Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems 논문 리뷰

https://youtu.be/50Lz8DmpwCA

서론

- TLS는 PQC(post-quantum cryptography)를 지원하도록 변형되어야 함
  - **하지만, 아직 post-quantum TLS는 표준화되지 않았음.**
  - **특히, 리소스 제한이 있는 IoT 기기에서 post-quantum TLS의 전체적인 성능은 많이 알려져 있지 않음.**
- 따라서, 본 논문은 **TLS1.3 아키텍쳐 수정하여 TLS 1.3을 quantum-safe 하게 마이그레이션 하는 방법 소개**
- NIST PQC 프로세서에서 표준화를 위해

→ 대부분의 post-quantum 알고리즘의 pqm4와 PQClean library implementation과 Round 4 후보군 알고리즘을 통합

→제안된 솔루션과 성능 평가는 **ARM Cortex-M4**에서 측정

  - 실행 시간, 메모리 사용량, 네트워크 트래픽에 대한 체계적이고 철저한 평가 수행

- 저자가 아는 한, **최초로 임베디드 상에서 NIST PQC 프로세스 알고리즘에 대해 PQ TLS 1.3의 실행시간, 메모리 사용량, 네트워크 트래픽 성능 측정(2022년 기준)**

제안 기법

- WolfSSL은 Mbed TLS와 함께 임베디드 시스템용 오픈소스 TLS 솔루션에서 가장 유명하고 널리 사용
- **최신 버전에서는 Liboqs와 통합하여 CRYSTALS-Kyber, SABER, NTRU, Falcon 지원**
  - 하지만, 2021년 기준 pqm4의 Kyber512만 사용하는 실험 설정을 제외하고 Cortex-M4용 최적화 코드가 포함되어 있지 않음

- **본 논문은, ARM Cortex-M4상에서 최적화된 PQC 알고리즘을 모두 지원하기 위해 WolfSSL 라이브러리 수정**

제안 기법

- WolfSSL의 주요 구성요소
  - **WolfCrypt**
    - 모든 암호 알고리즘, 인증서, 키 파일을 처리하는 프로그램 포함
    - 다양한 아키텍처에 **최적화 코드**와 **선택한 플랫폼에 대한 하드웨어** 지원을 제공
  - **WolfSSL**
    - DTLS(Datagram Transport Layer Security)와 같은 **다른 프로토콜을 구현하는 모든 프로토콜 관련 코드 포함**
    - **TLS의 모든 설정과 환경**, TCP와 같은 **하위 수준 프로토콜**이나 운영체제(e.g. FreeRTOS)와 같은 **상위 수준과 통신하기 위한 인터페이스 제공**
  - **유틸리티**
    - 테스트용 벤치마크나 프로그램과 같은 non-essential 유틸리티 포함
  - **본 논문에서는 성능 측정을 위해 2개의 벤치마크 프로그램 사용**
    - ***Wolfcrypt-benchmark*** **: 모든 암호 알고리즘의 성능 측정과 통계 제공**

    → **기존 알고리즘과 비교하기 위해 PQC 알고리즘의 시간 측정에 사용**
    - ***Wolfssl-tls-bench :*** **TLS 세션과 관련한 metrics 측정**

    → **기존 알고리즘을 사용하는 TLS 성능과 PQC 알고리즘을 사용하는 TLS 성능 측정에 사용**

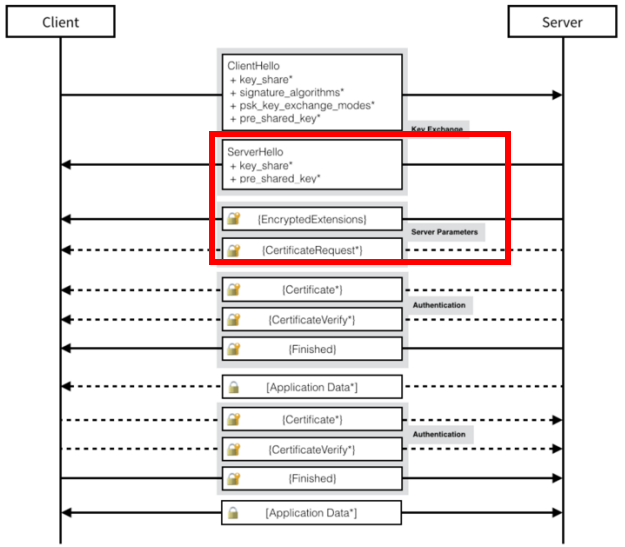
제안 기법

- **WolfSSL 라이브러리와 TLS 1.3 표준 수정**
  - **2개의 TLS Extension fields 수정**

  → **"Support Group", "Signature Algorithms"**

```
struct {
  ExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
} Extension;

enum {
server_name(0),                             /* RFC 6066 */
max_fragment_length(1),                     /* RFC 6066 */
status_request(5),                          /* RFC 6066 */
supported_groups(10),                       /* RFC 8422, 7919 */
signature_algorithms(13),                   /* this document */
use_srtp(14),                               /* RFC 5764 */
heartbeat(15),                              /* RFC 6520 */
application_layer_protocol_negotiation(16), /* RFC 7301 */
signed_certificate_timestamp(18),           /* RFC 6962 */
client_certificate_type(19),                /* RFC 7250 */
server_certificate_type(20),                /* RFC 7250 */
padding(21),                                /* RFC 7685 */
pre_shared_key(41),                         /* this document */
early_data(42),                             /* this document */
supported_versions(43),                     /* this document */
cookie(44),                                 /* this document */
psk_key_exchange_modes(45),                 /* this document */
certificate_authorities(47),                /* this document */
oid_filters(48),                            /* this document */
post_handshake_auth(49),                    /* this document */
signature_algorithms_cert(50),              /* this document */
key_share(51),                              /* this document */
(65535)
} ExtensionType;
```



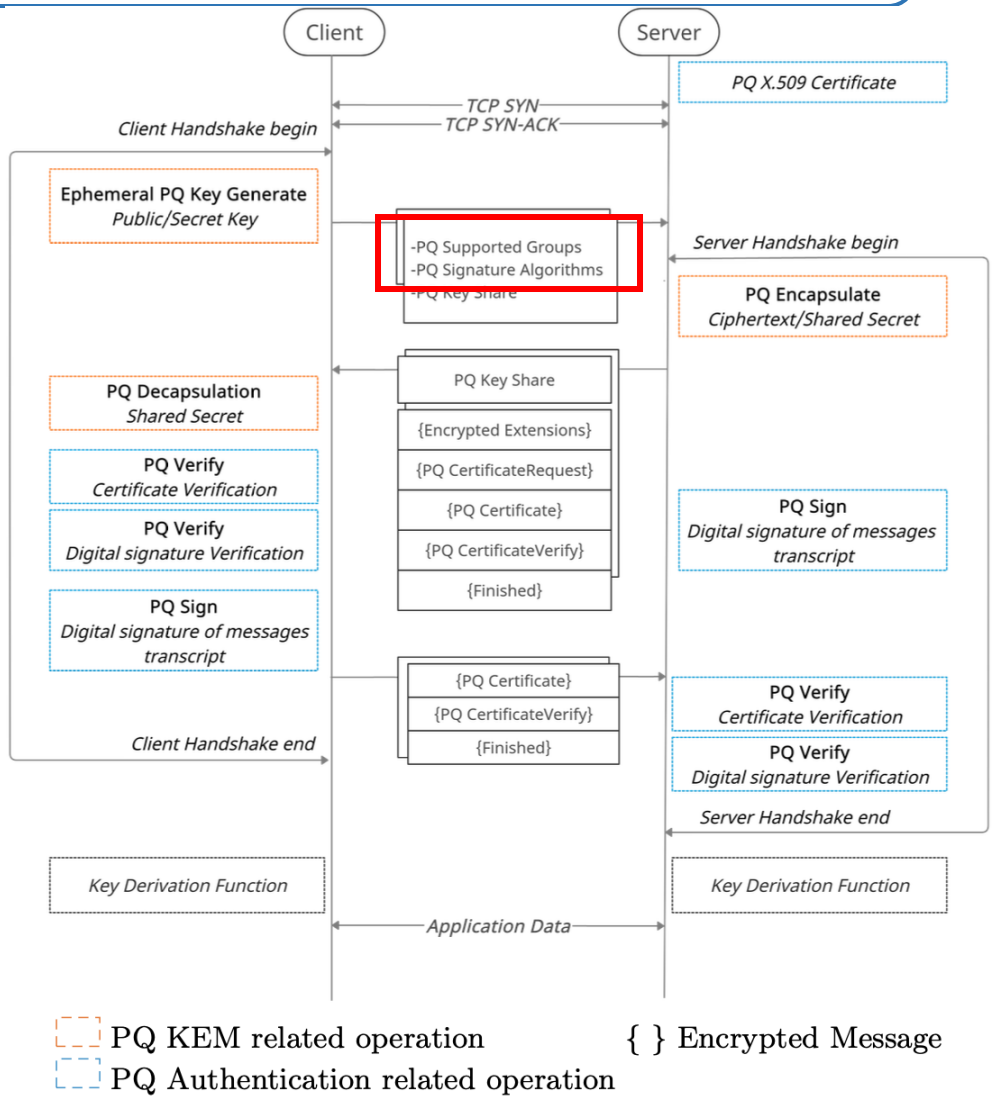


Fig. 1. Post-quantum TLS 1.3 handshake messages.

https://www.davidwong.fr/tls13/#section-4.2

제안 기법

- **Support Group**
  - 키교환 알고리즘을 선택할 수 있도록 선호 순서대로 인코딩 된 식별자인 코드포인트 목록 전송

  * 코드포인트는 "Named Groups"로 불리고, 프로토콜 자체에서 각 지원되는 알고리즘에 대해 정의됨.

→ 다른 라이브러리와의 상호운용성을 위해 **OQS의 OpenSSL fork와 동일한 codepoints 사용**

```
enum {
  /* Elliptic Curve Groups (ECDHE) */
  secp256r1(0x0017), secp384r1(0x0018), secp521r1(0x0019),
  x25519(0x001D), x448(0x001E),

  /* Finite Field Groups (DHE) */
  ffdhe2048(0x0100), ffdhe3072(0x0101), ffdhe4096(0x0102),
  ffdhe6144(0x0103), ffdhe8192(0x0104),

  /* Reserved Code Points */
  ffdhe_private_use(0x01FC..0x01FF),
  ecdhe_private_use(0xFE00..0xFEFF),
  (0xFFFF)
} NamedGroup;

struct {
  NamedGroup named_group_list<2..2^16-1>;
} NamedGroupList;
```

bikel1	0x0241
p256_bikel1	0x2F41
x25519_bikel1	0x2FAE
bikel3	0x0242
p384_bikel3	0x2F42
x448_bikel3	0x2FAF
bikel5	0x0243

open-quantum-safe / oqs-provider Public

Code | Issues 41 | Pull requests 4 | Discussions | Actio

main | oqs-provider / ALGORITHMS.md

baentsch update MLKEM code points (#511)

285 lines (271 loc) · 17.4 KB

Preview | Code | Blame | Raw

Algorithms supported

This page lists all quantum-safe algorithms supported by oqs-provider.

Some algorithms by default may not be enabled for use in the master code-generator template file.

As standardization for these algorithms within TLS is not done, all TLS code points/IDs can be changed from their default values to values set by environment variables. This facilitates interoperability testing with TLS1.3 implementations that use different IDs.

Code points / algorithm IDs

Algorithm name	default ID	enabled	environment variable
frodo640aes	0x0200	Yes	OQS_CODEPOINT_FRODO640AES
p256_frodo640aes	0x2F00	Yes	OQS_CODEPOINT_P256_FRODO640AES
x25519_frodo640aes	0x2F80	Yes	OQS_CODEPOINT_X25519_FRODO640AES
frodo640shake	0x0201	Yes	OQS_CODEPOINT_FRODO640SHAKE

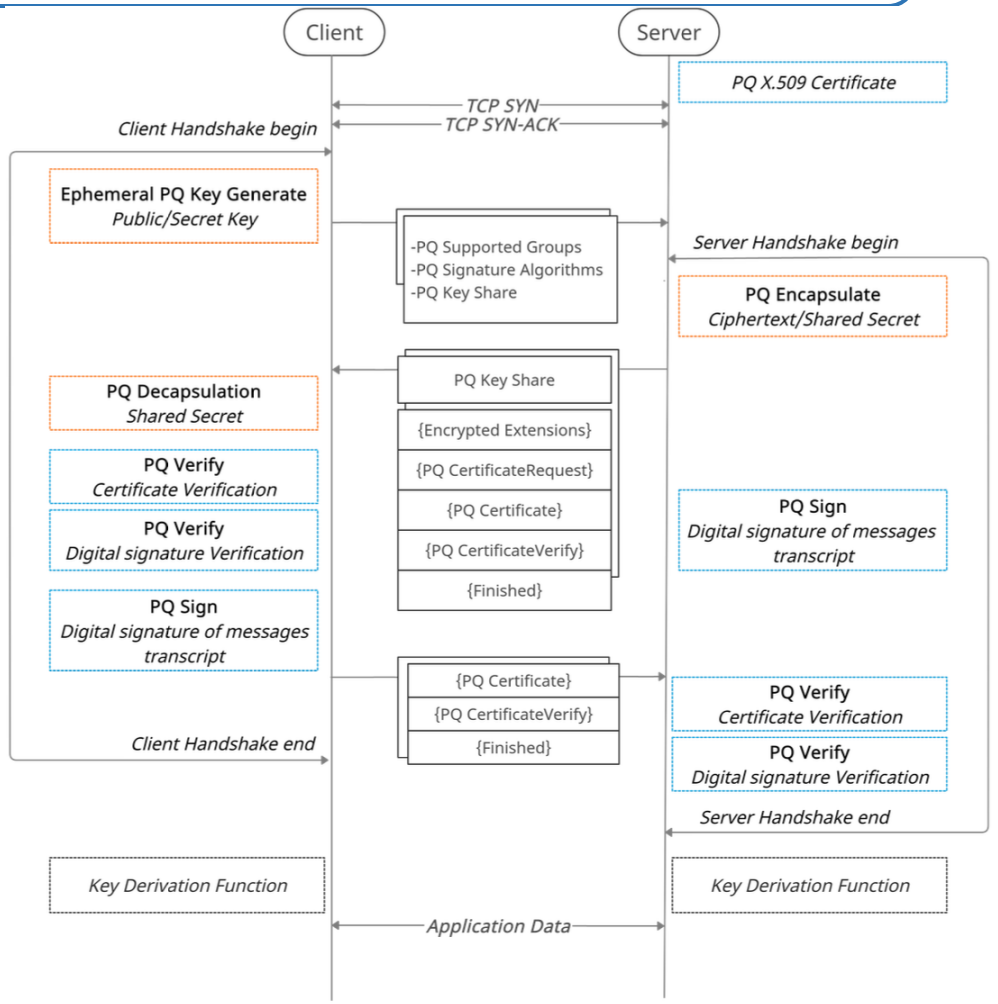


Fig. 1. Post-quantum TLS 1.3 handshake messages.

제안 기법

- **Signature Algorithms**
  - 지원하는 서명 알고리즘에 대한 선호도를 제공

→ **PQ TLS1.3 설계의 quantum 전자서명 알고리즘에 대한 새로운 codepoints 도입**

→ **OQS의 OpenSSL fork와 동일한 codepoints 사용**

```
enum {
  /* RSASSA-PKCS1-v1_5 algorithms */
  rsa_pkcs1_sha256(0x0401),
  rsa_pkcs1_sha384(0x0501),
  rsa_pkcs1_sha512(0x0601),

  /* ECDSA algorithms */
  ecdsa_secp256r1_sha256(0x0403),
  ecdsa_secp384r1_sha384(0x0503),
  ecdsa_secp521r1_sha512(0x0603),

  /* RSASSA-PSS algorithms with public key OID rsaEncryption */
  rsa_pss_rsae_sha256(0x0804),
  rsa_pss_rsae_sha384(0x0805),
  rsa_pss_rsae_sha512(0x0806),

  /* EdDSA algorithms */
  ed25519(0x0807),
  ed448(0x0808),

  /* RSASSA-PSS algorithms with public key OID RSASSA-PSS */
  rsa_pss_pss_sha256(0x0809),
  rsa_pss_pss_sha384(0x080a),
  rsa_pss_pss_sha512(0x080b),

  /* Legacy algorithms */
  rsa_pkcs1_sha1(0x0201),
  ecdsa_sha1(0x0203),

  /* Reserved Code Points */
  private_use(0xFE00..0xFFFF),
  (0xFFFF)
} SignatureScheme;

struct {
  SignatureScheme supported_signature_algorithms<2..2^16-2>;
} SignatureSchemeList;
```

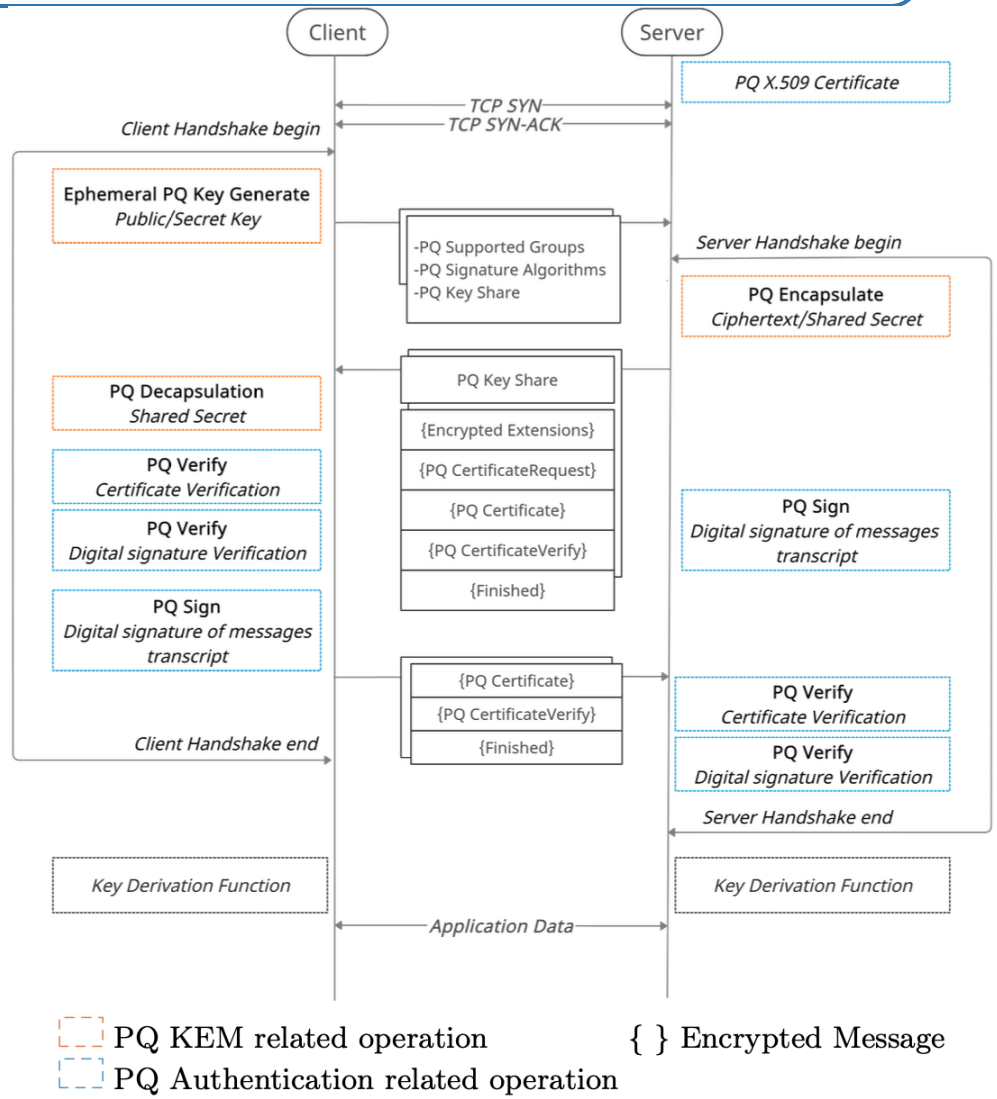


Fig. 1. Post-quantum TLS 1.3 handshake messages.

제안 기법

- **WolfSSL 라이브러리와 TLS 1.3 표준 수정**
  - **Key Encapsulation Mechanism Support/Adaption**
    - TLS 1.3 표준에서는 타원곡선 Diffie-Hellman 키 교환만 지원

    → **따라서, 키 교환 메커니즘을 KEM 스킴으로 변경**
      - **CRYSTALS-Kyber KEM 기반 키 교환 체계에서 제안된 방식 채택**
  - **Digital Certificates Support**
    - **OQS의 OpenSSL API를 통해 본 논문에서 평가되는 PQC 알고리즘을 지원하는 디지털 인증서 생성**
    - 서버와 클라이언트에 대해 상호 인증되는 디지털 인증서 생성 목표

    → 따라서, **PQ CA(Certificate Authority) 도입하여 PQ CA에서 확인 가능한 서버용과 클라이언트용 인증서 생성**
    - 단순화를 위해 **체인의 모든 인증서는 동일한 서명 알고리즘 (Dilithium2 등) 사용**

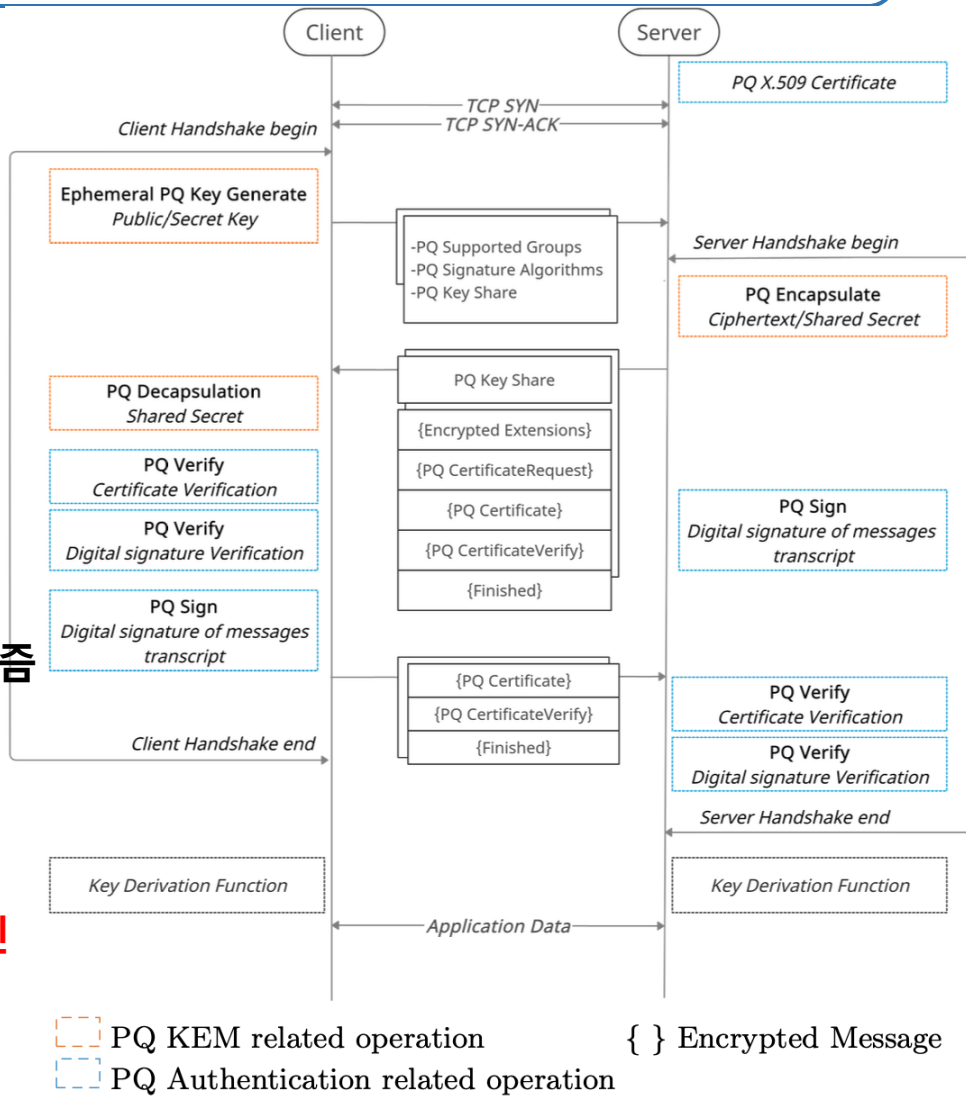


Fig. 1. Post-quantum TLS 1.3 handshake messages.

제안 기법

- **NIST PQC Round 4 알고리즘과 Round3의 일부 알고리즘 채택**
  - KEM : **CRYSTALS-Kyber, SABER, NTRU, SIKE, BIKE, HQC, NTRU LPRime, FrodoKEM**
  - 전자서명 : **CRYSTALS-Dilithium, Falcon, SPHINCS+, Picnic3**
- **pqm4 구현을 WolfSSL 코드에 통합**
- **네트워크 통신을 위해 TCP/IP 프로토콜 제품군의 경량 구현인 lwIP(lightweight IP) 라이브러리 활용**

- 실험 장치
  - 임베디드 장치: **Nucleo-F439ZI 보드**
    - 32비트 ARM Cortex-M4, 180 MHz, 192 KB SRAM, 2MB 플래시 메모리
  - 클라우드 서버 역할 장치 : **Ubuntu 20.04**, x86_64 아키텍처, **Intel i7-1165G7 processor**, 2.8 GHz
- 실험 장치는 RTT이 0.493ms인 Ethernet interface를 통해 동일한 액세스 포인트에 연결

RRT : Round-Trip Time

각 알고리즘의 공개 키 크기 및 암호문/서명, 측정된 실행 시간

- **Kyber가 가장 높은 종합적 성능을 보임**

Table 1. Summary of Traditional and Post-quantum Primitives.

KEM Algorithm	*NIST level*	*Public Key (bytes)*	*Ciphertext (bytes)*	*Key Generate (ms)*	*Encapsulate (ms)*	*Decapsulate (ms)*	*Notation*
FFDHE [1]	0	256	256	203.920	204.080 [4]	-	*FFDHE*
ECDHE [2]	0	32	32	8.428	17.687 [4]	-	*ECDHE*
Kyber512	1	800	768	8.133	6.239	3.419	*Kyb1*
BIKE-L1	1	1541	1573	200.620	25.969	411.308	*Bike1*
HQC128	1	2249	4481	30.202	50.682	72.775	*Hqc1*
SIKEp434	1	330	346	309.235	498.227	530.200	*Sike1*
SIKEp503	2	378	402	423.708	690.188	733.071	*Sike2*
SIKEp610	3	462	486	732.786	1341.125	1346.500	*Sike3*
Kyber768	3	1184	1088	12.224	11.412	7.924	*Kyb3*
SIKEp751	5	564	596	1262.750	2036.000	2183.000	*Sike5*
Kyber1024	5	1568	1568	12.918	11.623	8.539	*Kyb5*
Auth. Algorithm	*NIST level*	*Public Key (bytes)*	*Signature (bytes)*	*Key Generate (ms)*	*Sign (ms)*	*Verify (ms)*	*Notation*
RSA [3]	0	256	256	12.853/450 [5]	448.250	12.500	*RSA*
ECDSA [2]	0	32	32	8.428	12.305	25.193	*ECDSA*
Sphincs128s	1	32	7856	8674.000	66 239.000	61.588	*Sphi1s*
Sphincs128f	1	32	17088	137.750	3361.000	190.167	*Sphi1f*
Falcon512	1	897	666	1266.667	243.881	3.275	*Falc1*
Dilithium2	2	1312	2420	12.063	25.404 [6]	9.569	*Dil2*
Dilithium3	3	1952	3293	19.438	39.309 [7]	16.244	*Dil3*
Falcon1024	5	1793	1280	4802.667	527.789	6.852	*Falc5*

[1] 3072-bit, [2] secp256r1 curve, [3] 2048-bit, [4] key agreement time, [5] public / private key generation time, [6] (11/114) (min/max), [7] (15/138) (min/max) of execution time over 1000 signatures

각 알고리즘의 공개 키 크기 및 암호문/서명, 측정된 실행 시간

- **Dilithium이 가장 균형 잡힌 성능 제공**
  - 공개키와 서명의 크기가 중간 크기에 해당되지만, **실행 시간 측면에서 뛰어난 성능 제공**

Table 1. Summary of Traditional and Post-quantum Primitives.

KEM Algorithm	*NIST level*	*Public Key (bytes)*	*Ciphertext (bytes)*	*Key Generate (ms)*	*Encapsulate (ms)*	*Decapsulate (ms)*	*Notation*
FFDHE [1]	0	256	256	203.920	204.080 [4]	-	*FFDHE*
ECDHE [2]	0	32	32	8.428	17.687 [4]	-	*ECDHE*
Kyber512	1	800	768	8.133	6.239	3.419	*Kyb1*
BIKE-L1	1	1541	1573	200.620	25.969	411.308	*Bike1*
HQC128	1	2249	4481	30.202	50.682	72.775	*Hqc1*
SIKEp434	1	330	346	309.235	498.227	530.200	*Sike1*
SIKEp503	2	378	402	423.708	690.188	733.071	*Sike2*
SIKEp610	3	462	486	732.786	1341.125	1346.500	*Sike3*
Kyber768	3	1184	1088	12.224	11.412	7.924	*Kyb3*
SIKEp751	5	564	596	1262.750	2036.000	2183.000	*Sike5*
Kyber1024	5	1568	1568	12.918	11.623	8.539	*Kyb5*
Auth. Algorithm	***NIST level***	***Public Key (bytes)***	***Signature (bytes)***	***Key Generate (ms)***	***Sign (ms)***	***Verify (ms)***	***Notation***
RSA [3]	0	256	256	12.853/450 [5]	448.250	12.500	*RSA*
ECDSA [2]	0	32	32	8.428	12.305	25.193	*ECDSA*
Sphincs128s	1	32	7856	8674.000	66 239.000	61.588	*Sphi1s*
Sphincs128f	1	32	17088	137.750	3361.000	190.167	*Sphi1f*
Falcon512	1	897	666	1266.667	243.881	3.275	*Falc1*
Dilithium2	2	1312	2420	12.063	25.404 [6]	9.569	*Dil2*
Dilithium3	3	1952	3293	19.438	39.309 [7]	16.244	*Dil3*
Falcon1024	5	1793	1280	4802.667	527.789	6.852	*Falc5*

[1] 3072-bit, [2] secp256r1 curve, [3] 2048-bit, [4] key agreement time, [5] public / private key generation time, [6] (11/114) (min/max), [7] (15/138) (min/max) of execution time over 1000 signatures

각 알고리즘의 공개 키 크기 및 암호문/서명, 측정된 실행 시간

- 서버 전용 인증을 사용하는 시나리오(임베디드 보드가 클라이언트 역할)

→ **Falcon을 사용할 때, 실행 시간 크게 향상**

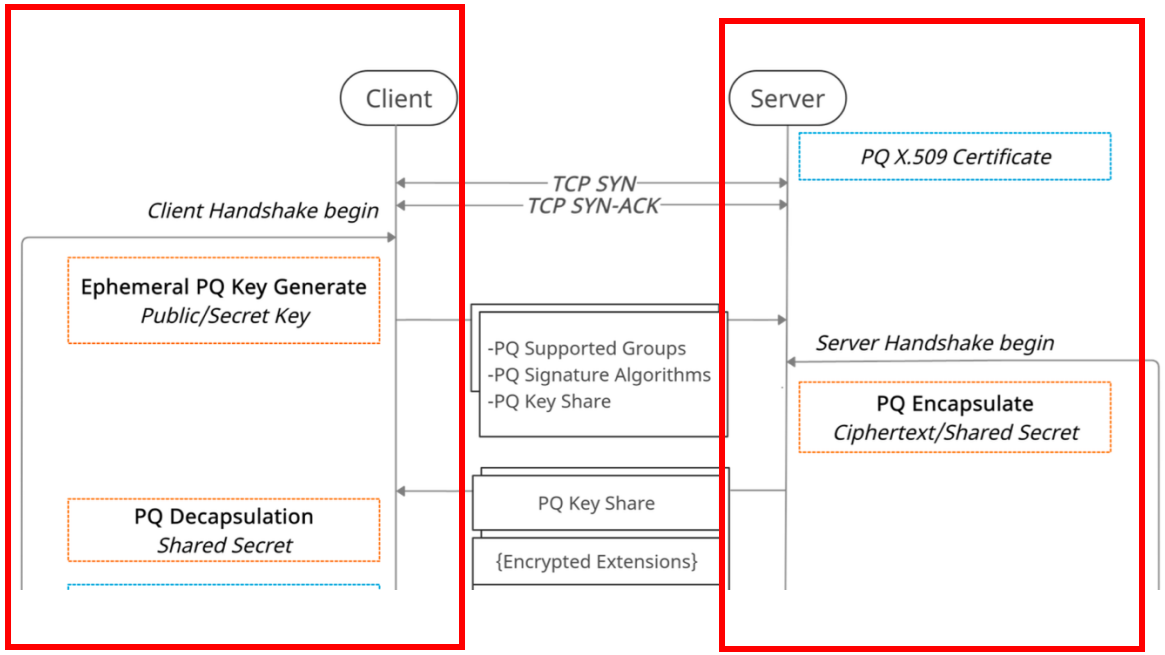
Table 1. Summary of Traditional and Post-quantum Primitives.

KEM Algorithm	*NIST level*	*Public Key (bytes)*	*Ciphertext (bytes)*	*Key Generate (ms)*	*Encapsulate (ms)*	*Decapsulate (ms)*	*Notation*
FFDHE [1]	0	256	256	203.920	204.080 [4]	-	*FFDHE*
ECDHE [2]	0	32	32	8.428	17.687 [4]	-	*ECDHE*
Kyber512	1	800	768	8.133	6.239	3.419	*Kyb1*
BIKE-L1	1	1541	1573	200.620	25.969	411.308	*Bike1*
HQC128	1	2249	4481	30.202	50.682	72.775	*Hqc1*
SIKEp434	1	330	346	309.235	498.227	530.200	*Sike1*
SIKEp503	2	378	402	423.708	690.188	733.071	*Sike2*
SIKEp610	3	462	486	732.786	1341.125	1346.500	*Sike3*
Kyber768	3	1184	1088	12.224	11.412	7.924	*Kyb3*
SIKEp751	5	564	596	1262.750	2036.000	2183.000	*Sike5*
Kyber1024	5	1568	1568	12.918	11.623	8.539	*Kyb5*
Auth. Algorithm	***NIST level***	***Public Key (bytes)***	***Signature (bytes)***	***Key Generate (ms)***	***Sign (ms)***	***Verify (ms)***	***Notation***
RSA [3]	0	256	256	12.853/450 [5]	448.250	12.500	*RSA*
ECDSA [2]	0	32	32	8.428	12.305	25.193	*ECDSA*
Sphincs128s	1	32	7856	8674.000	66 239.000	61.588	*Sphi1s*
Sphincs128f	1	32	17088	137.750	3361.000	190.167	*Sphi1f*
Falcon512	1	897	666	1266.667	243.881	3.275	*Falc1*
Dilithium2	2	1312	2420	12.063	25.404 [6]	9.569	*Dil2*
Dilithium3	3	1952	3293	19.438	39.309 [7]	16.244	*Dil3*
Falcon1024	5	1793	1280	4802.667	527.789	6.852	*Falc5*

[1] 3072-bit, [2] secp256r1 curve, [3] 2048-bit, [4] key agreement time, [5] public / private key generation time, [6] (11/114) (min/max), [7] (15/138) (min/max) of execution time over 1000 signatures

Handshake 측정

- 클라이언트에서 수집한 측정값 : Key Generate, Decapsulation
- 서버에서 수집한 측정값 : Encapsulation
- **따라서, 보드가 클라이언트와 서버로 동작할 때, TLS Handshake의 비대칭적인 실행 시간 발생**



Handshake 측정

- 실험 측정 결과와 정적 메모리 소비, Handshake의 통신 바이트 크기
  - Bike의 경우, 클라이언트 handshake 시간이 서버보다 5.6배 느림
  - Bike1과 Hqc2를 제외하였을 때, **일반적으로 평균 6.25ms 더 작음**

Table 2. PQ TLS 1.3 Handshake Measurements.

Notation	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (ms)*	
				client	*server*
		Selected Algorithms for Standardisation			
Dil2-Kyb1	49 648	0	14 748	96.318	91.062
Falc1-Kyb1	3680	39 936	6833	288.305	285.951
Dil3-Kyb3	69 072	0	20 224	157.126	153.492
Falc5-Kyb3	4200	79 872	11 789	594.495	589.058
Dil3-Kyb5	69 104	0	21 088	165.590	152.537
Falc5-Kyb5	4712	79 872	12 647	601.827	592.302
Sph1s-Kyb1	800	0	33 892	66 977.000	66 776.000
		4rth Round Algorithms			
Dil2-Bike1	81 528	49	16 292	690.000	121.756
Dil2-Hqc1	71 672	0	19 910	198.603	145.989
Dil2-Sike1	49 648	0	13 858	886.359	566.125
Dil2-Sike2	49 648	0	13 962	1196.510	760.265
Dil2-Sike3	49 648	0	14 130	2089.690	1416.368
Dil2-Sike5	49 648	0	14 342	3403.222	2149.923
Dil3-Sike3	69 232	0	18 902	2246.077	1529.143
Dil3-Sike5	69 104	0	19 114	3450.167	2170.840
		Traditional Algorithms			
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

Handshake 측정

- 상호인증의 경우, **KEM+Dilithium의 조합은 KEM+Falcon 보다 성능이 뛰어남**
  - **Falon이 "Verify"는 매우 빠르지만, "Sign"은 느리기 때문**

Table 2. PQ TLS 1.3 Handshake Measurements.

Notation	*Static Usage (bytes)*	*.bss Usage (bytes)*	*Communication Sizes (bytes)*	*Avg Handshake Time (ms)*	
				client	*server*
		Selected Algorithms for Standardisation			
Dil2-Kyb1	49 648	0	14 748	96.318	91.062
Falc1-Kyb1	3680	39 936	6833	288.305	285.951
Dil3-Kyb3	69 072	0	20 224	157.126	153.492
Falc5-Kyb3	4200	79 872	11 789	594.495	589.058
Dil3-Kyb5	69 104	0	21 088	165.590	152.537
Falc5-Kyb5	4712	79 872	12 647	601.827	592.302
Sph1s-Kyb1	800	0	33 892	66 977.000	66 776.000
		4rth Round Algorithms			
Dil2-Bike1	81 528	49	16 292	690.000	121.756
Dil2-Hqc1	71 672	0	19 910	198.603	145.989
Dil2-Sike1	49 648	0	13 858	886.359	566.125
Dil2-Sike2	49 648	0	13 962	1196.510	760.265
Dil2-Sike3	49 648	0	14 130	2089.690	1416.368
Dil2-Sike5	49 648	0	14 342	3403.222	2149.923
Dil3-Sike3	69 232	0	18 902	2246.077	1529.143
Dil3-Sike5	69 104	0	19 114	3450.167	2170.840
		Traditional Algorithms			
RSA-ECDHE	2368	0	3742	540.220	538.158
ECDSA-ECDHE	2368	0	2353	109.171	106.927

Q & A