슐화 과정에서 **에러 벡터 e를 복원하는 과정**이 많은 비중을 차지 **(다항식 산술 연산 사용)**
  - 전체적으로 PALOMA 알고리즘에서 **키 생성 과정이 큰 비중**을 차지하며 키 생성 과정의 **Scrambled_code 과정에서 많은 반복 논리 연산**이 많은 비중 차지



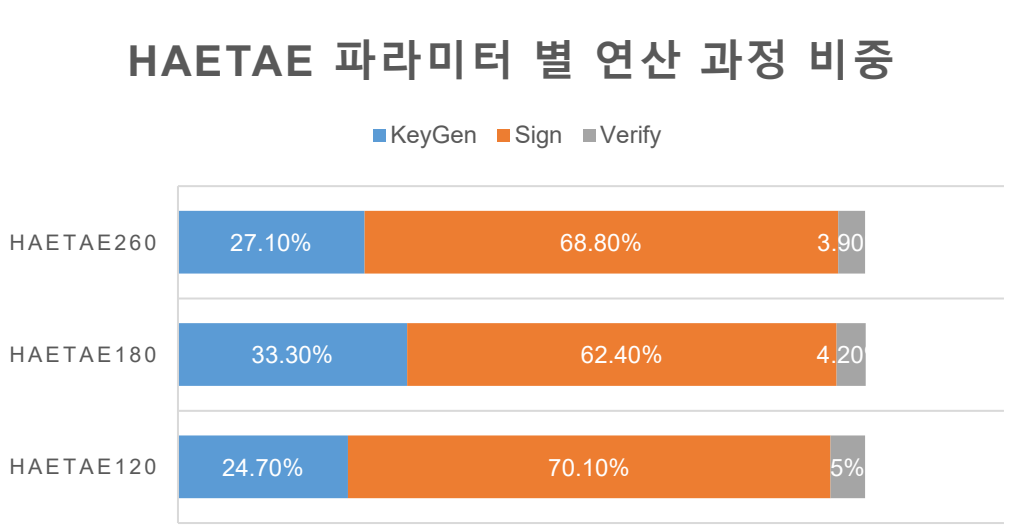


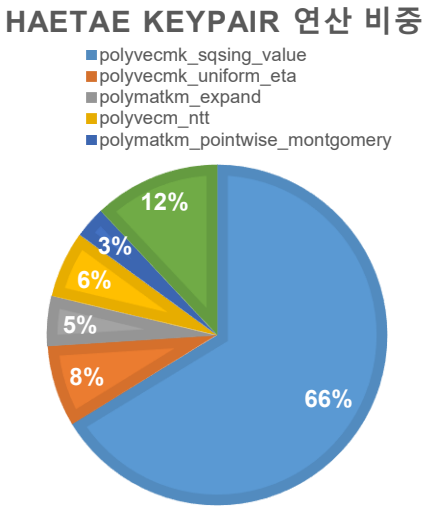


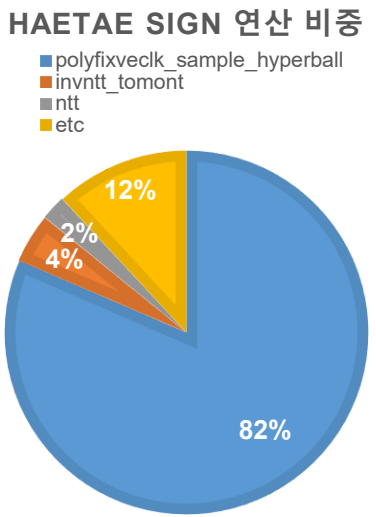


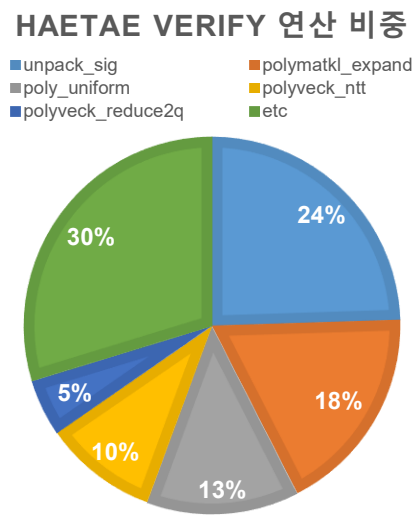
(DSA) HAETAE 알고리즘 연산 분석 결과

- **HAETAE에서 가장 큰 비중을 차지하는 과정은 서명 과정**
  - 서명 과정에서는 **난수 생성 과정**이 큰 비중을 차지하며, 그 과정에서 **SHAKE 관련 연산**이 많이 사용
  - 키 생성 과정에서는 **FFT 연산과 SHAKE 관련 연산**이 많이 사용
  - 검증 과정에서는 **Decode 과정과 SHAKE 관련 연산**이 많이 사용
  - 전체적으로 HAETAE 알고리즘에서 많이 사용되는 연산은 **SHAKE 관련 연산**





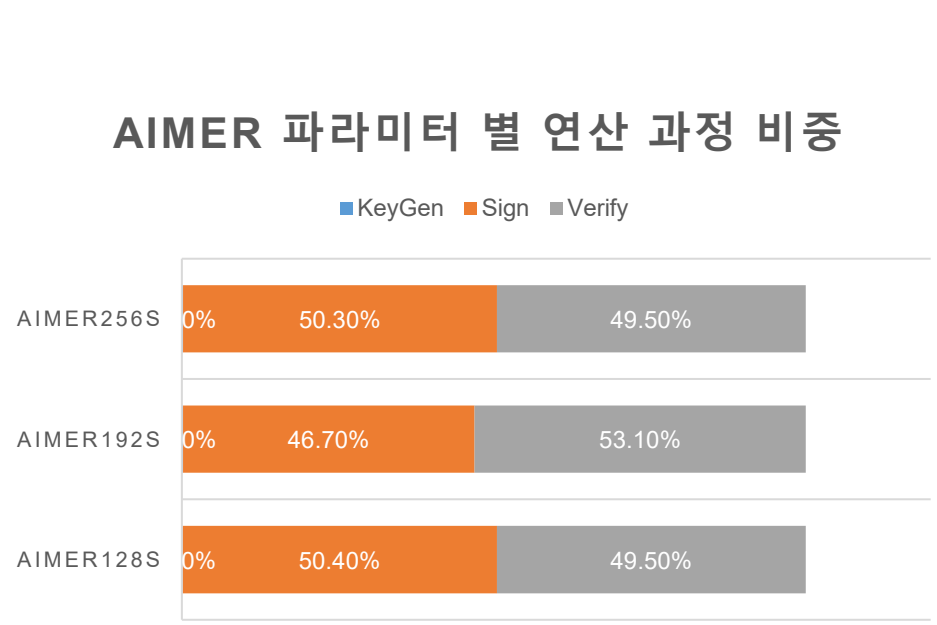


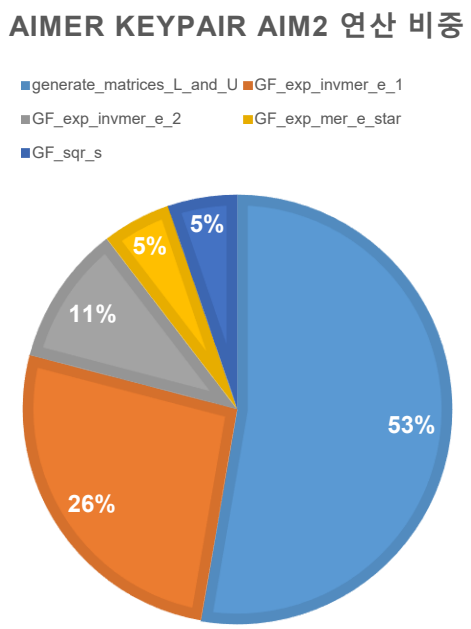


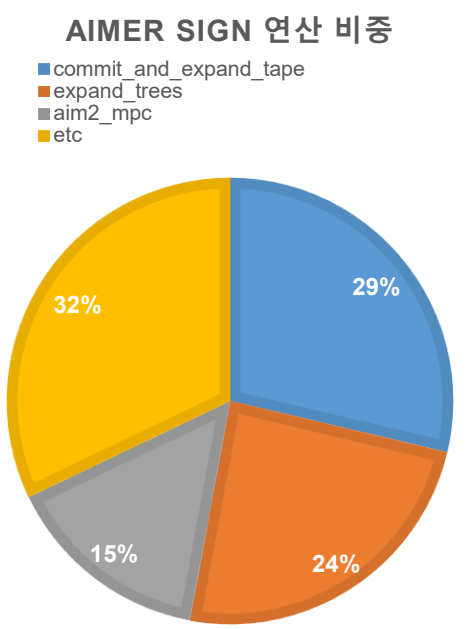
(DSA) AIMER_s 알고리즘 연산 분석 결과

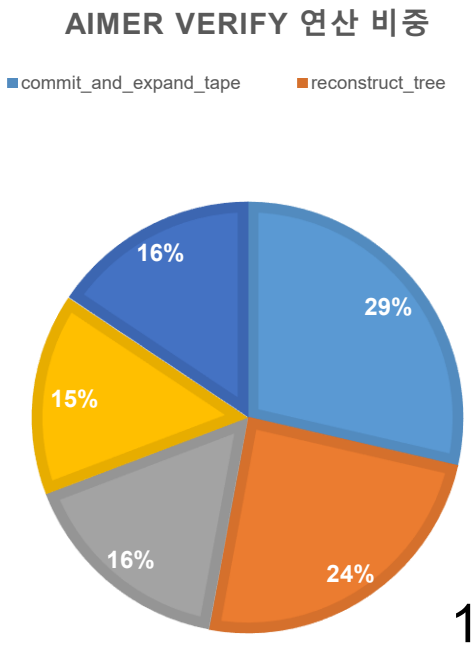
- **AIMER_s에서 서명과 검증 과정 비슷한 연산 비중 차지**
  - AIMER에서는 키 생성, 서명, 검증 과정 구분 없이 **SHAKE 관련 연산이 가장 큰 비중**으로 사용
  - **그 다음으로는 GF 상에서의 곱셈/제곱 연산이 많이 사용**
    - AIMER_f 파라미터 셋에서도 비슷한 분석 결과를 보여줌

- 결과적으로 AIMER 알고리즘에서 가장 비중이 큰 연산은 **SHAKE 관련 연산과 GF 상에서의 곱셈/제곱 연산**



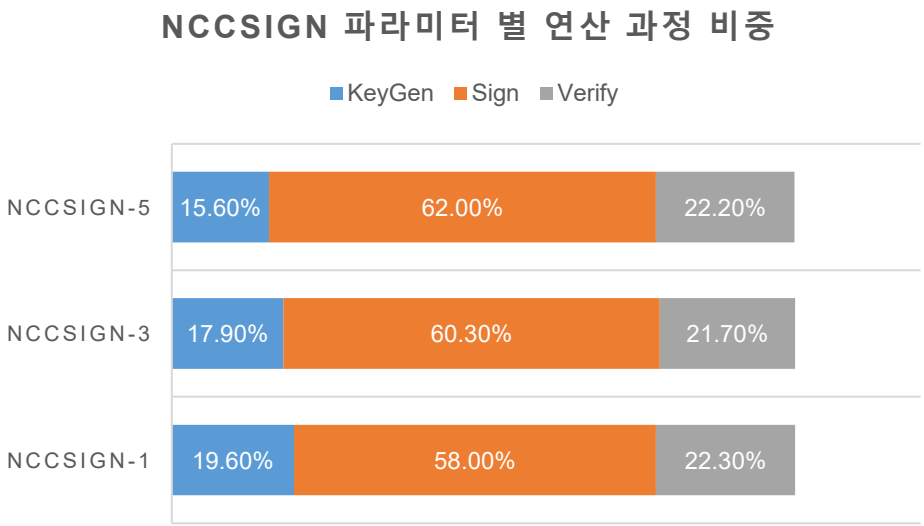


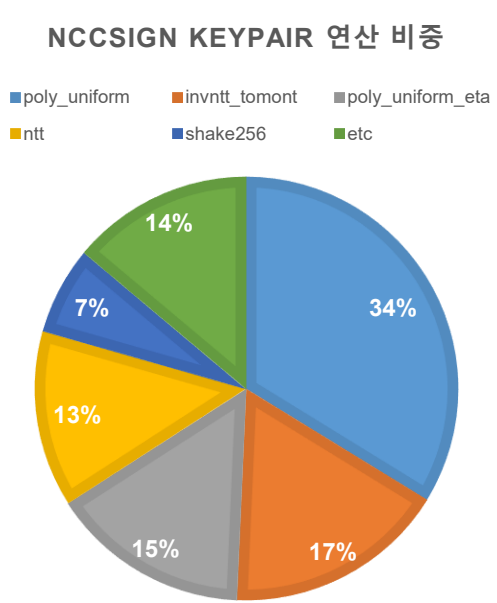


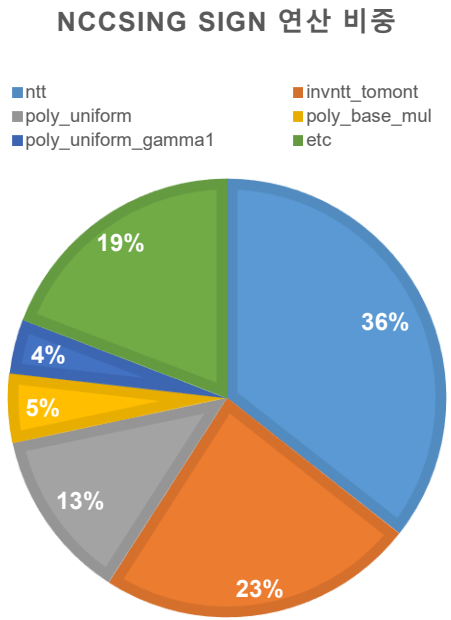


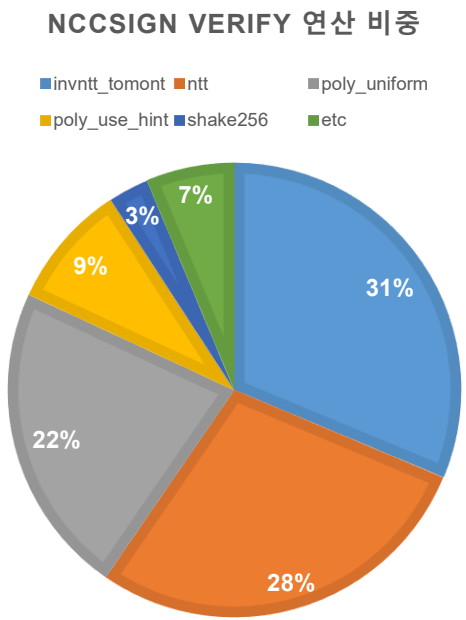
(DSA) NCCSign 알고리즘 연산 분석 결과

- **NCCSign에서 가장 큰 비중을 차지하는 과정은 서명 과정**
  - 서명 과정에서는 다항식 곱셈 관련 **NTT 연산 과정**이 가장 큰 연산 비중 차지
  - 검증 과정에서는 서명 과정과 마찬가지로 다항식 곱셈 관련 **NTT 연산 과정**이 가장 큰 연산 비중 차지
  - 키 생성 과정에서 **SHAKE 관련 연산**과 다항식 곱셈 관련 **NTT 연산 과정**이 가장 큰 연산 비중 차지
  - 전체적으로 NCCSign 알고리즘에서의 주요 연산은 다항식 곱셈 관련 **NTT 연산**과 **SHAKE 관련 연산**



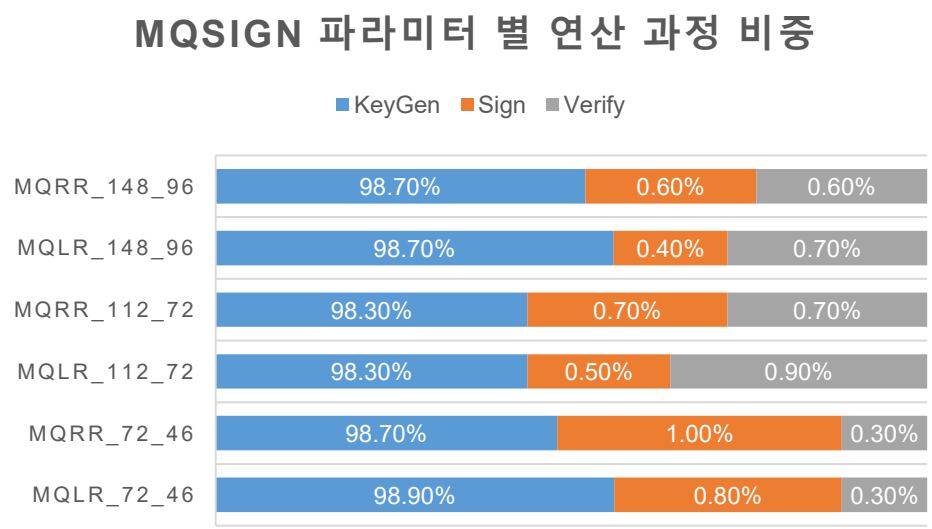


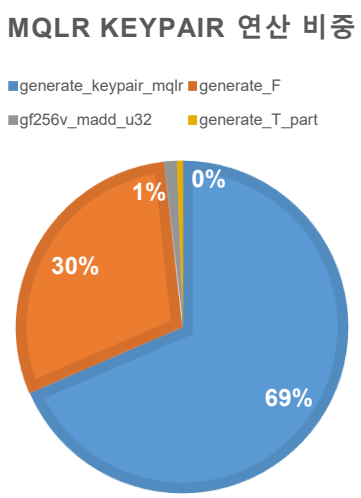


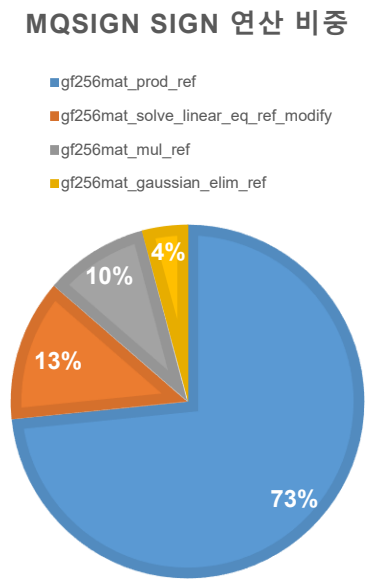


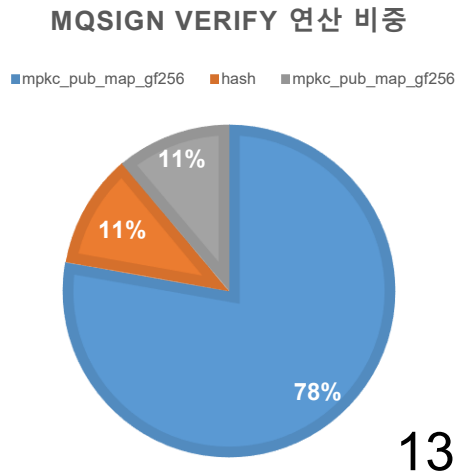
(DSA) MQSign 알고리즘 연산 분석 결과

- **MQSign에서 가장 큰 비중을 차지하는 과정은 키 생성 과정**
  - 키 생성 과정에서는 GF상에서의 **행렬 연산 과정**이 가장 큰 연산 비중을 차지
  - 서명 과정에서는 **벡터 간의 산술 연산**이 가장 큰 연산 비중을 차지
  - 검증 과정에서는 **GF256v_madd 연산**이 가장 큰 연산 비중을 차지
  - MQSign 알고리즘에서 **행렬 연산 과정**과 **벡터 간의 산술 연산**이 가장 큰 비중을 차지
    - 행렬 연산 과정과 벡터 간의 산술 연산 모두에서 **GF256v_madd_u32 함수**가 가장 많이 사용됨









프로파일링 정리 (연산 부하가 높은 순으로 정렬)

- **KEM**
  - 연산과정 분류 : **키생성** > 캡슐화, 디캡슐화
  - 연산자 분류 : **SHA / LSH** > NTT, 산술연산

- **DSA**
  - 연산과정 분류 : **서명** > 키생성, 검증
  - 연산자 분류 : **SHA** > NTT / FFT, 산술연산

이기종의 양자내성암호가 선정될 경우 **SHA / LSH에 대한 가속기**의 수요가 높아질 것으로 예상
상호 호환 용이성을 위해 LSH를 SHA로 대체하는 것도 하나의 옵션
그 다음으로는 **격자 중심의 경우에는 NTT 가속기**

최신 업데이트 기준 알고리즘 벤치마크

성능 측정 환경

Testing Environment1 (clean, avx2)

- OS: Ubuntu 23.10.1
- CPU: Ryzen 7 4800H (2.90 GHz)
- RAM: 16GB
- Compiler: gcc 13.2.0
- Optimization Level: -O3

Testing Environment3 (clean)

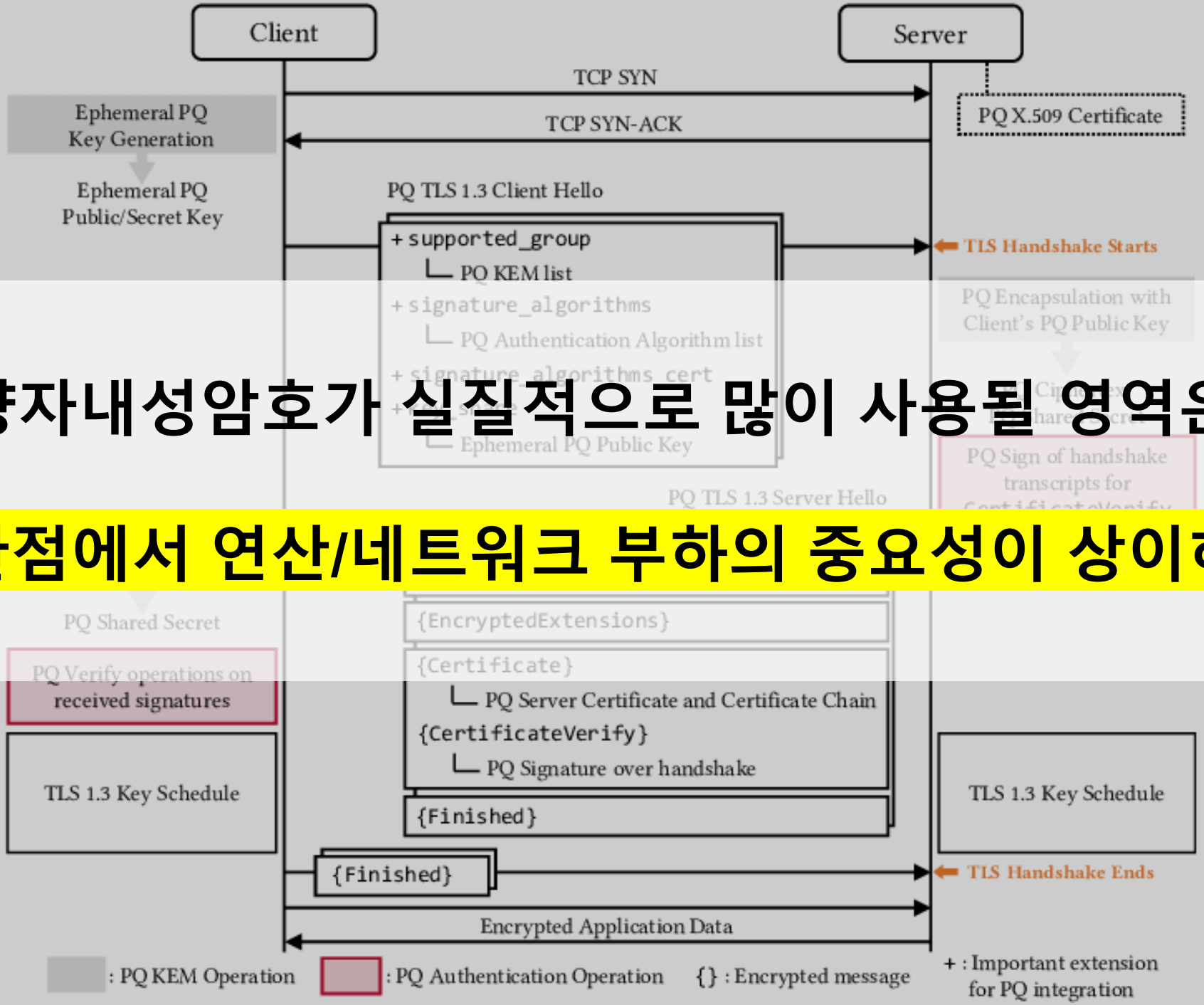
- OS: macOs Sonoma 14.4.1
- CPU: Apple M2 (3.23 GHz)
- RAM: 8GB
- Compiler: Apple clang 15.0.0
- Optimization Level: -O3

Testing Environment2(clean, avx2)

- OS: Ubuntu 23.10.1
- CPU: Intel i5-8259U (2.30 GHz)
- RAM: 16GB
- Compiler: gcc 13.2.0
- Optimization Level: -O3

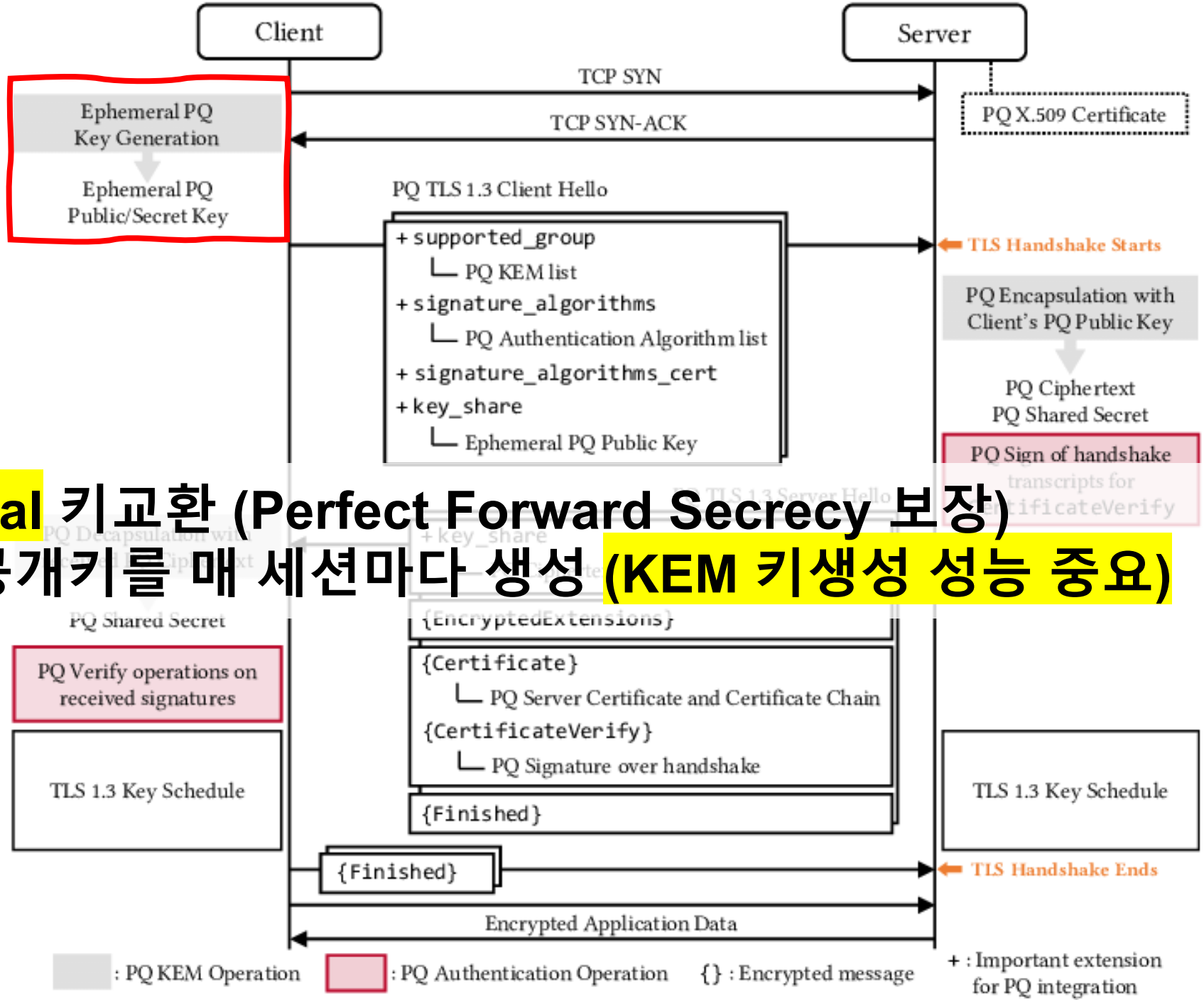
명령어셋에서 오는 차이점 발생

다만 **일반적인 명령어 셋은 공유하는 경향이 있음**



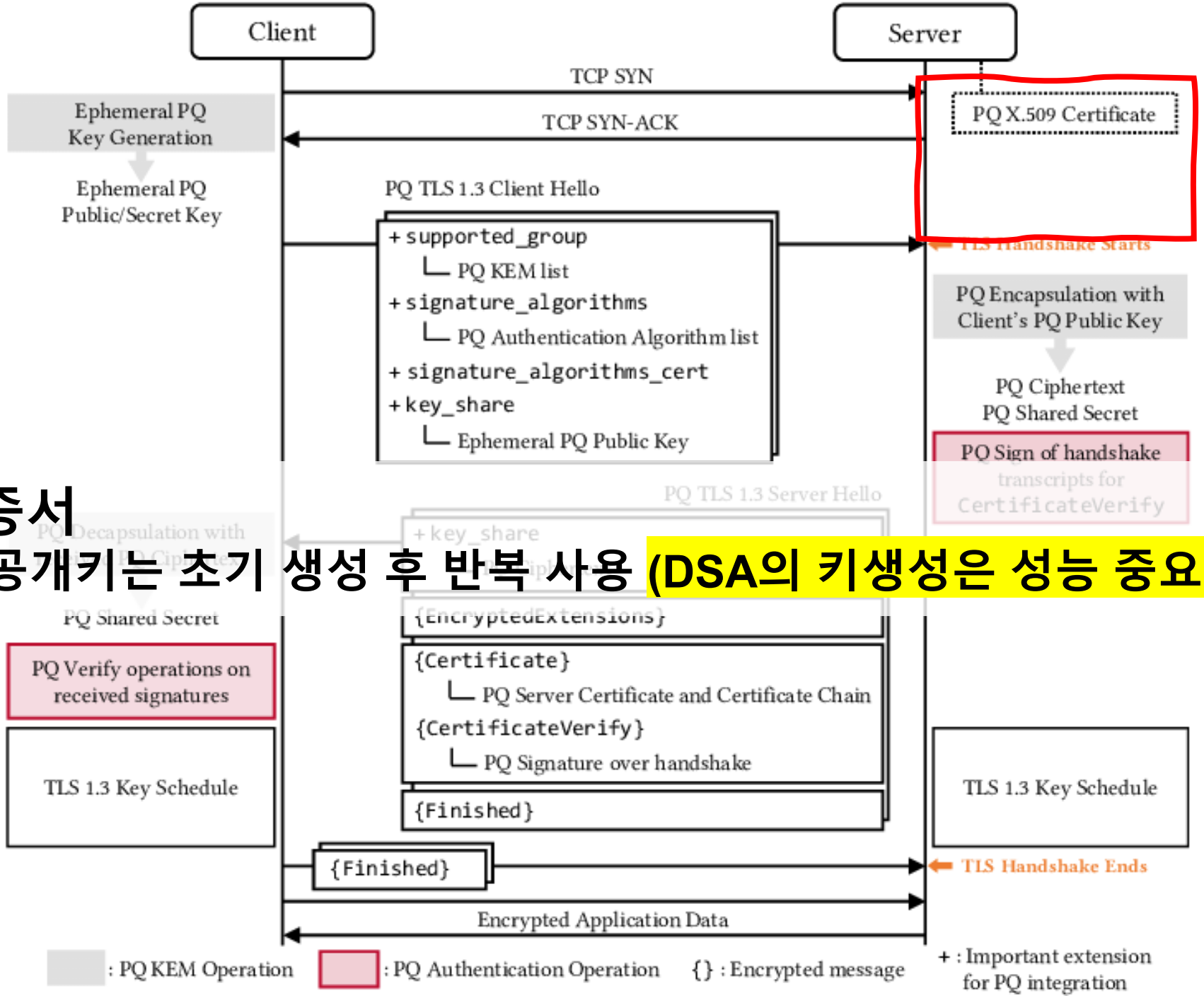
양자내성암호가 실질적으로 많이 사용될 영역은 TLS

TLS 관점에서 연산/네트워크 부하의 중요성이 상이하게 나타남



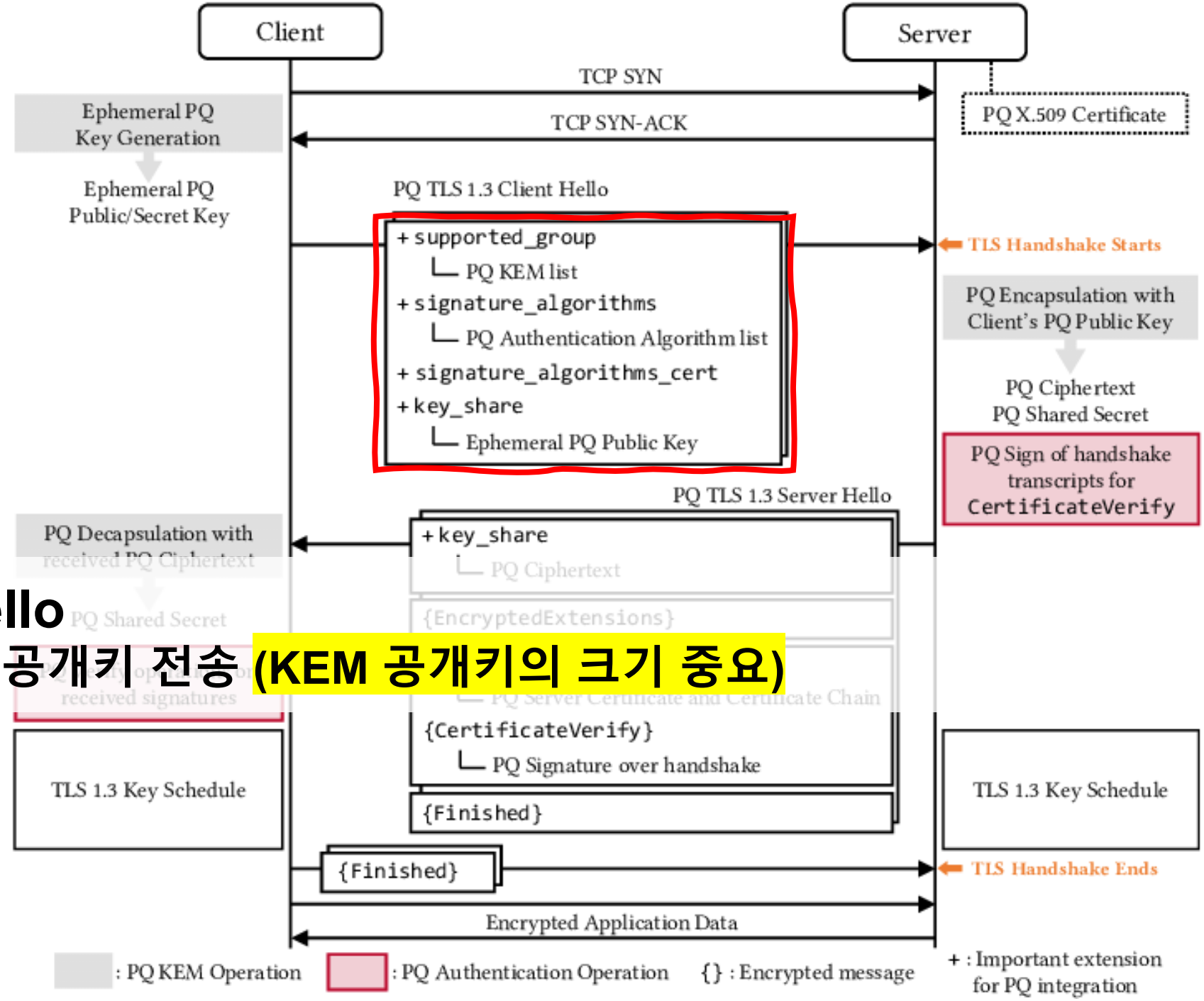
Ephemeral 키교환 (Perfect Forward Secrecy 보장)

- 임시 공개키를 매 세션마다 생성 **(KEM 키생성 성능 중요)**



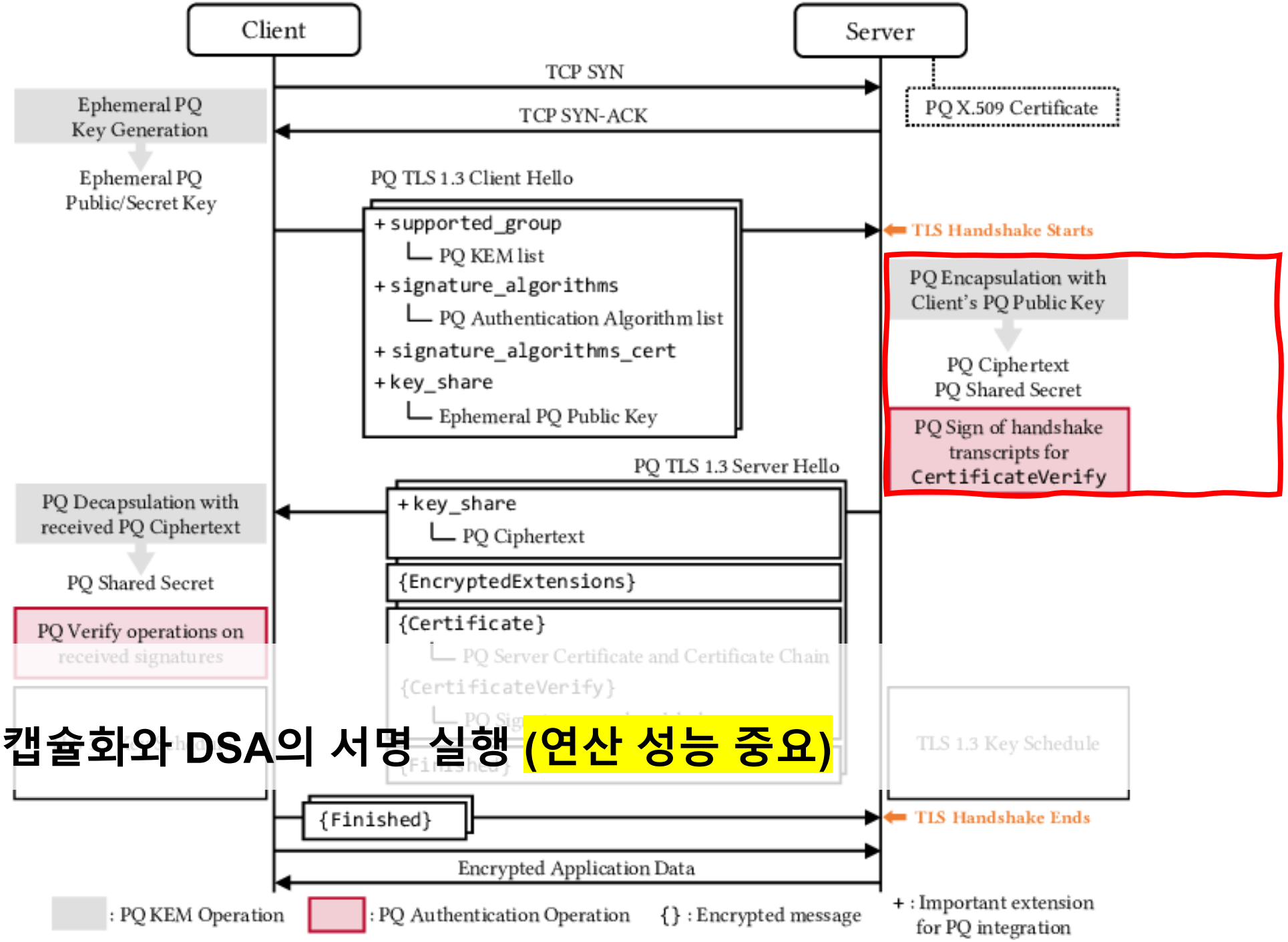
X.509 인증서

- 서명용 공개키는 초기 생성 후 반복 사용 **(DSA의 키생성은 성능 중요도 낮음)**



Client Hello

- KEM의 공개키 전송 **(KEM 공개키의 크기 중요)**

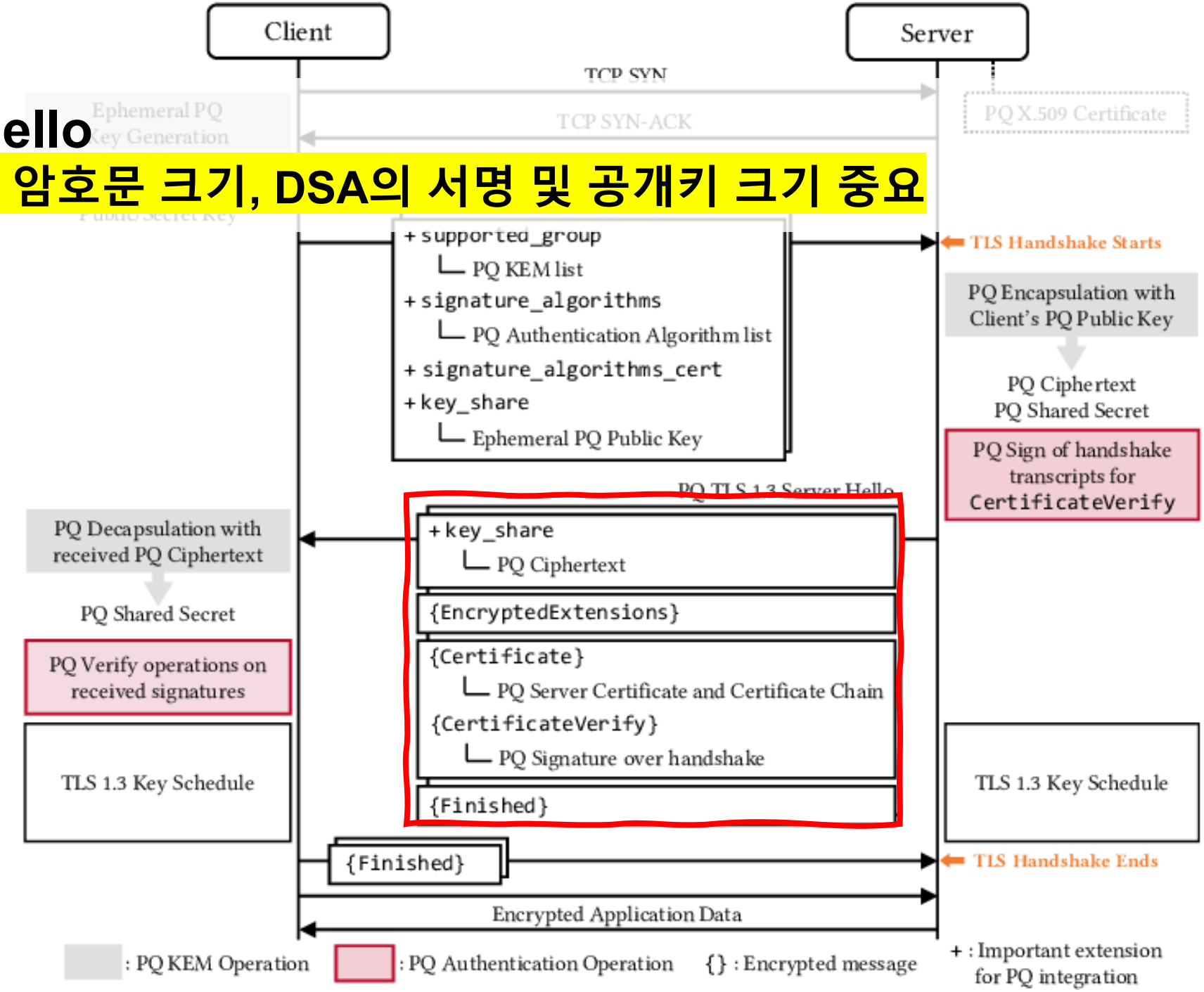


서버연산

- KEM의 캡슐화와 DSA의 서명 실행 (연산 성능 중요)

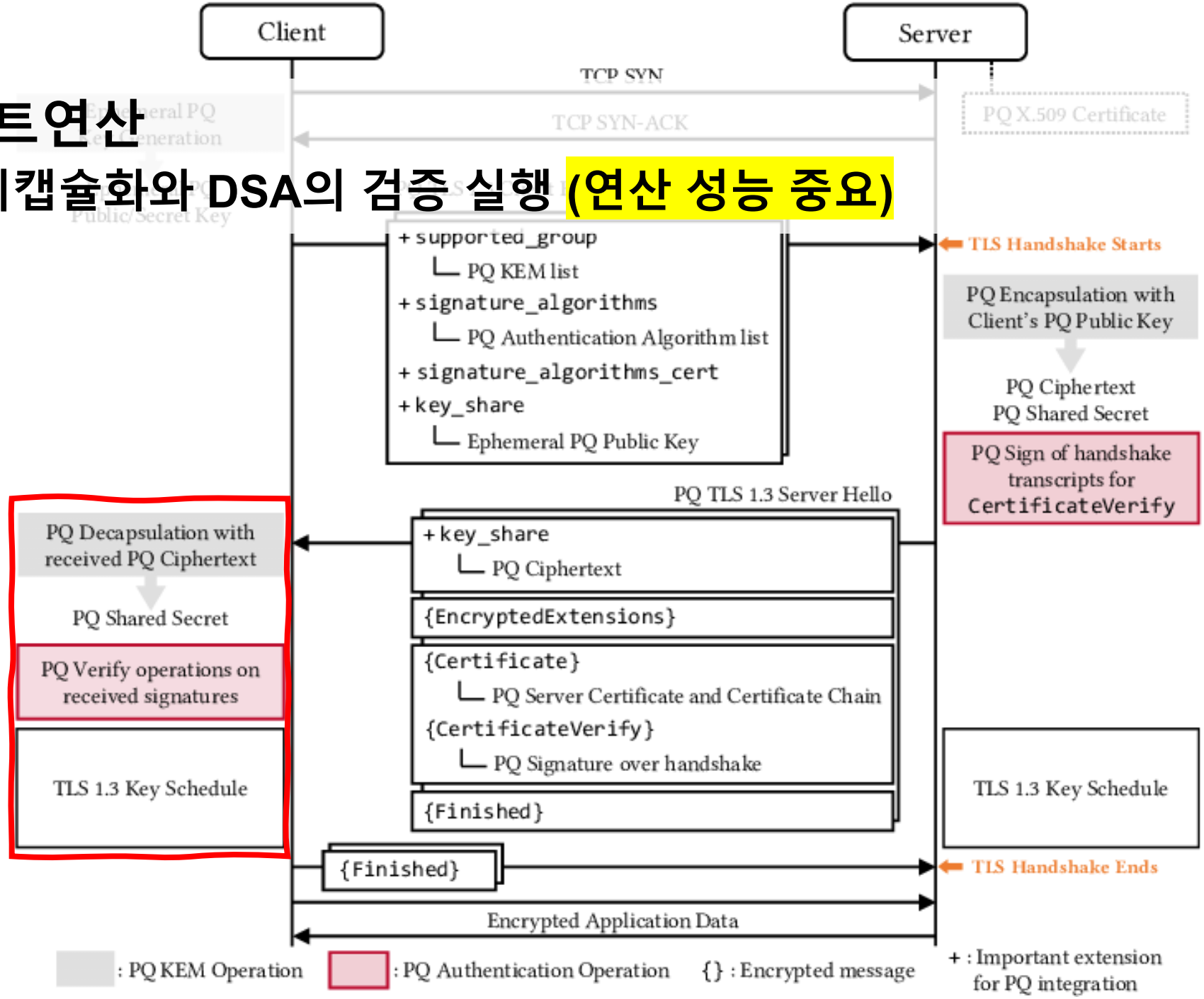
Server Hello

- **KEM의 암호문 크기, DSA의 서명 및 공개키 크기 중요**



클라이언트연산

- KEM의 디캡슐화와 DSA의 검증 실행 (연산 성능 중요)



TLS 관점에서 PQC 최적화 중요도

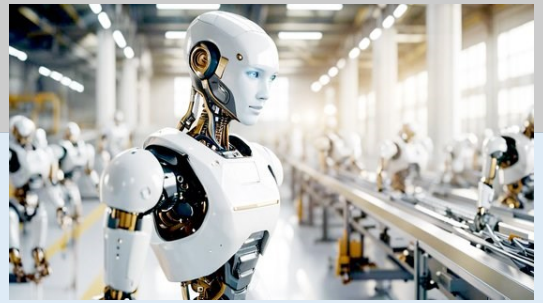
클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	**(오프라인) DSA 키생성** KEM 캡슐화 DSA 서명

일반적으로 클라이언트가 성능이 낮은 만큼 클라이언트 쪽의 PQC연산이 효율적일수록 TLS 적용에 유리

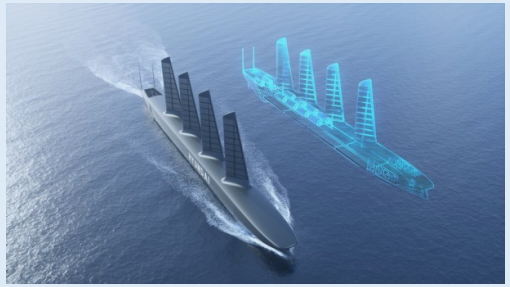
- **네트워크 패킷 교환 관점 (작을수록 좋음)**
  - KEM 공개키
  - KEM 암호문
  - DSA 서명
  - DSA 공개키

클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	(오프라인) DSA 키생성 KEM 캡슐화 DSA 서명

연산속도 중요 서비스 **(컴퓨팅 파워 제약 사항)**



- **네트워크 패킷 교환 관점 (작을수록 좋음)**
  - KEM 공개키
  - KEM 암호문
  - DSA 서명
  - DSA 공개키



전송속도 중요 서비스 **(통신회선 제약 사항)**

new (PKE/KEM) 성능 비교 (clean)

Environment 1: Ryzen

키생성

scheme	avg
SMAUG-Timer	79,115
SMAUG-T1	84,395
SMAUG-T3	160,405
NTRU+KEM576	178,774
NTRU+PKE576	180,894
NTRU+PKE768	210,248
NTRU+KEM768	210,261
NTRU+KEM864	230,556
NTRU+PKE864	230,793
SMAUG-T5	232,659
NTRU+PKE1152	344,151
NTRU+KEM1152	349,307
PALOMA128	55,486,704
PALOMA192	248,335,754
PALOMA256	283,806,633

최고와 최저의 차이가 많이 발생

캡슐화

scheme	avg
SMAUG-T1	61,518
SMAUG-Timer	61,666
NTRU+PKE576	77,588
NTRU+KEM576	82,246
NTRU+KEM768	115,843
NTRU+PKE768	115,866
NTRU+KEM864	121,771
NTRU+PKE864	123,215
PALOMA128	125,725
SMAUG-T3	126,248
NTRU+PKE1152	153,492
NTRU+KEM1152	157,981
PALOMA192	175,054
SMAUG-T5	206,697
PALOMA256	207,753

최고와 최저의 차이가 크지 않음

디캡슐화

scheme	avg
NTRU+KEM576	73,735
NTRU+PKE576	75,289
SMAUG-Timer	81,694
SMAUG-T1	82,241
NTRU+KEM768	114,679
NTRU+PKE768	114,768
NTRU+KEM864	124,974
NTRU+PKE864	125,821
SMAUG-T3	157,720
NTRU+PKE1152	159,135
NTRU+KEM1152	161,589
SMAUG-T5	252,221
PALOMA128	2,447,435
PALOMA256	9,509,082
PALOMA192	9,549,977

최고와 최저의 차이가 많이 발생

TLS 관점에서 PQC 최적화 중요도 (Clean 관점)

클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	(오프라인) DSA 키생성 KEM 캡슐화 DSA 서명

알고리즘	키생성	디캡슐화	총합
SMAUG-Timer	79,115	81,694	160,809
NTRU+KEM576	178,774	73,735	252,509
PALOMA128	55,486,704	2,447,435	57,934,139

알고리즘	캡슐화	총합
SMAUG-Timer	61,666	61,666
NTRU+KEM576	77,588	77,588
PALOMA128	125,725	125,725

Environment 1: Ryzen

격자기반 대칭키기반 다변수기반

new (DSA) 성능 비교 (clean)

키생성

scheme	avg
AIMer128s	108,704
AIMer128f	159,134
AIMer192s	187,591
NCC-Sign3_shake_NTT_MODE_1	194,578
NCC-Sign5_shake_NTT_MODE_1	201,215
NCC-Sign1_shake_NTT_MODE_1	201,325
AIMer192f	201,497
NCC-Sign3_shake_NTT_MODE_3	245,099
NCC-Sign5_shake_NTT_MODE_3	248,687
NCC-Sign1_shake_NTT_MODE_3	273,995
NCC-Sign1-AES_NTT_MODE_1	323,828
NCC-Sign5_shake_NTT_MODE_5	375,236
NCC-Sign3_shake_NTT_MODE_5	376,189
NCC-Sign1_shake_NTT_MODE_5	380,698
AIMer256f	413,606
NCC-Sign3-AES_NTT_MODE_1	423,110
NCC-Sign1-AES_NTT_MODE_3	423,451
NCC-Sign3-AES_NTT_MODE_3	423,841
NCC-Sign5-AES_NTT_MODE_3	428,798
AIMer256s	454,476
NCC-Sign5-AES_NTT_MODE_5	616,114
NCC-Sign3-AES_NTT_MODE_5	616,295
NCC-Sign5-AES_NTT_MODE_1	617,154
NCC-Sign1-AES_NTT_MODE_5	620,159
HAETAE2	1,060,674
HAETAE5	1,379,694
HAETAE3	1,796,195
MQLR_256_72_46	75,382,639
MQRR_o_72_46	97,426,775
MQRR_s_72_46	100,871,813
MQRR_m_72_46	100,898,767
MQLR_256_112_72	292,262,140
MQRR_s_112_72	374,184,501
MQRR_o_112_72	374,358,898
MQRR_m_148_96	970,586,332

최고와 최저의 차이가 많이 발생

최저를 제외하고는 차이가 크지 않음

서명

scheme	avg
NCC-Sign3_shake_NTT_MODE_1	422,889
NCC-Sign5_shake_NTT_MODE_1	425,517
MQLR_256_72_46	483,044
NCC-Sign5_shake_NTT_MODE_3	505,538
NCC-Sign3-AES_NTT_MODE_1	742,379
NCC-Sign5_shake_NTT_MODE_5	811,556
MQRR_o_72_46	820,027
MQRR_s_72_46	820,539
MQRR_m_72_46	823,567
NCC-Sign3_shake_NTT_MODE_3	947,492
NCC-Sign3_shake_NTT_MODE_5	1,150,546
NCC-Sign1-AES_NTT_MODE_1	1,216,509
NCC-Sign1_shake_NTT_MODE_1	1,238,036
MQLR_256_112_72	1,307,986
NCC-Sign1_shake_NTT_MODE_3	1,346,835
NCC-Sign3-AES_NTT_MODE_3	1,384,464
NCC-Sign5-AES_NTT_MODE_1	1,528,635
NCC-Sign1_shake_NTT_MODE_5	1,593,985
NCC-Sign3-AES_NTT_MODE_5	1,606,453
NCC-Sign1-AES_NTT_MODE_5	1,624,432
NCC-Sign1-AES_NTT_MODE_3	1,916,567
MQRR_m_112_72	2,035,399
MQRR_s_112_72	2,036,364
MQRR_o_112_72	2,058,603
NCC-Sign5-AES_NTT_MODE_3	2,178,587
NCC-Sign5-AES_NTT_MODE_5	2,371,471
HAETAE3	2,740,577
MQLR_256_148_96	2,780,855
AIMer128f	3,742,915
HAETAE2	3,911,718
MQRR_m_148_96	4,228,334
MQRR_o_148_96	4,234,679
MQRR_s_148_96	4,255,687
HAETAE5	5,300,714
[illegible]	[illegible]
[illegible]	41,270
[illegible]	94,493
AIMer256s	119,340,970

최고와 최저의 차이가 많이 발생

HAETAE와 AIMER가 느림

검증

scheme	avg
HAETAE2	155,656
NCC-Sign5_shake_NTT_MODE_1	216,628
NCC-Sign3_shake_NTT_MODE_1	217,872
NCC-Sign1_shake_NTT_MODE_1	221,245
NCC-Sign1_shake_NTT_MODE_3	252,170
NCC-Sign5_shake_NTT_MODE_3	262,487
NCC-Sign3_shake_NTT_MODE_3	262,991
HAETAE3	286,273
NCC-Sign1-AES_NTT_MODE_1	324,448
HAETAE5	337,215
NCC-Sign1_shake_NTT_MODE_5	409,797
NCC-Sign5_shake_NTT_MODE_5	410,587
NCC-Sign3_shake_NTT_MODE_5	415,953
NCC-Sign3-AES_NTT_MODE_1	424,907
NCC-Sign5-AES_NTT_MODE_3	426,546
NCC-Sign3-AES_NTT_MODE_3	429,083
NCC-Sign1-AES_NTT_MODE_3	435,950
NCC-Sign3-AES_NTT_MODE_5	623,024
NCC-Sign5-AES_NTT_MODE_1	634,439
NCC-Sign5-AES_NTT_MODE_5	640,643
NCC-Sign1-AES_NTT_MODE_5	645,942
MQLR_256_72_46	690,874
MQRR_s_72_46	691,622
MQRR_m_72_46	697,515
MQRR_o_72_46	698,096
MQRR_s_112_72	2,019,703
MQLR_256_112_72	2,023,707
MQRR_m_112_72	2,027,657
MQRR_o_112_72	2,027,969
AIMer128f	3,545,576
MQLR_256_148_96	4,157,115
MQRR_o_148_96	4,158,176
MQRR_m_148_96	4,161,130
MQRR_s_148_96	4,169,491
AIMer192f	7,800,523
AIMer192s	62,821,167
AIMer256s	119,925,648

최고와 최저의 차이가 많이 발생

TLS 관점에서 PQC 최적화 중요도 (Clean 관점)

클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	(오프라인) DSA 키생성 KEM 캡슐화 DSA 서명

알고리즘	키생성	디캡슐화	총합
SMAUG-Timer	79,115	81,694	160,809
NTRU+KEM576	178,774	73,735	252,509
PALOMA128	55,486,704	2,447,435	57,934,139

알고리즘	검증	총합
HAETAE2	155,656	155,656
NCC-Sign1	221,245	221,245
MQLR	690,874	690,874
AIMer128f	3,545,576	3,545,576

알고리즘	캡슐화	총합
SMAUG-Timer	61,666	61,666
NTRU+KEM576	77,588	77,588
PALOMA128	125,725	125,725

알고리즘	서명	총합
MQLR	483,044	483,044
NCC-Sign1	1,238,036	1,238,036
AIMer128f	3,742,915	3,742,915
HAETAE2	3,911,718	3,911,718

new 성능 비교 (clean)

Environment 1: Ryzen
Environment 2: Intel
Environment 3: Apple Silicon

<PEK/KEM>

- **Keygen**
  - Environment 1, 2: **SMAUG-Timer** (1st)
  - Environment 3: **SMAUG-T1** (1st)
- **Encapsulation**
  - Environment 1,2,3: **SMAUG–T1** (1st)
- **Decapsulation**
  - Environment 1,2 : **NTRU+KEM576** (1st)
  - Environment 3: **NTRU+PKE576**(1st)

SMAUG와 NTRU의 성능 우수

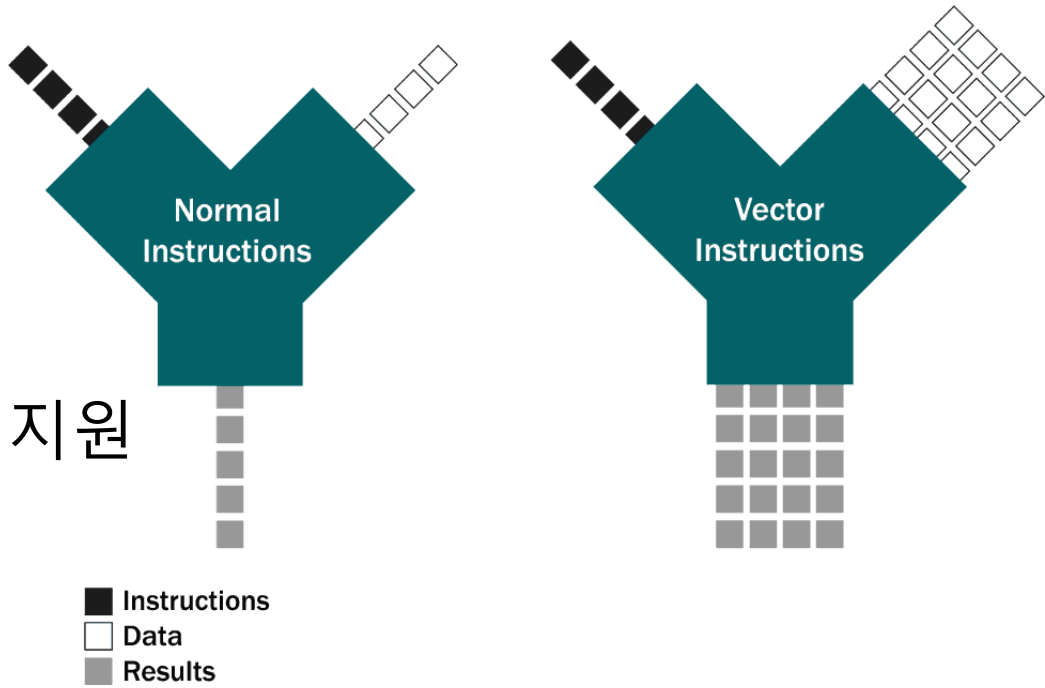
<Digital signature>

- **Keygen**
  - Environment 1,2,3: **AIMer128s** (1st)
- **Signature**
  - Environment 1: **NCC_Sign3_shake_NTT_MODE_1**(1st)
  - Environment 2: **NCC-Sign1-Shake_NTT_MODE_1**(1st)
  - Environment 3: **NCC-Sign5-shake_NTT_MODE_1**(1st)
- **Verify**
  - Environment 1, 2, 3: **HAETAE2** (1st)

가장 성능이 우수한 알고리즘은 없음

Clean과 AVX2의 차이점

- **Clean은 pure C를 의미; AVX2는 Intel의 SIMD 명령어 셋을 의미**
  - Clean은 Client 그리고 AVX2는 Server 에 가까움
- **Clean에 유리한 구현? (AVX2 대비)**
  - 연산에 패턴이 없이 복잡한 경우
- **AVX2에 유리한 구현?**
  - 입력 인자의 크기가 32-비트 이하
  - **동일한 연산 패턴이 반복되는 경우**
  - 마이크로 컨트롤러를 제외하고는 유사 SIMD 지원



new (PKE/KEM) 성능 비교 (avx2)

Environment 1: Ryzen

키생성

scheme	avg
SMAUG-T1 kem 90s	22,611
SMAUG-T3 kem 90s	31,190
NTRU+PKE576	34,697
NTRU+KEM576	34,909
SMAUG-T1 kem	38,745
NTRU+KEM864	39,844
NTRU+PKE864	40,851
SMAUG-T5 kem 90s	41,279
SMAUG-T3 kem	53,750
NTRU+KEM768	53,887
NTRU+PKE768	56,783
SMAUG-T5 kem	67,161
NTRU+KEM1152	71,971
NTRU+PKE1152	73,764

3.49x

5.15x

최고와 최저의 차이가 크지 않음

캡슐화

scheme	avg
SMAUG-T1 kem 90s	15,754
SMAUG-T1 kem	19,517
NTRU+KEM576	20,572
NTRU+PKE576	20,601
SMAUG-T3 kem 90s	25,610
NTRU+PKE768	27,032
NTRU+KEM768	27,070
NTRU+PKE864	29,828
NTRU+KEM864	29,864
SMAUG-T5 kem 90s	35,141
SMAUG-T3 kem	38,019
NTRU+PKE1152	38,536
NTRU+KEM1152	38,637
SMAUG-T5 kem	52,113

3.90x

3.77x

최고와 최저의 차이가 크지 않음

디캡슐화

scheme	avg
NTRU+KEM576	12,783
NTRU+PKE576	13,358
NTRU+PKE768	17,088
NTRU+KEM768	17,119
NTRU+KEM864	19,272
NTRU+PKE864	19,369
NTRU+PKE1152	24,687
NTRU+KEM1152	24,748
SMAUG-T1 kem 90s	25,363
SMAUG-T1 kem	29,091
SMAUG-T3 kem 90s	39,697
SMAUG-T3 kem	51,584
SMAUG-T5 kem 90s	52,377
SMAUG-T5 kem	70,154

5.76x

3.22x

최고와 최저의 차이가 크지 않음

격자기반

TLS 관점에서 PQC 최적화 중요도 (AVX2 관점)

클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	(오프라인) DSA 키생성 KEM 캡슐화 DSA 서명

알고리즘	키생성	디캡슐화	총합
NTRU+KEM576	34,697	12,783	47,480
SMAUG-T1	22,611	25,363	47,974

알고리즘	캡슐화	총합
SMAUG-T1	15,754	15,754
NTRU+KEM576	20,572	20,572

new (DSA) 성능 비교 (avx2)

Environment 1: Ryzen

키생성

scheme	avg
AIMer128f	40,172
NCC-Sign3-AES_NTT_MODE_1	46,139
NCC-Sign5-AES_NTT_MODE_1	51,753
NCC-Sign1-AES_NTT_MODE_1	52,061
NCC-Sign5-AES_NTT_MODE_3	55,441
NCC-Sign1-AES_NTT_MODE_3	61,174
NCC-Sign3-AES_NTT_MODE_3	61,816
NCC-Sign5-AES_NTT_MODE_5	83,253
NCC-Sign3-AES_NTT_MODE_5	85,755
NCC-Sign3_shake_NTT_MODE_1	86,524
NCC-Sign5_shake_NTT_MODE_1	86,864
NCC-Sign1_shake_NTT_MODE_1	86,909
NCC-Sign1-AES_NTT_MODE_5	91,723
AIMer128s	93,,037
AIMer192s	97,972
AIMer192f	99,173
NCC-Sign5_shake_NTT_MODE_3	107,482
NCC-Sign1_shake_NTT_MODE_3	107,588
NCC-Sign3_shake_NTT_MODE_3	107,782
NCC-Sign5_shake_NTT_MODE_5	153,359
NCC-Sign1_shake_NTT_MODE_5	161,262
NCC-Sign3_shake_NTT_MODE_5	162,757
AIMer256f	236,956
AIMer256s	242,895
HAETAE2	453,534
HAETAE3	657,020
HAETAE5	703,772
MQRR_o_72_46	2,928,163
MQLR_256_72_46	3,679,090
MQRR_m_72_46	4,061,374
MQRR_s_72_46	4,868,979
MQRR_s_112_72	13,086,048
MQRR_o_112_72	13,107,964
MQLR_256_112_72	16,766,445
MQRR_m_148_96	55,220,680

2.70x

4.21x

2.33x

25.744x

최고와 최저의 차이가 많이 발생

최저를 제외하고는 차이가 크지 않음

서명

scheme	avg
MQLR_256_72_46	46,695
MQRR_o_72_46	63,076
MQRR_s_72_46	63,269
MQRR_m_72_46	63,741
NCC-Sign5-AES_NTT_MODE_1	74,930
NCC-Sign5-AES_NTT_MODE_3	100,381
NCC-Sign1_shake_NTT_MODE_1	121,967
NCC-Sign5_shake_NTT_MODE_1	123,040
MQLR_256_112_72	124,081
NCC-Sign3-AES_NTT_MODE_3	151,906
NCC-Sign5-AES_NTT_MODE_5	156,623
NCC-Sign1-AES_NTT_MODE_1	159,687
NCC-Sign1_shake_NTT_MODE_3	161,059
NCC-Sign5_shake_NTT_MODE_3	163,247
MQRR_m_112_72	170,728
MQRR_o_112_72	170,938
MQRR_s_112_72	173,875
NCC-Sign3_shake_NTT_MODE_1	177,436
MQLR_256_148_96	207,041
NCC-Sign1-AES_NTT_MODE_3	211,760
NCC-Sign3_shake_NTT_MODE_3	229,961
NCC-Sign1_shake_NTT_MODE_5	245,694
NCC-Sign5_shake_NTT_MODE_5	245,829
NCC-Sign3_shake_NTT_MODE_5	246,718
HAETAE2	270,121
NCC-Sign3-AES_NTT_MODE_1	285,581
MQRR_s_148_96	294,936
MQRR_o_148_96	298,475
MQRR_m_148_96	303,145
NCC-Sign1-AES_NTT_MODE_5	325,009
HAETAE3	377,726
NCC-Sign3-AES_NTT_MODE_5	480,938
AIMer128f	811,275
HAETAE5	1,091,046
AIMer192f	2,210,305
AIMer256f	4,071,768
AIMer256s	29,154,407

9.05x

6.44x

13.85x

4.82x

최고와 최저의 차이가 많이 발생

검증

scheme	avg
MQRR_o_72_46	34,569
MQLR_256_72_46	34,585
MQRR_m_72_46	34,799
MQRR_s_72_46	35,024
NCC-Sign1-AES_NTT_MODE_1	58,010
NCC-Sign5-AES_NTT_MODE_1	58,208
NCC-Sign3-AES_NTT_MODE_1	58,267
HAETAE2	69,956
NCC-Sign5-AES_NTT_MODE_3	70,806
NCC-Sign1-AES_NTT_MODE_3	70,894
NCC-Sign3-AES_NTT_MODE_3	71,563
NCC-Sign1_shake_NTT_MODE_1	88,848
NCC-Sign5_shake_NTT_MODE_1	89,230
NCC-Sign3_shake_NTT_MODE_1	90,329
HAETAE3	109,668
NCC-Sign1-AES_NTT_MODE_5	109,805
NCC-Sign3-AES_NTT_MODE_5	109,898
NCC-Sign5-AES_NTT_MODE_5	110,339
MQRR_m_112_72	110,676
MQRR_s_112_72	110,792
MQRR_o_112_72	110,987
NCC-Sign3_shake_NTT_MODE_3	111,231
NCC-Sign1_shake_NTT_MODE_3	112,321
NCC-Sign5_shake_NTT_MODE_3	113,883
MQLR_256_112_72	114,097
HAETAE5	133,173
NCC-Sign1_shake_NTT_MODE_5	170,910
NCC-Sign3_shake_NTT_MODE_5	171,433
NCC-Sign5_shake_NTT_MODE_5	171,678
MQRR_s_148_96	195,273
MQLR_256_148_96	201,476
MQRR_m_148_96	206,062
MQRR_o_148_96	209,508
AIMer128f	783,010
AIMer192f	2,131,677

20.19x

5.59x

2.22x

4.52x

최고와 최저의 차이가 많이 발생

최저를 제외하고는 차이가 크지 않음

AVX2에서 성능이 급격히 상승한 경우?

- **데이터 인자의 크기가 작을 수록 데스크 톱에서의 성능 향상 극대화**
  - 예시) 8-비트 연산을 Intel 데스크톱에서 수행하면?
    → 56-비트가 활용되지 못함

  - 예시) 8-비트 연산을 Intel AVX2로 수행하면?
    → 32개의 8-비트 연산 수행

단순 계산 시 8-비트 연산을 SIMD로 변경하면 256배 (8 X 32) 성능 향상 가능 (이는 이론적이긴 함)

TLS 관점에서 PQC 최적화 중요도 (AVX2 관점)

클라이언트	서버
KEM 키생성 KEM 디캡슐화 DSA 검증	(오프라인) DSA 키생성 KEM 캡슐화 DSA 서명

알고리즘	키생성	디캡슐화	총합
NTRU+KEM576	34,697	12,783	47,480
SMAUG-T1	22,611	25,363	47,974

알고리즘	캡슐화	총합
SMAUG-T1	15,754	15,754
NTRU+KEM576	20,572	20,572

알고리즘	검증	총합
MQLR	34,569	34,569
NCC-Sign1	58,010	58,010
HAETAE2	69,956	69,956
AIMer128f	783,010	783,010

알고리즘	서명	총합
MQLR	46,695	46,695
NCC-Sign1	121,967	121,967
HAETAE2	270,121	270,121
AIMer128f	811,275	811,275

new 성능 비교 (avx2)

Environment 1: Ryzen
Environment 2: Intel

<PEK/KEM>

- **Keygen**
  - Environment 1,2: **SMAUG-T1 kem 90s** (1st)
- **Encapsulation**
  - Environment 1,2: **SMAUG-T1 kem 90s** (1st)
- **Decapsulation**
  - Environment 1 : **NTRU+KEM576** (1st)
  - Environment 2 : **NTRU+PKE576** (1st)

여전히 SMAUG와 NTRU의 성능 우수

<Digital signature>

- **Keygen**
  - Environment 1, 2: **AIMer128f** (1st)
- **Signature**
  - Environment 1, 2: **MQLR_256_72_46** (1st)
- **Verification**
  - Environment 1, 2: **MQRR_o_72_46** (1st)

MQLR이 AVX2 구현에서는 성능 우수
앞에서 살펴본 AVX2 최적화 구현 적합도가 높음

노란색 음영 처리된 부분은 Clean 버전과 상이한 결과

순위가 바뀐 경우 병렬화가 더 용이하다고 판단 가능

이전 결과와 업데이트된 결과 비교(clean)

대체로 성능이 향상되었으며

상황에 따라 성능 저하도 관찰됨

이전 결과와 업데이트된 결과 비교(avx2)

AVX2 구현 상에서 대체로 성능 향상

SMAUG-T 업데이트 관련

Hyoeun Seong 2024. 10. 4. 오후 8:39:13

받는사람 KpqC-bulletin

Dear all,
SMAUG-T version 4.0 is now available on GitHub.
Thanks to several analysis reports(vulnerabilities found from TIMECOP integration, improved Meet-LWE, and reports on code errors), we have made improvements to the design and parameters.

The key updates are as follows:

- Parameter updates: The parameter has been updated to increase the Hamming weight for the secret key. This update enhances secret randomness and reduces DFP while still meeting the targeted security.
- New sparse CBD sampler: A newly designed secure and efficient sparse CBD is now used for ephemeral randomness generation in encap/decap. To improve side-channel resistance and efficiency, the fixed-weight sampler has been avoided in encap/decap.
- HWT sampler: The fixed-weight sampler in key generation has been updated with a ternary version of the secure HWT sampler (2024/548) based on shuffling. Considering the reports on HWT sampling, the coefficient representation has been applied.
- Multiplication: Polynomial multiplication has also been updated to use coefficient representation instead of its indices, utilizing NTT/Toom-Cook.

For the TiMER parameter, reference implementation has also been updated and the AVX2 implementation of it will be released shortly.

Best regards,
Team SMAUG-T

SMAUG-T 업데이트 코드 결과

clean

Keygen

scheme	avg
SMAUG-Timer	69,234
NEW_SMAUG-Timer	79,115
NEW_SMAUG-T1	84,395
SMAUG-T1	143,625
NEW_SMAUG-T3	160,405
SMAUG-T3	193,935
NEW_SMAUG-T5	232,659
SMAUG-T5	1,566,958

Encap

scheme	avg
NEW_SMAUG-T1	61,518
NEW_SMAUG-Timer	61,666
SMAUG-T1	66,403
SMAUG-Timer	76,165
NEW_SMAUG-T3	126,248
NEW_SMAUG-T5	206,697
SMAUG-T3	243,970
SMAUG-T5	1,663,364

Decap

scheme	avg
NEW_SMAUG-Timer	81,694
NEW_SMAUG-T1	82,241
SMAUG-T1	104,474
NEW_SMAUG-T3	157,720
SMAUG-T3	421,956

키생성을 제외하고는 모두 성능 향상
디캡슐화에서 가장 큰 폭으로 성능 향상

AVX2

scheme	avg
NEW_SMAUG-T1 kem 90s	22,611
NEW_SMAUG-T3 kem 90s	31,190
NEW_SMAUG-T1 kem	38,745
SMAUG -T1 (KEM)	41,139
NEW_SMAUG-T5 kem 90s	41,279
NEW_SMAUG-T3 kem	53,750
SMAUG -T3 (KEM)	64,350
NEW_SMAUG-T5 kem	67,161
SMAUG -T1 (KEM90s)	70,489
SMAUG -T3 (KEM90s)	91,033
SMAUG -T5 (KEM90s)	108,447
SMAUG -T5 (KEM)	146,222

scheme	avg
New_SMAUG-T1 kem 90s	15,754
New_SMAUG-T1 kem	19,517
New_SMAUG-T3 kem 90s	25,610
SMAUG -T1 (KEM)	33,704
New_SMAUG-T5 kem 90s	35,141
SMAUG -T1 (KEM90s)	35,396
New_SMAUG-T3 kem	38,019
SMAUG -T3 (KEM90s)	45,414
New_SMAUG-T5 kem	52,113
SMAUG -T3 (KEM)	55,982
SMAUG -T5 (KEM90s)	62,174
SMAUG -T5 (KEM)	107,760

scheme	avg
New_SMAUG-T1 kem 90s	25,363
New_SMAUG-T1 kem	29,091
SMAUG -T1 (KEM90s)	39,227
New_SMAUG-T3 kem 90s	39,697
New_SMAUG-T3 kem	51,584
New_SMAUG-T5 kem 90s	52,377
SMAUG -T3 (KEM90s)	57,594
SMAUG -T3 (KE	
SMAUG -T1 (KE	
SMAUG -T5 (KEM)	109,709

모든 과정에서 성능 향상
키생성, 캡슐화 디캡슐화에서 각각 1.8배, 2.2배 그리고 1.5배 성능 향상

PALOMA 업데이트 관련

==========

[GenKeyPair]

1. Computation of parity-check matrix
 - Ver.1.1: Naive matrix multiplication
 - Ver.1.2: Polynomial evaluation-based matrix multiplication

2. Row-Column switching during RREF Generation
 - Ver.1.1: Naive method
 - Ver.1.2: Bit mask method for matrix transpose

[Decap]

1. Decap: Polynomial multiplication
 - Ver.1.1: Textbook multiplication
 - Ver.1.2: Karatsuba multiplication

2. Decap: Polynomial modular multiplication
 - Ver.1.1: Textbook multiplication, then naive reduction
 - Ver.1.2: Textbook multiplication and naive reduction with fixed polynomial length (for constant time)

3. Decap: Computation of modular inverse of polynomial
 - Ver.1.1: Extended Euclid algorithm
 - Ver.1.2: Modular exponentiation (for constant time)

4. Decap: Finding error vector from error locator polynomial σ(X)
 - Ver.1.1: Exhaustive search using polynomial evaluation
 - Ver.1.2: Gao-Mateer additive FFT

5. Decap: Get degree of polynomial
 - Ver.1.1: Naive method
 - Ver.1.2: Bit operation (for constant time)

6. Decap: Checking if Hamming weight is t
 - Ver.1.1: -
 - Ver.1.2: If the degree of σ(X) is not t, the extended Patterson decoder returns a zero vector in constant time.

7. Decap: In the case where g12(X) = 1
 - Ver.1.1: -
 - Ver.1.2: Extended Patterson returns a zero vector.

PALOMA 업데이트 코드 결과 (clean)

Keygen

Testing Environment1(clean)

scheme	avg
NEW_PALOMA128	55,486,704
PALOMA128	131,189,151
NEW_PALOMA192	248,335,754
NEW_PALOMA256	283,806,633
PALOMA256	769,657,392
PALOMA192	650,967,499

Testing Environment2(clean)

scheme	avg
NEW_PALOMA128	65,025,193
PALOMA128	147,357,235
NEW_PALOMA192	308,497,763
NEW_PALOMA256	328,543,370
PALOMA192	700,965,299
PALOMA256	819,112,319

Testing Environment3(clean)

scheme	avg
NEW_PALOMA128	55,704,862
PALOMA128	119,572,480
NEW_PALOMA192	274,419,093
NEW_PALOMA256	321,854,651
PALOMA192	561,481,159
PALOMA256	662,511,849

Encap

scheme	avg
NEW_PALOMA128	125,725
PALOMA128	133,345
NEW_PALOMA192	175,054
PALOMA192	176,834
NEW_PALOMA256	207,753
PALOMA256	215,487

scheme	avg
PALOMA128	87,441
NEW_PALOMA128	107,373
PALOMA192	136,963
NEW_PALOMA256	161,877
NEW_PALOMA192	170,696
PALOMA256	200,987

scheme	avg
PALOMA128	58,133
NEW_PALOMA128	59,438
PALOMA192	83,030
NEW_PALOMA192	91,479
PALOMA256	97,140
NEW_PALOMA256	100,780

Decap

scheme	avg
NEW_PALOMA128	2,447,435
PALOMA128	8,424,379
NEW_PALOMA256	9,509,082
NEW_PALOMA192	9,549,977
PALOMA192	41,793,889
PALOMA256	43,735,841

scheme	avg
NEW_PALOMA128	2,868,445
PALOMA128	8,
NEW_PALOMA256	
NEW	
PALOMA192	41,802,948
PALOMA256	45,735,541

scheme	avg
NEW_PALOMA128	2,020,284
A192	7,610,358
PALOMA192	39,835,958
PALOMA256	41,789,539

모든 과정에서 성능 향상

키생성과 디캡슐화에서 각각 2.4배 그리고 3.4배 성능 향상

NTRU+ 업데이트 관련

Changes to the implementation are as follows:

1. Source Code for the Hash Functions
We replaced the source code of SHA256 and additionally used the source code for SHAKE256, adapted from https://github.com/kpqc-cryptocraft/KpqClean_ver2.

2. Changes in the Key Generation
To improve the efficiency of key generation, we adopted an early abort approach when checking the invertibility of a polynomial.

When checking the invertibility of a polynomial, we need to verify that it is invertible in all rings Z_q[x]/⟨x^d − ₩zeta_i⟩. For efficiency, we abort as soon as we find the first ring in which the polynomial is not invertible.

One may wonder whether this type of early abort could leak information about the randomness used to sample the polynomial.

However, since the rejected polynomial is not reused as part of the secret key, we believe this approach is secure, provided that the underlying randombytes function is forward-secure.

3. Changes in Ring Multiplication and Inversion
To improve the efficiency of key generation, we adopted lazy Montgomery reduction in the implementation of ring operations (multiplication and inversion) in ₩prod_{i=0}^{n/d−1} Z_q[x]/⟨x^d −w⟩ for d = 3, 4.

Also, to enhance the efficiency of the modular inversion using Fermat's Little Theorem, a −1 ≡ a q−2 (mod q), we leveraged the binary structure of q − 2 = 3455 = 110101111111(2), inspired by the fast modular inversion in Curve25519.

Thank you.

Best regards,
NTRU+ Team

NTRU+ 업데이트 코드 결과

clean

Keygen

scheme	avg
NEW_NTRU+KEM576	178,774
NEW_NTRU+PKE576	180,894
NEW_NTRU+PKE768	210,248
NEW_NTRU+KEM768	210,261
NEW_NTRU+KEM864	230,556
NEW_NTRU+PKE864	230,793
NTRU+KEM576	253,031
NTRU+PKE768	310,102
NTRU+PKE576	329,712
NEW_NTRU+PKE1152	344,151

Encap

scheme	avg
NEW_NTRU+PKE576	77,588
NEW_NTRU+KEM576	82,246
NTRU+PKE576	83,245
NTRU+KEM576	85,541
NTRU+KEM768	106,690
NTRU+PKE768	107,632
NEW_NTRU+KEM768	115,843
NEW_NTRU+PKE768	115,866
NEW_NTRU+KEM864	121,771
NEW_NTRU+PKE864	123,215
NTRU+PKE864	128,465
NTRU+KEM864	128,823
NEW_NTRU+PKE1152	153,492
NEW_NTRU+KEM1152	157,981
NTRU+PKE1152	164,153
NTRU+KEM1152	164,274

Decap

scheme	avg
NEW_NTRU+KEM576	73,735
NEW_NTRU+PKE576	75,289
NTRU+PKE576	107,302
NTRU+KEM576	110,140
NEW_NTRU+KEM768	114,679
NEW_NTRU+PKE768	114,768
NEW_NTRU+KEM864	124,974
NEW_NTRU+PKE864	125,821
NTRU+PKE768	140,803
NTRU+KEM768	141,220
NEW_NTRU+PKE1152	159,135
NEW_NTRU+KEM1152	161,589
NTRU+PKE86	
NTRU+KEM86	

키생성에서 성능 향상

키생성 그리고 디캡슐화에서 각각 1.4배 성능 향상

AVX2

scheme	avg
New_NTRU+PKE576	34,697
New_NTRU+KEM576	34,909
NTRU+PKE576	34,911
NTRU+KEM576	35,261
New_NTRU+KEM864	39,844
NTRU+KEM864	40,050
NTRU+PKE864	40,681
New_NTRU+PKE864	40,851
New_NTRU+KEM768	53,887
NTRU+KEM768	56,223
New_NTRU+PKE768	56,783
NTRU+PKE768	57,667
New_NTRU+KEM1152	
NTRU	
NTRU	
New_NTRU	

scheme	avg
NTRU+KEM576	20,453
New_NTRU+KEM576	20,572
NTRU+PKE576	20,575
New_NTRU+PKE576	20,601
NTRU+PKE768	26,708
NTRU+KEM768	27,029
New_NTRU+PKE768	27,032
New_NTRU+KEM768	27,070
NTRU+PKE864	29,554
NTRU+KEM864	29,813
New_NTRU+PKE864	29,828
New_NTRU+KEM864	29,864
NTRU+PKE1152	37,910
New_NTRU+PKE1152	38,536
New_NTRU+KEM1152	38,637

scheme	avg
NTRU+KEM576	12,695
New_NTRU+KEM576	12,783
NTRU+PKE576	13,358
New_NTRU+PKE576	13,358
NTRU+KEM768	17,053
New_NTRU+PKE768	17,088
New_NTRU+KEM768	17,119
NTRU+PKE768	17,502
NTRU+KEM864	19,269
New_NTRU+KEM864	19,272
New_NTRU+PKE864	19,369
NTRU+PKE	
New_NTRU+PI	
New_NTRU+KEM1152	24,748
NTRU+KEM1152	24,794
NTRU+PKE1152	25,363

큰 성능 변화는 없음

NCC-Sign 업데이트 관련

NCC-Sign
- Fixed SUPERCOP checksum issue in the AVX2 optimized version.
- Bug in the rejection process has been fixed.
- Applied some suggestions and patches from Prof. D. J. Bernstein.
- Improved the reference source code structure for better performance.
- Released Cortex-M4 optimized version.

NCC-SIGN 업데이트 코드 결과

clean

Keygen

scheme	avg
NEW_NCC-Sign3_shake_NTT_MODE_1	194,578
NEW_NCC-Sign5_shake_NTT_MODE_1	201,215
NEW_NCC-Sign1_shake_NTT_MODE_1	201,325
NEW_NCC-Sign3_shake_NTT_MODE_3	245,099
NEW_NCC-Sign5_shake_NTT_MODE_3	248,687
NEW_NCC-Sign1_shake_NTT_MODE_3	273,995
NCC-Sign1	300,401
NEW_NCC-Sign1-AES_NTT_MODE_1	323,828
NCC-Sign3	352,126
NEW_NCC-Sign5_shake_NTT_MODE_5	375,236
NEW_NCC-Sign3_shake_NTT_MODE_5	376,189
NEW_NCC-Sign1_shake_NTT_MODE_5	380,698
NCC-Sign3-AES_NTT_MODE_1	423,110
NCC-Sign1-AES_NTT_MODE_3	423,451
NEW_NCC-Sign3-AES_NTT_MODE_3	423,841
NEW_NCC-Sign5-AES_NTT_MODE_3	428,798
NCC-Sign5	455,065

Encap

scheme	avg
NEW_NCC-Sign3_shake_NTT_MODE_1	422,889
NEW_NCC-Sign5_shake_NTT_MODE_1	425,517
NCC-Sign1	479,198
NEW_NCC-Sign5_shake_NTT_MODE_3	505,538
NCC-Sign3	600,143
NEW_NCC-Sign3-AES_NTT_MODE_1	742,379
NEW_NCC-Sign5_shake_NTT_MODE_5	811,556
NEW_NCC-Sign3_shake_NTT_MODE_3	947,492
NEW_NCC-Sign3_shake_NTT_MODE_5	1,150,546
NCC-Sign1-AES_NTT_MODE_1	1,216,509
NCC-Sign1_shake_NTT_MODE_1	1,238,036
NEW_NCC-Sign1_shake_NTT_MODE_3	1,346,835
NEW_NCC-Sign3-AES_NTT_MODE_3	1,384,464
NEW_NCC-Sign5-AES_NTT_MODE_1	1,528,635
NCC-Sign5	1,585,378

Decap

scheme	avg
NEW_NCC-Sign5_shake_NTT_MODE_1	216,628
NEW_NCC-Sign3_shake_NTT_MODE_1	217,872
NEW_NCC-Sign1_shake_NTT_MODE_1	221,245
NEW_NCC-Sign1_shake_NTT_MODE_3	252,170
NEW_NCC-Sign5_shake_NTT_MODE_3	262,487
NEW_NCC-Sign3_shake_NTT_MODE_3	262,991
NCC-Sign1	285,502
NEW_NCC-Sign1-AES_NTT_MODE_1	324,448
NCC-Sign3	356,543
NCC-Sign1_shake_NTT_MODE_5	409,797
NEW_NCC-Sign5_shake_NTT_MODE_5	410,587
NEW_NCC-Sign3_shake_NTT_MODE_5	415,953
NEW_NCC-Sign3-AES_NTT_MODE_5	623,024

키생성에서 1.5배 성능 향상

캡슐화와 디캡슐화에서도 성능향상

AVX2

scheme	avg
New_NCC-Sign3-AES_NTT_MODE_1	46,139
New_NCC-Sign5-AES_NTT_MODE_1	51,753
New_NCC-Sign1-AES_NTT_MODE_1	52,061
New_NCC-Sign5-AES_NTT_MODE_3	55,441
New_NCC-Sign1-AES_NTT_MODE_3	61,174
New_NCC-Sign3-AES_NTT_MODE_3	61,816
New_NCC-Sign5-AES_NTT_MODE_5	83,253
New_NCC-Sign3-AES_NTT_MODE_5	85,755
New_NCC-Sign3_shake_NTT_MODE_1	86,524
New_NCC-Sign5_shake_NTT_MODE_1	86,864
New_NCC-Sign1_shake_NTT_MODE_1	86,909
New_NCC-Sign1-AES_NTT_MODE_5	91,723
New_NCC-Sign5_shake_NTT_MODE_3	107,482
New_NCC-Sign1_shake_NTT_MODE_3	107,588
New_NCC-Sign3_shake_NTT_MODE_3	107,782
NCC-Sign1	137,199
New_NCC-Sign5_shake_NTT_MODE_5	
New_NCC-Sign1_shake_NTT_MODE_5	
New_NCC-Sign3_shake_NTT_MODE_5	
NCC-Sign3	
NCC-Sign5	

scheme	avg
New_NCC-Sign5-AES_NTT_MODE_1	74,930
New_NCC-Sign5-AES_NTT_MODE_3	100,381
New_NCC-Sign1_shake_NTT_MODE_1	121,967
New_NCC-Sign5_shake_NTT_MODE_1	123,040
New_NCC-Sign3-AES_NTT_MODE_3	151,906
New_NCC-Sign5-AES_NTT_MODE_5	156,623
New_NCC-Sign1-AES_NTT_MODE_1	159,687
New_NCC-Sign1_shake_NTT_MODE_3	161,059
New_NCC-Sign5_shake_NTT_MODE_3	163,247
New_NCC-Sign3_shake_NTT_MODE_1	177,436
NCC-Sign1	207,323
:S_NTT_MODE_3	211,760
ake_NTT_MODE_3	229,961
ake_NTT_MODE_5	245,694
ake_NTT_MODE_5	245,829
ake_NTT_MODE_5	246,718
:S_NTT_MODE_1	285,581
:S_NTT_MODE_5	325,009
gn3	374,557
gn5	422,955
:S_NTT_MODE_5	480,938

scheme	avg
New_NCC-Sign1-AES_NTT_MODE_1	58,010
New_NCC-Sign5-AES_NTT_MODE_1	58,208
New_NCC-Sign3-AES_NTT_MODE_1	58,267
New_NCC-Sign5-AES_NTT_MODE_3	70,806
New_NCC-Sign1-AES_NTT_MODE_3	70,894
New_NCC-Sign3-AES_NTT_MODE_3	71,563
New_NCC-Sign1_shake_NTT_MODE_1	88,848
New_NCC-Sign5_shake_NTT_MODE_1	89,230
New_NCC-Sign3_shake_NTT_MODE_1	90,329
New_NCC-Sign1-AES_NTT_MODE_5	109,805
New_NCC-Sign3-AES_NTT_MODE_5	109,898
New_NCC-Sign5-AES_	
New_NCC-Sign3_shake_	
New_NCC-Sign3_shake_NTT_MODE_5	171,433

모든 과정에서 성능 향상

키생성, 서명, 검증에서 각각 2.9배, 2.7배 그리고 3.5배 성능 향상

MQ-Sign 업데이트 관련

MQ-Sign
- Made Crypto API identical to the NIST PQC standard.
- Changes to Makefile rules allow for more concise compilation.
- By modifying the calculation process, performance is slightly improved.
- Removed some legacy parts of the public key, partially reducing the PK size.
- Applied some suggestions and patches from Prof. D. J. Bernstein.

MQ-SIGN 업데이트 코드 결과

clean

Keygen

scheme	avg
MQ-Sign-MQLR-256-72-46	74,764,874
NEW_MQLR_256_72_46	75,382,639
NEW_MQRR_o_72_46	97,426,775
NEW_MQRR_s_72_46	100,871,813
NEW_MQRR_m_72_46	100,898,767
MQ-Sign-MQRR-256-72-46	100,975,612
MQ-Sign-MQLR-256-112-72	286,180,511
NEW_MQLR_256_112_72	292,262,140
NEW_MQRR_s_112_72	374,184,501
NEW_MQRR_o_112_72	374,358,898
NEW_MQRR_m_112_72	382,777,875
MQ-Sign-MQRR-256-112-72	387,732,599
MQ-Sign-MQLR-256-148-96	750,300,850
NEW_MQLR_256_148_96	753,283,752
NEW_MQRR_o_148_96	957,171,748
NEW_MQRR_s_148_96	957,470,061
NEW_MQRR_m_148_96	970,586,332
MQ-Sign-MQRR-256-148-96	997,169,889

Encap

scheme	avg
MQ-Sign-MQLR-256-72-46	463,964
NEW_MQLR_256_72_46	483,044
MQ-Sign-MQRR-256-72-46	809,523
NEW_MQRR_o_72_46	820,027
NEW_MQRR_s_72_46	820,539
NEW_MQRR_m_72_46	823,567
MQ-Sign-MQLR-256-112-72	1,289,973
NEW_MQLR_256_112_72	1,307,986
NEW_MQRR_m_112_72	2,035,399
NEW_MQRR_s_112_72	2,036,364
MQ-Sign-MQRR-256-112-72	2,057,398
NEW_MQRR_o_112_72	2,058,603
MQ-Sign-MQLR-256-148-96	2,762,744
NEW_MQLR_256_148_96	2,780,855

Decap

scheme	avg
NEW_MQLR_256_72_46	690,874
NEW_MQRR_s_72_46	691,622
NEW_MQRR_m_72_46	697,515
NEW_MQRR_o_72_46	698,096
MQ-Sign-MQRR-256-72-46	706,565
MQ-Sign-MQLR-256-72-46	733,587
MQ-Sign-MQRR-256-112-72	1,986,691
MQ-Sign-MQLR-256-112-72	2,016,272
NEW_MQRR_s_112_72	2,019,703
NEW_MQLR_256_112_72	2,023,707
NEW_MQRR_m_112_72	2,027,657
NEW_[illegible]	[illegible]
NEW_[illegible]	[illegible]
NEW_[illegible]	[illegible]
MQ-Sign-MQRR-256-148-96	4,272,864
MQ-Sign-MQLR-256-148-96	4,307,611

키생성 그리고 캡슐화에서 성능 하락

디캡슐화에서 성능 향상

AVX2

Keygen

scheme	avg
New_MQRR_o_72_46	2,928,163
New_MQLR_256_72_46	3,679,090
New_MQRR_m_72_46	4,061,374
New_MQRR_s_72_46	4,868,979
New_MQRR_s_112_72	13,086,048
New_MQRR_o_112_72	13,107,964
New_MQLR_256_112_72	16,766,445
New_MQRR_m_112_72	18,176,437
New_MQLR_256_148_96	44,547,526
New_MQRR_s_148_96	44,820,296
New_MQRR_o_148_96	45,153,897
MQLR-256-72-46	4,947,896
New_MQRR_m_148_96	55,220,680

Encap

scheme	avg
New_MQLR_256_72_46	46,695
New_MQRR_o_72_46	63,076
New_MQRR_s_72_46	63,269
New_MQRR_m_72_46	63,741
MQRR-256-72-46	76,323
New_MQLR_256_112_72	124,081
MQLR-256-112-72	164,022
New_MQRR_m_112_72	170,728
New_MQRR_o_112_72	170,938
New_MQRR_s_112_72	173,875
MQRR-256-112-72	206,977
New_MQLR_256_148_96	207,041
MQLR-256-148-96	260,591
New_MQRR_s_148_96	294,936

Decap

scheme	avg
New_MQRR_o_72_46	34,569
New_MQLR_256_72_46	34,585
New_MQRR_m_72_46	34,799
MQRR-256-72-46	44,568
MQLR-256-72-46	45,235
New_MQRR_s_72_46	35,024
New_MQRR_m_112_72	110,676
New_MQRR_s_112_72	110,792
New_MQRR_o_112_72	110,987
New_MQLR_256_112_72	114,097
MQRR-256-112-72	141,516
MQLR-256-112-72	162,990
[illegible]	273
[illegible]	476
MQLR-256-148-96	279,745
MQRR-256-148-96	290,724

모든 과정에서 성능 향상

키생성, 서명, 검증에서 각각 1.6배, 1.5배 그리고 1.2배 성능 향상

HAETAE 업데이트 관련

Hyeongmin Choe

받는사람 KpqC-bulletin

Hi all,

Haetae has recently been updated to v3.0. Please find it on our website: https://www.kpqc.cryptolab.co.kr/haetae

To summarize the changes,

- We corrected the private key sizes in the specification, which were missing 32 bytes.
- B′ has been reduced by 0.01 to satisfy Lemma 5; the effect on the implementation is negligible.
- We significantly improved the key generation procedure and achieved 40-60% reduced clock cycles in AVX2 implementation.

Best,
Team Haetae

HAETAE 업데이트 코드 결과

clean

Keygen

scheme	avg
NEW_HAETAE2	1,060,674
HAETAE2	1,160,435
NEW_HAETAE5	1,379,694
NEW_HAETAE3	1,796,195
HAETAE5	1,998,427
HAETAE3	2,103,397

Sign

scheme	avg
HAETAE3	2,739,863
NEW_HAETAE3	2,740,577
HAETAE5	3,369,567
NEW_HAETAE2	3,911,718
HAETAE2	5,032,357
NEW_HAETAE5	5,300,714

Verify

scheme	avg
NEW_HAETAE2	155,656
HAETAE2	158,529
HAETAE3	275,419
NEW_HAETAE3	286,273
NEW_HAETAE5	337,215

서명을 제외하고는 모두 성능 향상

AVX2

Keygen

scheme	avg
New_HAETAE2	453,534
New_HAETAE3	657,020
New_HAETAE5	703,772
HAETAE2	834,651
HAETAE3	1,423,068
HAETAE5	1,924,879

Sign

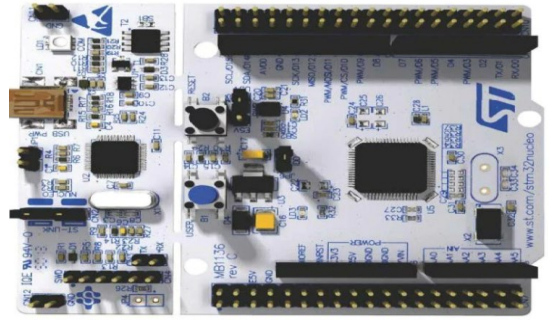
scheme	avg
New_HAETAE2	270,121
New_HAETAE3	377,726
HAETAE3	374,489
HAETAE5	1,077,729
New_HAETAE5	1,091,046
HAETAE2	4,946,575

Verify

scheme	avg
HAETAE2	67,068
New_HAETAE2	69,956
New_HAETAE3	109,668
HAETAE3	128,981
New_HAETAE5	133,173
HAETAE5	134,969

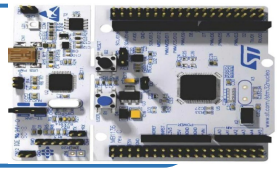
키생성 과정에서 1.84배 성능 향상

메모리 사용량 관점에서의 분석



메모리 사용량 측정 (PKE/KEM)

수십 KB가 저전력 장비에 적합한 메모리 사용량
*REDOG은 측정 불가로 제외



Algorithm	Stack+Heap (bytes)	Stack (bytes)	Heap (bytes)	extra_heap(bytes)
NTRUplus-KEM-576	19,632	18,600	1,024	8
NTRUplus-KEM-768	25,024	23,992	1,024	8
NTRUplus-KEM-864	27,728	26,696	1,024	8
NTRUplus-KEM-1152	35,824	34,792	1,024	8
NTRUplus-PKE-576	19,696	18,664	1,024	8
NTRUplus-PKE-768	PALOMA의 경우 메모리 사용량이 많은 코드 기반 암호의 특징을 공유			8
NTRUplus-PKE-864	27,856	26,824	1,024	8
NTRUplus-PKE-1152	36,016	34,984	1,024	8
PALOMA-w-128	2,201,944	1,943,432	197,377	61,135
PALOMA-w-192	7,900,112	7,641,600	197,377	61,135
PALOMA-w-256	9,282,512	9,024,000	197,377	61,135
PALOMA-wo-128	1,944,464	1,943,432	1,024	8
PALOMA-wo-192	5,303,920	5,302,888	1,024	8
PALOMA-wo-256	9,025,032	9,024,000	1,024	8
SMAUG-T1	12,912	10,368	2,472	72
SMAUG-T3	19,888	17,344	2,472	72
SMAUG-T5	37,888	35,344	2,472	72

메모리 사용량 측정 (Digital Signature)

수십 KB가 저전력 장비에
적합한 메모리 사용량

Algorithm	Stack+Heap (bytes)	Stack (bytes)	Heap (bytes)	extra_heap(bytes)
AIMer-128f	142,256	141,008	1,232	16
AIMer-128s	923,936	922,904	1,024	8
AIMer-192f	313,600	312,352	1,232	16
AIMer-192s	2,035,488	2,034,456	1,024	8
AIMer-256f	649,872	648,624	1,232	16
AIMer-256s	4,182,504	4,181,264	1,232	8
HAETAE-II	369,736	111,224	197,377	61,135
HAETAE-III	AIMER와 MQSIGN의 경우에도 큰 메모리를 사용하는 경향이 있음			61,135
HAETAE-V	481,896	223,384	197,377	61,135
MQSign-MQLR-256-72-46	1,318,960	590,048	728,858	54
MQSign-MQLR-256-112-72	4,963,632	2,219,328	2,744,250	54
MQSign-MQLR-256-148-96	11,584,144	5,187,168	6,396,930	46
MQSign-MQRR-256-72-46	1,185,680	579,400	606,226	54
MQSign-MQRR-256-112-72	4,478,288	2,193,960	2,284,282	46
MQSign-MQRR-256-148-96	10,473,984	5,143,080	5,330,866	38
NCCSign-I	60,712	60,712	0	0
NCCSign-III	80,248	80,248	0	0
NCCSign-V	120,440	120,440	0	0

AIMER 업데이트 관련

We would like to announce AIMer version update v2.1.
The updates are as follows:

Implementation Updates

- updated our implementations to be more friendly to PQClean project and run all tests of PQClean test framework.
- merged Reference C and Optimized C version of the v2.0 implementation to Reference C in the v2.1 implementation.
- renamed (additional) AVX2 implementation to Optimized implementation.
- added mem_opt C implementation targeting on memory-constrained devices.
- added aarch64_shake_opt implementation which utilizes ARM Advanced SIMD instructions on SHAKE.
- aarch64 and aarch64_shake_opt implementations can be compiled for ARM-based Apple SoCs (Apple M series).

AIMER를 임베디드 디바이스 상에서 최적화하기 위한 결과물 외에 다양한 구현 결과 도출

Software Patches for TIMECOP

In response to software recommendations from Prof. D. J. Bernstein, we have applied following patches:

- Since the variables of patch-1-reveal, patch-7-commits, and patch-8-alpha were public data, we have utilized crypto_declassify function.
- patch-2-poly64: added recommended poly64_mul as poly64_mul_s, and applied it to arithmetic related to secret data.
- patch-3-htole: replaced htole64() and ltohe64() with recommended byte computations, and removed the portable_endian.h file.
- patch-4-loadstore: replaced _load_ and _store_ with _loadu_ and _storeu_ in the AVX2 implementation.
- patch-5-square: modified all implementations to use recommended code for square arithmetic in the Reference and mem_opt implementations.
- patch-6-selfaddmask: removed the selfaddmask function from all implementations.
- patch-9-initialize: added the recommended initialization process to the AVX2 implementation.
- Lastly, we have included TIMECOP results for all TIMECOP-supported implementations.

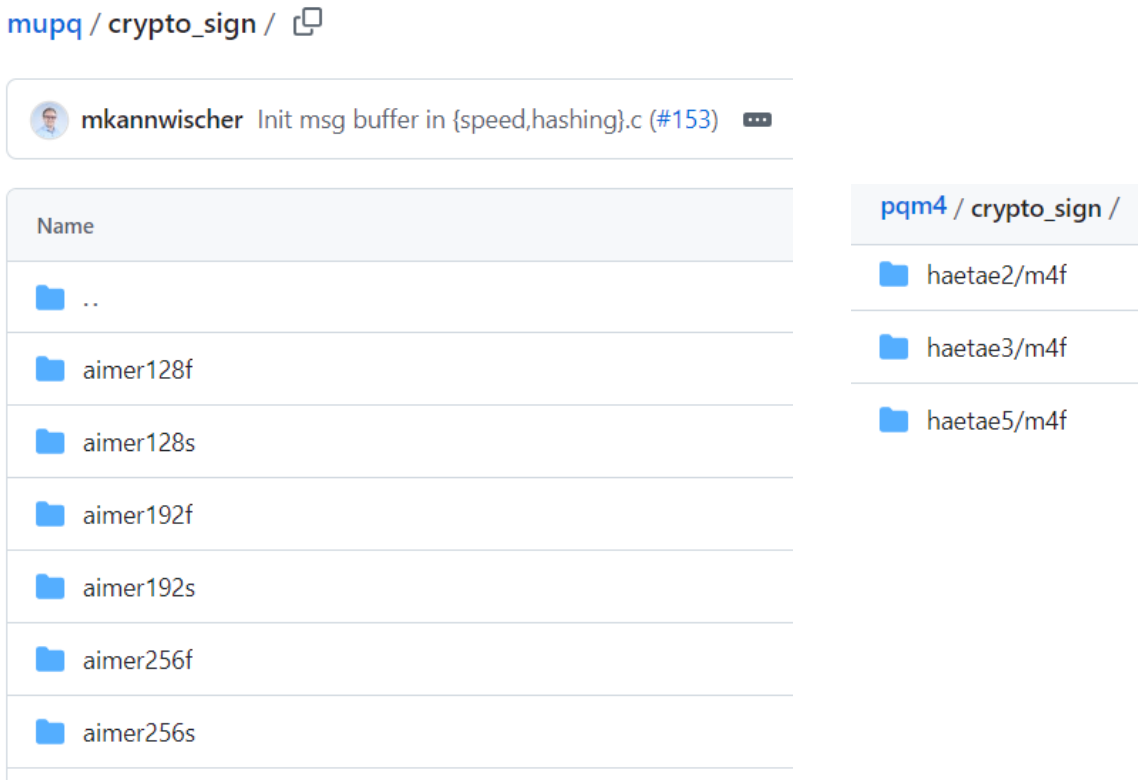
The AIMer specification v2.1 is now available on our website: https://aimer-signature.org/docs/AIMer-specification-v2.1.pdf
This document contains the following performance summary of AIMer v2.1 measured on Intel Core i7-10750H @ 2.6 GHz

Parameters	Implementation	Performance (cycles)			Memory usage (KB)	
		Keygen	Sign	Verify	Sign	Verify
aimer128f	Reference	130,398	4,174,175	3,882,168	129.5	21.5
	mem_opt	130,578	6,549,458	3,850,860	20.2	21.4
	Optimized (AVX2)	87,926	1,289,604	1,233,725	131.0	25.2
aimer128s	Reference	130,302	32,678,777	32,512,507	915.5	38.9
	mem_opt	130,334	51,999,628	32,260,032	32.0	38.9
	Optimized (AVX2)	87,749	9,726,954	9,626,252	924.9	73.4

Additionally, all versions of AIMer are uploaded on https://github.com/samsungsds-research-papers/AIMer

pqm4 상의 KpqC

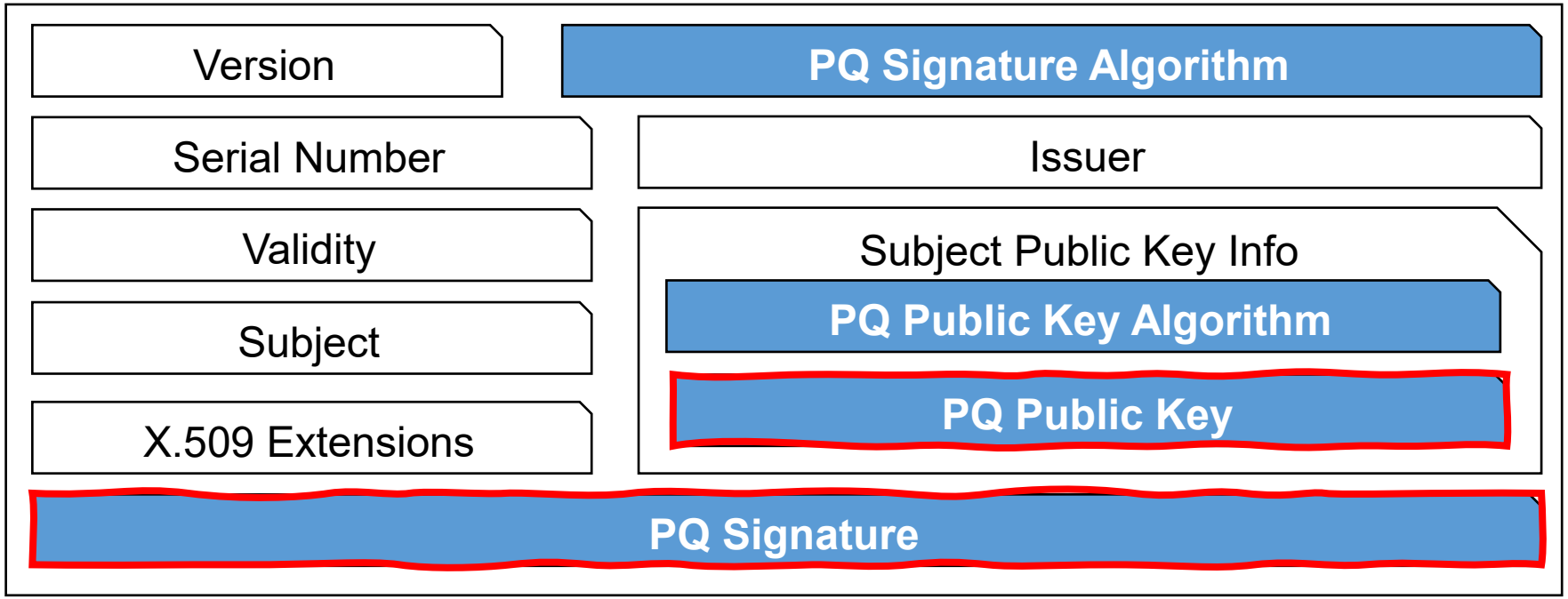
- **haetae와 aimer의 경우 pqm4상에 포함**
- **임베디드 보드에 적합하도록 메모리 이슈를 해결한 구현 가능**
  - **스트리밍 구현 적용을 통해 메모리의 사용량을 제한할 경우 가능**
  - **다만 스트리밍 구현이 KCMVP에 탑재가 될 수 있을까에 대한 의문점…**



메모리 사용량과 PQ X.509

PQ X.509

- **X.509 : 공개 키 인프라(PKI) 인증서의 형식을 정의하는 표준**
  - 다양한 인터넷 프로토콜, 특히 HTTPS와 SSL/TLS 같은 암호화된 통신 프로토콜에서 중요한 역할
- X.509를 PQ X.509 전환 시 **PQ 서명 알고리즘, PQ 공개 키 알고리즘, PQ 공개키, PQ 서명**을 포함한 핵심 요소 포함해야 함



PQ X.509

- **PQ 공개 키와 PQ 서명의 크기는 인증서의 전체 크기에 상당한 영향을 미치기 때문에 PQC 알고리즘을 선택하는 것은 중요한 문제**
  - 인증서가 효율적이고 관리 가능한 상태를 유지하면서 양자 위협에 대한 필요한 보안을 제공하는 데 필수적
- **따라서, PQ X.509 인증서에서 Legacy DSA 알고리즘과 KpqC DSA 알고리즘에 대한 공개키와 서명 크기의 최적의 조합 평가 (하이브리드 형식)**
  - 기존 X.509 표준에서 PQ X.509로 전환하는 할 때의 고려사항 분석
  - **Legacy DSA 알고리즘 :**

    RSA-2048, RSA-3072, secp256r1(P-256), secp384r1(P-384), secp521r1(P-521) 분석
  - KpqC와 NIST PQC를 비교하기 위해서 NIST PQC 표준화 알고리즘에 대해서도 분석

KpqC PKE/KEM, Digital Signature 파라미터

PKE/KEM (byte)			
Scheme	PUBLICKEY	SECRETKEY	CIPHERTEXT
NTRU+KEM576	864	1,760	864
NTRU+KEM768	1,152	2,336	1,152
NTRU+KEM864	1,296	2,624	1,296
NTRU+KEM1152	1,728	3,488	1,728
NTRU+PKE576	864	1,760	864
NTRU+PKE768	1,152	2,336	1,152
NTRU+PKE864	1,296	2,624	1,296
NTRU+PKE1152	1,728	3,488	1,728
PALOMA128	319,488	94,528	136
PALOMA192	812,032	357,568	240
PALOMA256	1,025,024	359,616	240
SMAUG-T1	672	832	672
SMAUG-T3	1,088	1,312	992
SMAUG-T5	1,440	1,792	1,376
SMAUG-Timer	672	832	608

Digital Signature (byte)			
Scheme	PUBLICKEY	SECRETKEY	SIGNATURE
HAETAE2	992	1,408	1,474
HAETAE3	1,472	2,112	2,349
HAETAE5	2,080	2,752	2,948
AIMer128f	32	48	5,888
AIMer128s	32	48	4,160
AIMer192f	48	72	13,056
AIMer192s	48	72	9,120
AIMer256f	64	96	25,120
AIMer256s	64	96	17,056
MQ-Sign-MQLR -256-72-46	328,441	160,881	150
MQ-Sign-MQLR -256-112-72	1,238,761	601,249	216
MQ-Sign-MQLR -256-148-96	2,892,961	1,400,113	276
MQ-Sign-MQRR -256-72-46	328,505	276,649	150
MQ-Sign-MQRR -256-112-72	1,238,825	1,044,385	216
MQ-Sign-MQRR -256-148-96	2,893,025	2,436,769	276
NCC-Sign1			2,912
NCC-Sign3	2,336	3,552	3,872
NCC-Sign5	3,200	5,568	6,080

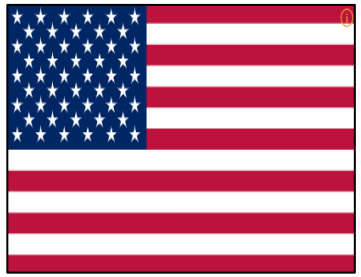
X.509에서는 DSA의 크기가 중요

PQ X.509

- **X.509에서의 적합한 알고리즘 비교를 위한 공개키 + 서명의 크기 비교**
  - **FALCON이 가장 조합의 크기가 우수**
  - **하이브리드 조합으로는 RSA보다는 ECC가 좋음**
  - 국가 기관에서 국내 표준으로 선정된 알고리즘을 사용하는 경우, HAETAE와 AIMer 사용이 적합할 수 있음

Rank	Scheme	PK+Sig
1	P256+FALCON-512	1,659
2	P384+FALCON-512	1,707
3	P512+FALCON-512	1,758
4	RSA2048+FALCON-512	2,075
5	RSA3072+FALCON-512	2,331
6	**P256+HAETAE2**	**2,562**
7	**P384+ HAETAE2**	**2,610**
8	**P512+HAETAE2**	**2,661**
9	**RSA2048+HAETAE2**	**2,978**
10	P256+FALCON-1024	3,169
11	P384+FALCON-1024	3,217
12	**RSA3072+HAETAE2**	**3,234**
13	P512+FALCON-1024	3,268
14	RSA2048+FALCON-1024	3,585
15	P256+Dilithium-II	3,828

Rank	Scheme	PK+Sig
16	RSA3072+FALCON-1024	3,841
17	P384+Dilithium-II	3,876
18	**P256+HAETAE3**	**3,917**
19	P512+Dilithium-II	3,927
20	**P384+ HAETAE3**	**3,965**
21	**P512+HAETAE3**	**4,016**
22	RSA2048+Dilithium-II	4,244
23	**P256+AIMer128s**	**4,288**
24	**RSA2048+HAETAE3**	**4,333**
25	P384+ AIMer128s	4,336
26	P512+AIMer128s	4,387
27	RSA3072+Dilithium-II	4,500
28	**RSA3072+HAETAE3**	**4,589**
29	**RSA2048+AIMer128s**	**4,704**
30	**P256+NCC-Sign1**	**4,768**



NIST PQC vs KpqC 성능 비교

KpqC 공모전 이후…

- **KpqC 공모전 선정결과는 어떻게 활용될 것인가? (실제로 산업체에서…)**
  - 아마도 KCMVP 암호모듈?

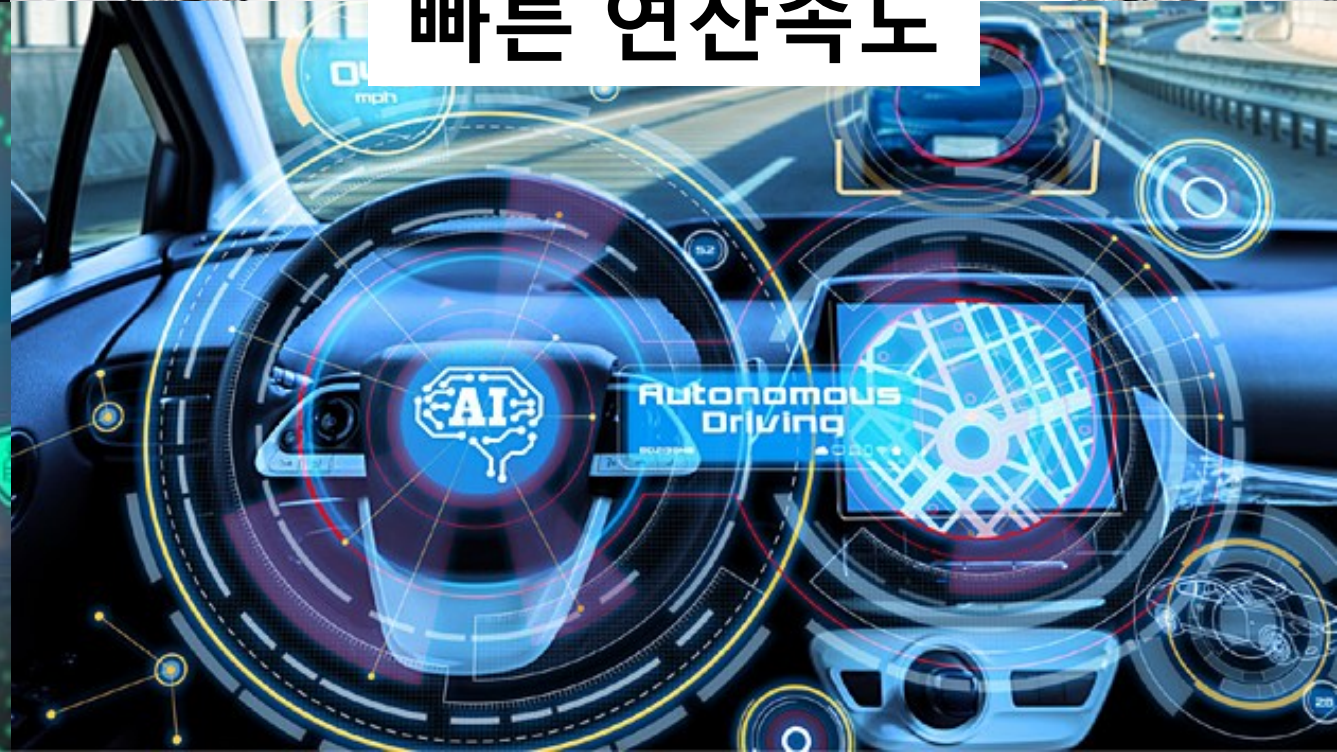
- **현재 KCMVP에서는 국제 표준 공개키 (RSA, ECC)와 국산 알고리즘 일부 (KCDSA, EC-KCDSA)를 사용 중**

- **가능한 시나리오**
  - KpqC 알고리즘만 KCMVP에 등록
  - NIST PQC 알고리즘만 KCMVP에 등록
  - **NIST PQC와 KpqC 알고리즘을 모두 KCMVP에 등록**
    → 해당 경우 NIST PQC 대신 KpqC가 널리 사용되려면 우리가 결정하기 어려운 여러가지 외부적인 요인 (표준화 및 국제화 지수) 고려 필요
    → 실제 서비스 상에서 구현성능이 우수해야 할 것으로 판단 (AES 공공기관 도입)

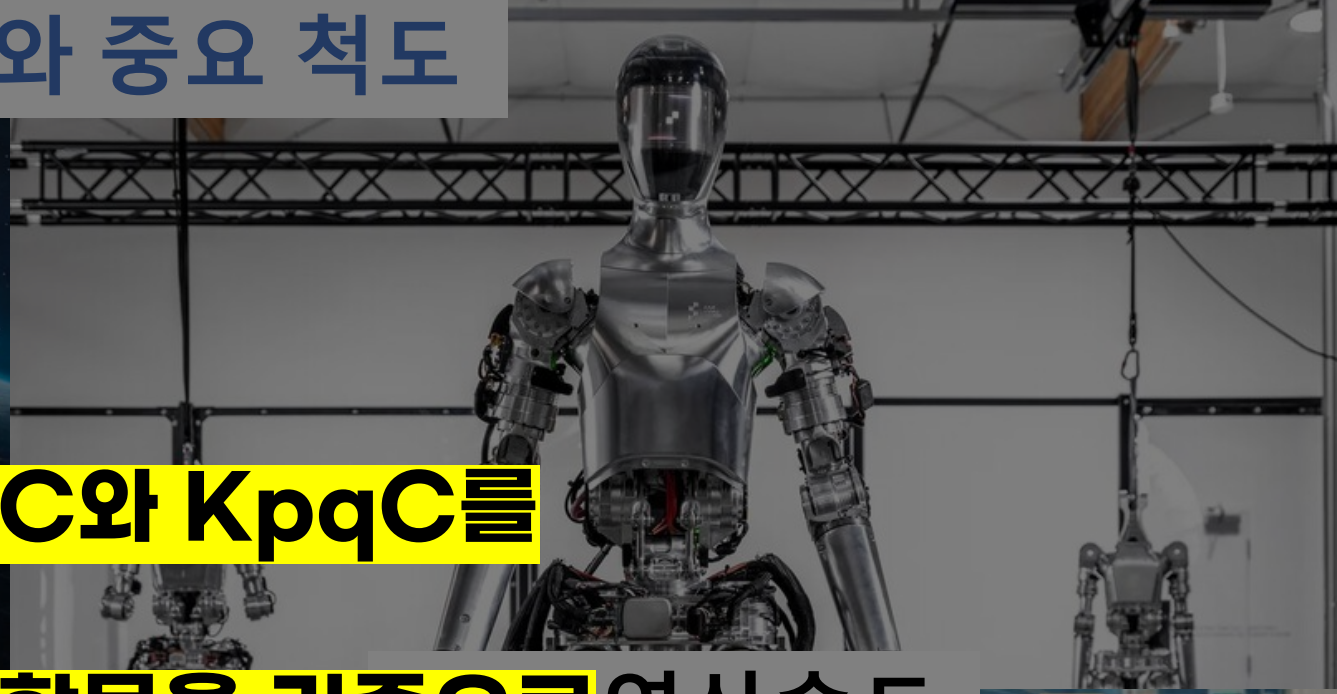
미래 서비스와 중요 척도

작은 네트워크 패킷

빠른 연산속도



미래 서비스와 중요 척도



NIST PQC와 KpqC를

작은 네트워크 두가지 평가 항목을 기준으로 연산속도

비교 분석 수행

Kyber vs SMAUG-T NTRU+ 키 크기 비교

비밀키는 전송 이슈가 없으므로 크리티컬하지 않음

Rank	Scheme	Public key
1	SMAUG-T1	672
	SMAUG-Timer	672
3	KYBER-512	800
4	NTRU+KEM576	864
	NTRU+PKE576	864
6	SMAUG-T3	1,088
7	NTRU+KEM768	1,152
	NTRU+PKE768	1,152
9	KYBER-768	1,184
10	NTRU+KEM864	1,296
	NTRU+PKE864	1,296
12	SMAUG-T5	1,440
13	KYBER-1024	1,568
14	NTRU+KEM1152	1,728
	NTRU+PKE1152	1,728

Rank	Scheme	Secret key
1	SMAUG-T1	832
	SMAUG-Timer	832
3	SMAUG-T3	1,312
4	KYBER-512	1,632
5	NTRU+KEM576	1,760
	NTRU+PKE576	1,760
7	SMAUG-T5	1,792
8	NTRU+KEM768	2,336
	NTRU+PKE768	2,336
10	KYBER-768	2,400
11	KYBER-1024	2,592
12	NTRU+KEM864	2,624
	NTRU+PKE864	2,624
14	NTRU+KEM1152	3,488
	NTRU+PKE1152	3,488

Rank	Scheme	Ciphertext
1	SMAUG-Timer	608
2	SMAUG-T1	672
3	KYBER-512	768
4	NTRU+KEM576	864
	NTRU+PKE576	864
6	SMAUG-T3	992
7	KYBER-768	1,088
8	NTRU+KEM768	1,152
	NTRU+PKE768	1,152
10	NTRU+KEM864	1,296
	NTRU+PKE864	1,296
12	SMAUG-T5	1,376
13	KYBER-1024	1,568
14	NTRU+KEM1152	1,728
	NTRU+PKE1152	1,728

공개키와 암호문 관점에서 Kyber와 유사하게 작음

McEliece HQC BIKE vs PALOMA REDOG 키 크기 비교

Rank	Scheme	Public key
1	HQC-128	2,249
2	REDOG-1	4,270
3	HQC-192	4,522
4	HQC-256	7,245
5	BIKE-I	12,323
6	REDOG-3	13,987
7	BIKE-III	24,659
8	REDOG-5	32,634
9	BIKE-V	40,973
10	mceliece348864	261,120
11	PALOMA128	319,488
12	mceliece460896	524,160
13	PALOMA192	812,032
14	PALOMA256	1,025,024
15	mceliece6688128	1,044,992
16	mceliece6960119	1,047,319
17	mceliece8192128	1,357,824

Rank	Scheme	Secret key
1	REDOG-1	666
2	REDOG-2	1,464
3	BIKE-I	2,244
4	HQC-128	2,289
5	REDOG-3	2,560
6	BIKE-III	3,346
7	HQC-192	4,562
8	BIKE-V	4,640
9	mceliece348864	6,492
10	HQC-256	7,285
11	mceliece460896	13,608
12	mceliece6688128	13,932
13	mceliece6960119	13,948
14	mceliece8192128	14,120
15	PALOMA128	94,528
16	PALOMA192	357,568
17	PALOMA256	359,616

Rank	Scheme	Ciphertext
1	mceliece348864	96
2	PALOMA128	136
3	mceliece460896	156
4	mceliece6688128	194
5	mceliece6960119	208
	mceliece8192128	208
7	PALOMA192	240
	PALOMA256	240
9	REDOG-1	389
10	REDOG-2	840
11	REDOG-3	1,475
12	HQC-128	4,497
13	HQC-192	9,042
14	BIKE-I	12,579
15	HQC-256	14,485
16	BIKE-III	24,915
17	BIKE-V	41,229

공개키 관점에서는 HQC가 월등히 작음
암호문 관점에서는 mceliece와 PALOMA가 유사하게 작음

Dililthium FALCON vs HAETAE/NCC 키 크기 비교

Rank	Scheme	Public key
1	FALCON-512	897
2	HAETAE2	992
3	Dilithium-II	1,312
4	HAETAE3	1472
5	NCC-Sign1	1,760
6	FALCON-1024	1,793
7	Dilithium-III	1,952
8	HAETAE5	2080
9	NCC-Sign3	2,336
10	Dilithium-V	2,592
11	NCC-Sign5	3,200

Rank	Scheme	Secret key
1	FALCON-512	1,281
2	HAETAE2	1,408
3	HAETAE3	2,112
4	FALCON-1024	2,305
5	Dilithium-II	2,528
6	NCC-Sign1	2,688
7	HAETAE5	2,752
8	NCC-Sign3	3,552
9	Dilithium-III	4,,000
10	Dilithium-V	4,864
11	NCC-Sign5	5,568

Rank	Scheme	Signature
1	FALCON-512	666
2	FALCON-1024	1,280
3	HAETAE2	1,474
4	HAETAE3	2,349
5	Dilithium-II	2,420
6	NCC-Sign1	2,912
7	HAETAE5	2,948
8	Dilithium-III	3,293
9	NCC-Sign3	3,872
10	Dilithium-V	4,595
11	NCC-Sign5	6,080

FALCON이 전반적으로 작은 키/서명 크기를 가짐

SPHINCS vs AIMer 키 크기 비교

Rank	Scheme	Public key
1	AIMer128f	32
	AIMer128s	32
	SPHINCS+128	32
4	AIMer192f	48
	AIMer192s	48
	SPHINCS+192	48
7	SPHINCS+256	49
8	AIMer256f	64
	AIMer256s	64

Rank	Scheme	Secret key
1	AIMer128f	48
	AIMer128s	48
3	SPHINCS+128	64
4	AIMer192f	72
	AIMer192s	72
6	AIMer256f	96
	AIMer256s	96
8	SPHINCS+192	96
9	SPHINCS+256	128

Rank	Scheme	Signature
1	AIMer128s	4,160
2	AIMer128f	5,888
3	SPHINCS+128s	7,856
4	AIMer192s	9,120
5	AIMer192f	13,056
6	SPHINCS+192s	16,224
7	AIMer256s	17,056
8	SPHINCS+128f	17,088
9	AIMer256f	25,120
10	SPHINCS+256s	29,792
11	SPHINCS+192f	35,664
12	SPHINCS+256f	49,856

SPHINCS 대비 AIMer가 모든 관점에서 키/서명 크기가 작음

Kyber vs new SMAUG-T, new NTRU+ 성능 비교(Intel)

Rank	Scheme	Keypair(avg)
1	SMAUG-Timer	79,115
2	SMAUG-T1	84,395
3	Kyber512	110,653
4	SMAUG-T3	160,405
5	NTRU+KEM576	166,616
6	NTRU+PKE576	167,487
7	NTRU+KEM768	185,971
8	NTRU+PKE768	186,231
9	Kyber768	198,651
10	SMAUG-T5	232,659
11	NTRU+KEM864	232,748
12	NTRU+PKE864	239,893
13	kyber1024	266,290
14	NTRU+KEM1152	312,277
15	NTRU+PKE1152	336,309

Rank	Scheme	encap(avg)
1	SMAUG-T1	57,806
2	SMAUG-Timer	58,033
3	NTRU+PKE576	81,018
4	NTRU+KEM576	82,890
5	NTRU+PKE768	105,425
6	NTRU+KEM768	116,915
7	Kyber512	125,375
8	NTRU+PKE864	127,953
9	SMAUG-T3	130,266
10	NTRU+KEM864	135,389
11	NTRU+KEM1152	142,665
12	NTRU+PKE1152	151,924
13	Kyber768	174,804
14	SMAUG-T5	202,564
15	Kyber1024	244,419

Rank	Scheme	decap(avg)
1	NTRU+KEM576	70,074
2	NTRU+PKE576	75,194
3	SMAUG-T1	75,928
4	SMAUG-Timer	77,452
5	NTRU+KEM768	104,460
6	NTRU+PKE768	105,068
7	NTRU+PKE864	132,479
8	NTRU+KEM864	132,608
9	NTRU+KEM1152	142,255
10	Kyber512	143,529
11	NTRU+PKE1152	151,553
12	SMAUG-T3	153,594
13	Kyber768	210,180
14	SMAUG-T5	236,725
15	Kyber1024	290,399

Kyber보다 KpqC 알고리즘들이 키생성, 캡슐화, 디캡슐화에서 1.3, 2.1, 그리고 2.0배 성능 우수

McEliece HQC vs new PALOMA 성능 비교(Intel)

Rank	Scheme	Keygen(avg)
1	hqc-128	4,308,666
2	hqc-192	12,098,383
3	hqc-256	22,078,541
4	**PALOMA128**	65,025,193
5	mceliece348864f	115,143,829
6	mceliece348864	257,322,142
7	**PALOMA192**	308,497,763
8	**PALOMA256**	328,543,370
9	mceliece460896f	353,874,031
10	mceliece6960119f	568,091,516
11	mceliece8192128f	617,997,240
12	mceliece6688128f	653,450,392
13	mceliece460896	804,708,960
14	mceliece6960119	1,640,855,766
15	mceliece6688128	1,886,622,666
16	mceliece8192128	1,951,587,331

Rank	Scheme	encap(avg)
1	**PALOMA128**	107,373
2	**PALOMA256**	161,877
3	**PALOMA192**	170,696
4	mceliece348864f	171,826
5	mceliece348864	178,119
6	mceliece460896f	355,743
7	mceliece460896	357,513
8	mceliece8192128f	465,714
9	mceliece8192128	503,102
10	mceliece6688128	563,525
11	mceliece6688128f	587,370
12	mceliece6960119	1,828,378
13	mceliece6960119f	1,847,889
14	hqc-128	8,551,947
15	hqc-192	24,193,559
16	hqc-256	44,083,667

Rank	Scheme	p(avg)
1	**PALOMA128**	2,868,445
2	**PALOMA192**	9,761,363
3	**PALOMA256**	11,290,786
4	hqc-128	12,077,461
5	hqc-192	36,473,745
6	mceliece348864f	38,731,109
7	mceliece348864	38,777,040
8	hqc-256	66,741,738
9	mceliece460896	93,483,536
10	mceliece460896f	93,624,278
11	mceliece6960119f	174,013,196
12	mceliece6960119	174,166,372
13	mceliece6688128	179,210,748
14	mceliece6688128f	180,083,946
15	mceliece8192128f	219,454,147
16	mceliece8192128	219,602,742

키생성에서는 HQC가 월등히 높은 성능을 나타냄
캡슐화와 디캡슐화에서는 PALOMA가 월등히 높은 성능을 나타냄

Dilithium FALCON vs new HAETAE, new NCC 성능 비교(Intel)

Rank	Scheme	Keypair(avg)
1	NCC-Sign5-Shake_NTT_MODE_1	227,073
2	NCC-Sign1-Shake_NTT_MODE_1	228,461
3	NCC-Sign3-Shake_NTT_MODE_1	248,992
4	NCC-Sign5-Shake_NTT_MODE_3	277,435
5	dilithium2	278,954
6	NCC-Sign3-Shake_NTT_MODE_3	279,191
7	NCC-Sign1-Shake_NTT_MODE_3	286,621
8	NCC-Sign5-Shake_NTT_MODE_5	401,949
9	NCC-Sign3-Shake_NTT_MODE_5	406,242
10	NCC-Sign1-Shake_NTT_MODE_5	421,879
11	NCC-Sign1-AES_NTT_MODE_1	436,714
12	NCC-Sign5-AES_NTT_MODE_3	454,915
13	dilithium3	482,104
14	NCC-Sign3-AES_NTT_MODE_1	541,555
15	NCC-Sign3-AES_NTT_MODE_3	562,878
16	NCC-Sign1-AES_NTT_MODE_3	598,639
17	dilithium5	734,609
18	NCC-Sign5-AES_NTT_MODE_1	768,881
19	NCC-Sign3-AES_NTT_MODE_5	772,446
20	NCC-Sign5-AES_NTT_MODE_5	773,926
21	NCC-Sign1-AES_NTT_MODE_5	780,292
22	HAETAE2	1,343,166
23	HAETAE3	1,777,115
24	HAETAE5	1,983,237
25	falcon-padded-512	37,897,367
26	falcon-512	37,959,474
27	f[illegible]	[illegible]
28	falcon[illegible]	[illegible]

Rank	Scheme	Sign(avg)
1	NCC-Sign1-Shake_NTT_MODE_1	436,747
2	dilithium2	450,483
3	NCC-Sign3-Shake_NTT_MODE_1	439,620
4	NCC-Sign1-Shake_NTT_MODE_3	521,188
5	NCC-Sign5-Shake_NTT_MODE_1	616,361
6	NCC-Sign5-Shake_NTT_MODE_3	775,875
7	NCC-Sign3-AES_NTT_MODE_1	835,293
8	NCC-Sign3-Shake_NTT_MODE_5	836,839
9	NCC-Sign5-Shake_NTT_MODE_5	848,191
10	NCC-Sign1-Shake_NTT_MODE_5	852,461
11	NCC-Sign3-Shake_NTT_MODE_3	1,129,373
12	NCC-Sign5-AES_NTT_MODE_1	1,258,524
13	NCC-Sign5-AES_NTT_MODE_5	1,275,414
14	dilithium5	1,463,427
15	HAETAE5	1,564,403
16	NCC-Sign1-AES_NTT_MODE_1	1,613,659
17	NCC-Sign1-AES_NTT_MODE_3	1,693,349
18	NCC-Sign5-AES_NTT_MODE_3	2,304,319
19	NCC-Sign3-AES_MODE_5	2,362,183
20	NCC-Sign3-AES_MODE_3	2,414,125
21	dilithium3	2,465,069
22	NCC-Sign1-AES_MODE_5	2,783,767
23	HAETAE3	3,954,599
24	HAETAE2	4,273,817
25	falcon-512	10,617,704

Rank	Scheme	verify(avg)
1	falcon-512	132,067
2	falcon-padded-512	133,669
3	HAETAE2	204,595
4	NCC-Sign5-ShakeNTT_MODE_1	240,970
5	NCC-Sign1-Shake_NTT_MODE_1	246,625
6	NCC-Sign3-Shake_NTT_MODE_1	257,882
7	falcon-padded-1024	281,205
8	falcon-1024	281,777
9	dilithium2	281,907
10	NCC-Sign1-Shake_NTT_MODE_3	311,455
11	NCC-Sign5-Shake_NTT_MODE_3	311,750
12	NCC-Sign3-Shake_NTT_MODE_3	319,218
13	HAETAE3	333,241
14	NCC-Sign1-AE_NTT_MODE_1	372,305
15	HAETAE5	431,209
16	dilithium3	440,805
17	NCC-Sign5-Shake_NTT_MODE_5	467,284
18	NCC-Sign3-Shake_NTT_MODE_5	469,470
19	NCC-Sign3-AES_NTT_MODE_1	507,244
20	NCC-Sign3-AES_NTT_MODE_3	516,100
21	NCC-Sign1-AES_NTT_MODE_3	521,632
22	NCC-Sign1-Shake_NTT_MODE_5	541,151
23	NCC-Sign5-AES_NTT_MODE_3	547,977
24	dilithium5	737,849
25	NCC-Sign1-AESI_NTT_MODE_5	743,799
[illegible]	NCC-Sign5-AES_NTT_MODE_5	763,579
[illegible]	[illegible]	811,063
[illegible]	[illegible]	814,684

키생성과 서명에서는 유사한 성능을 나타냄

검증에서는 FALCON이 1.5배 나은 성능을 나타냄 (다만 FALCON은 키생성과 서명이 매우 느림)

SPHINCS vs AIMer 성능 비교(Intel)

Rank	Scheme	Keypair(avg)
1	AIMer128s	85,760
2	AIMer128f	86,858
3	AIMer192s	221,051
4	AIMer192f	221,939
5	AIMer256f	532,954
6	AIMer256s	536,662
7	sphincs-sha2-128f-simple	3,512,168
8	sphincs-sha2-192f-simple	5,106,785
9	sphincs-shake-128f-simple	5,676,001
10	sphincs-shake-192f-simple	8,233,255
11	sphincs-sha2-256f-simple	13,540,513
12	sphincs-shake-256f-simple	21,778,482
13	sphincs-sha2-256s-simple	213,627,804
14	sphincs-sha2-128s-simple	222,113,099
15	sphincs-sha2-192s-simple	323,514,039
16	sphincs-shake-256s-simple	346,536,078
17	sphincs-shake-128s-simple	3
18	sphincs-shake-192s-simple	524,022,262

Rank	Scheme	sign(avg)
1	AIMer128f	7,020,534
2	AIMer192f	13,314,965
3	AIMer256f	34,079,571
4	AIMer128s	54,057,979
5	sphincs-sha2-128f-simple	81,191,592
6	AIMer192s	103,955,683
7	sphincs-shake-128f-simple	131,239,147
8	sphincs-sha2-192f-simple	133,555,092
9	sphincs-shake-192f-simple	212,061,614
10	AIMer256s	263,703,008
11	sphincs-sha2-256f-simple	272,214,600
12	sphincs-shake-256f-simple	440,803,080
13	sphincs-sha2-128s-simple	1,690,989,832
14	sphincs-sha2-256s-simple	2,630,577,802
15	sphincs-shake-128s-simple	2,725,259,275
16	sphincs-sha2-192s-simple	2,989,370,882

Rank	Scheme	verify(avg)
1	sphincs-sha2-128s-simple	1,769,297
2	sphincs-sha2-192s-simple	2,394,749
3	sphincs-shake-128s-simple	2,763,529
4	sphincs-sha2-256s-simple	3,554,691
5	sphincs-shake-192s-simple	3,872,933
6	sphincs-sha2-128f-simple	4,989,027
7	sphincs-shake-256s-simple	5,538,012
8	AIMer128f	6,429,942
9	sphincs-sha2-256f-simple	7,081,467
10	sphincs-sha2-192f-simple	7,123,456
11	sphincs-shake-128f-simple	7,812,153
12	sphincs-shake-192f-simple	11,374,784
13	sphincs-shake-256f-simple	11,778,032
14	AIMer192f	13,025,861
15	AIMer256f	31,939,622
16	AIMer128s	53,876,227
17	AIMer192s	102,567,956
18	AIMer256s	257,224,646

키생성과 서명에서는 AIMER가 40배와 11배의 성능 향상을 도출
검증에서는 SPHINCS가 3.6배 나은 성능 도출

결론

- **다양한 기반 문제들에 대한 성능 지표를 CLEAN/AVX2 코드를 통해 분석**
  - 다만 기반 문제들의 고유 특성으로 인해 넘어설 수 없는 성능 차이 발생 **(격자의 강인함)**
    - **SPHINCS+의 경우 long-term security를 만족 (archival system과 root certificates에 적합)**
  - 따라서 동일한 기반 문제 간의 비교가 보다 공평할 것으로 판단 **(NIST vs KpqC)**
  - NIST의 표준화 선택과 유사한 방향성? **(First Lattice, Second Code, Last Hash)**

- **NIST PQC와 성능 비교 시 KpqC가 우위를 가지는 부분 존재**
  - KpqC 알고리즘을 보다 고도화 및 **국제 표준화할 경우 충분한 경쟁력**을 가지고 활용 가능
    → 다만 **NIST PQC에서도 Kyber & Dilithium이 많이 활용될 가능성 (Falcon은 낮은 관심)**
  - 산업체에서 이를 수용하는 것은 또 다른 문제

- **KpqC 선정 결과는 하나가 아닌 다수로 결정 필요**
  - 다양한 서비스 환경에서 다양한 사용자 니즈를 만족시킬 수 있는 알고리즘 필요
  - 특정 기반문제 해킹 시 이를 보완할 수 있어야 함 (e.g., SIKE)
  - **다수 선정 시 구현 생태계 고려 필요 (최소한의 자원으로 모든 알고리즘/프로토콜 커버 필요)**

Q & A