

PQC TLS+ PQC 인증서 실습 (NIST PQC)

<https://youtu.be/Abe6yhnxt4I>

PQC TLS+ PQC 인증서 테스트

- Case 1
 - 기존 인증서(현재 사용되는 인증서) 사용+ 하이브리드 TLS
- Case 2
 - PQC 인증서 + 하이브리드 TLS
- Case 3
 - PQC 인증서 + PQC TLS
- M1 chip에서 테스트 수행
- 사전 설치 항목 : Homebrew
- liboqs와 openssl3 활용

***TLS의 경우 모두 키 교환 과정이 하이브리드 혹은 PQC를 의미**

open-quantum-safe project

- Open Quantum Safe(OQS) 프로젝트

- 양자 컴퓨터 시대에도 안전한 암호 알고리즘을 개발 및 프로토타이핑하고 이를 프로토콜 및 애플리케이션에 통합 지원하는 것을 목표로 함
- Linux Foundation 산하 PQCA 프로젝트 소속
- 구성
 - liboqs 프로젝트
 - 프로토콜과 애플리케이션에 대한 프로토타입 통합
- 최신 업데이트 : 2025.04.17 liboqs 버전 0.13.0 출시

Recent updates

- April 17, 2025: Release of [liboqs version 0.13.0](#)
- April 10, 2025: [OQS community survey open for responses](#)
- April 9, 2025: [Blog post about 1 year anniversary of Post-Quantum Cryptography Alliance](#)
- April 3, 2025: Trail of Bits releases [public report on their 2024 security assessment of liboqs](#)
- January 30, 2025: Release of [oqs](#) and [oqs-sys 0.10.1 Rust crates](#)
- January 24, 2025: Release of [liboqs-java 0.2.0](#)
- January 21, 2025: Release of [OQS-BoringSSL snapshot 2025-01](#)
- January 16, 2025: Release of [liboqs-cpp 0.12.0](#), [liboqs-go 0.12.0](#), and [liboqs-python 0.12.0](#)
- December 24, 2024: Release of [oqs-provider 0.8.0](#)
- December 9, 2024: Release of [liboqs version 0.12.0](#)
- October 10, 2024: Release of [OQS-BoringSSL snapshot 2024-10](#)
- October 8, 2024: Release of [oqs-provider 0.7.0](#)
- September 27, 2024: Release of [liboqs version 0.11.0](#)
- August 30, 2024: Release of [OQS-OpenSSH snapshot 2024-08](#)
- June 14, 2024: Release of [oqs-provider 0.6.1](#)
- June 7, 2024: Release of [liboqs version 0.10.1](#)

- liboqs

- OQS 프로젝트 중 하나로, 양자 내성 암호 알고리즘들을 모아 제공하는 오픈 소스 라이브러리
- PQC 알고리즘을 실제 시스템에 적용하기 위해 주로 활용됨
- 다양한 KEM(Key Encapsulation Mechanism)과 전자서명 알고리즘에 대한 공통 API 제공
- OpenSSL PQC 통합(OQS OpenSSL 3 Provider), SSH, VPN 등의 프로토콜 데모로 활용

• 특징

- 공통 API 데이터 구조 : OQS_KEM, OQS_SIG 구조체 사용
 - 공통 구조체를 사용하여 알고리즘 이름, 키 크기, 보안 수준, 함수 포인터 등을 보관
 - OQS_KEM_keypair, OQS_KEM_encaps
- 다수의 구현체 통합
 - 공식 구현 참조, PQClean 구현 등 코드 통합
- 테스트 및 벤치마크 지원
 - 각 알고리즘에 대한 KAT(Known Answer Test), 속도 벤치마크 코드 포함

liboqs / src / kem / ml_kem / kem_ml_kem_512.c

SWilson4 and Eddy-M-K Add DeriveKeyPair API (#2070)

```

1 // SPDX-License-Identifier: MIT
2
3 #include <stdlib.h>
4
5 #include <oqs/kem_ml_kem.h>
6
7 #if defined(OQS_ENABLE_KEM_ml_kem_512)
8
9 OQS_KEM *OQS_KEM_ml_kem_512_new(void) {
10
11     OQS_KEM *kem = OQS_MEM_malloc(sizeof(OQS_KEM));
12     if (kem == NULL) {
13         return NULL;
14     }
15     kem->method_name = OQS_KEM_alg_ml_kem_512;
16     kem->alg_version = "FIPS203";
17
18     kem->claimed_nist_level = 1;
19     kem->ind_cca = true;
20
21     kem->length_public_key = OQS_KEM_ml_kem_512_length_public_key;
22     kem->length_secret_key = OQS_KEM_ml_kem_512_length_secret_key;
23     kem->length_ciphertext = OQS_KEM_ml_kem_512_length_ciphertext;
24     kem->length_shared_secret = OQS_KEM_ml_kem_512_length_shared_secret;
25     kem->length_keypair_seed = OQS_KEM_ml_kem_512_length_keypair_seed;
26
27     kem->keypair = OQS_KEM_ml_kem_512_keypair;
28     kem->keypair_derand = OQS_KEM_ml_kem_512_keypair_derand;
29     kem->encaps = OQS_KEM_ml_kem_512_encaps;
30     kem->decaps = OQS_KEM_ml_kem_512_decaps;

```

[지원 알고리즘 목록]	KEM	DSA
NIST PQC	Kyber (Round 3 ver), ML-KEM (FIPS 203 ver) HQC (Round4 ver)	Dilithium (Round 3 ver), Falcon , ML-DSA (FIPS 204 ver), SPHINCS+
Additional Round2	-	MAYO , CROSS(ver 2) , UOV
상태 기반 해시	-	LMS , XMSS
NIST PQC Round 4	Classic McEliece , BIKE	-
기타	FrodoKEM , NTUR-Prime	-

TLS

- Transport Layer Security

- 컴퓨터 네트워크 상에서 안전한 통신을 보장하기 위한 암호화 프로토콜
- 일반적으로 TCP(Transmission Control Protocol, 전송 제어 프로토콜) 위에서 동작
- TLS 1.3은 2018년에 표준화된 최신 버전
 - 기존 버전보다 더 강화된 보안성, 지연 시간 감소, 단순화된 설정 제공
 - Client-Server 사이의 인증, 데이터 암호화, 무결성 제공

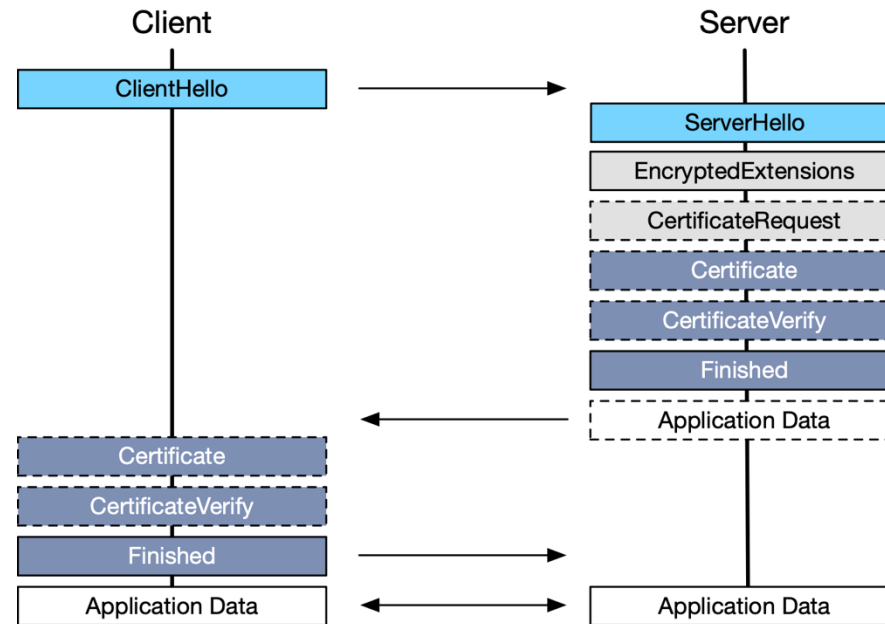


Fig. 1: TLS 1.3 Message Flow (based on RFC 8446)

사전 작업

1. 필요 패키지 설치

- brew install cmake ninja openssl@3 git bc

2. Liboqs 설치

cd ~

기존에 있다면 삭제

rm -rf liboqs

git clone --recursive https://github.com/open-quantum-safe/liboqs.git

cd liboqs

cmake -GNinja \

-DCMAKE_INSTALL_PREFIX=/usr/local \

-DCMAKE_OSX_ARCHITECTURES=arm64 \

-S . -B build

cmake --build build

sudo cmake --install build

-GNinja : Ninja 빌드 시스템 생성
-DCMAKE_INSTALL_PREFIX : make install 시 설치 경로 지정
-DCMAKE_OSX_ARCHITECTURES=arm64 : Apple Silicon 전용 컴파일
-S . -B build : 현재 소스 → build/ 폴더로 빌드 파일 생성

사전 작업

-DCMAKE_BUILD_TYPE=Release : 최적화 빌드
-DOPENSSL_ROOT_DIR=... : Homebrew OpenSSL 위치 지정
-Dliboqs_DIR=... : liboqs CMake 설정 파일 위치 지정
-DBUILD_SHARED_LIBS=ON : .dylib 형태로 빌드

3. oqs-provider 설치

cd ~

rm -rf oqs-provider

git clone https://github.com/open-quantum-safe/oqs-provider.git

cd oqs-provider

cmake -GNinja \

-DCMAKE_BUILD_TYPE=Release \

-DOPENSSL_ROOT_DIR=\$(brew --prefix openssl@3) \

-Dliboqs_DIR=/usr/local/lib/cmake/liboqs \

-DCMAKE_OSX_ARCHITECTURES=arm64 \

-DBUILD_SHARED_LIBS=ON \

-S . -B build

cmake --build build

Case 1) 기존 인증서 사용+ 하이브리드 TLS

- P-256+ML-KEM512 키 교환(TLS 1.3)

1) 쉘 환경 설정

(1) Homebrew OpenSSL 3.x 우선 호출

```
export PATH="$$(brew --prefix openssl@3)/bin:$PATH"
```

(2) oqs-provider 모듈 위치 지정(OpenSSL이 외부 provider 모듈을 찾는 경로 지정)

```
export OPENSSL_MODULES="$HOME/oqs-provider/build/lib"
```

#(3) 변경된 환경 변수 즉시 반영

```
source ~/.zshrc
```

```
[(base) minjoo@simminjuui-iMac oqs-provider % which openssl
/opt/homebrew/opt/openssl@3/bin/openssl
```

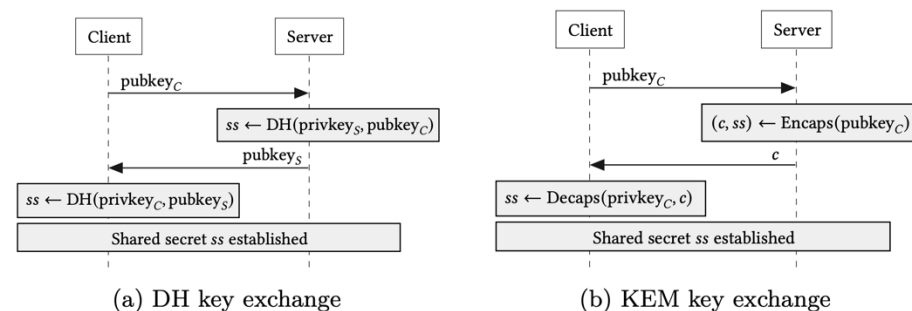


Figure 2: Key exchange diagrams

Case 1) 기존 인증서 사용+ 하이브리드 TLS

2) 서버용 인증서 및 키 준비

cd ~

ECDSA P-256 키·인증서 생성

openssl ecparam -name **prime256v1** -genkey -noout -out server.key

openssl req -x509 -key server.key -out server.crt -nodes \

-subj "/CN=localhost"

ecparam -name prime256v1 :
P-256(ECDSA) 파라미터 사용
-genkey -noout : 키만 생성(파라미터 출력 생략)
req -x509 : self-signed X.509 인증서 생성
-nodes : 키 암호화 없이 저장
-subj "/CN=localhost" : 인증서 Subject 설정

3) 벤치마크용 스크립트 생성

cd ~

cat << 'EOF' > bench.sh

#!/usr/bin/env bash

HOST=localhost

PORT=8443

N=200 # 반복 횟수 (원하는 만큼 조절 가능)

PROV_PATH="\$HOME/oqs-provider/build/lib"

시작 시간

START=\$(date +%s.%N)

for i in \$(seq 1 \$N); do

printf " | openssl s_client \

**-connect \${HOST}:\${PORT} **

**-tls1_3 **

**-groups p256_mlkem512 **

**-provider default -provider base -provider oqsprovider **

**-provider-path "\${PROV_PATH}" **

> /dev/null 2>&1

done

END=\$(date +%s.%N)

결과 계산

ELAPSED=\$(echo "\$END - \$START" | bc)

echo "→ \$N handshakes in \${ELAPSED}s"

echo "→ Avg handshake time: \$(echo "\$ELAPSED / \$N" | bc -l)s"

echo "→ Throughput: \$(echo "\$N / \$ELAPSED" | bc -l) handshakes/sec"

EOF

chmod +x bench.sh

Case 1) 기존 인증서 사용+ 하이브리드 TLS

4) TLS 서버 띄우기(서버용 새로운 터미널)

셸 환경 설정(앞과 동일, 새로운 터미널 켜면 무조건 수행)

(1) Homebrew OpenSSL 3.x 우선 호출

export PATH="\$\$(brew --prefix openssl@3)/bin:\$PATH"

(2) oqs-provider 모듈 위치 지정

export OPENSSL_MODULES="\$HOME/oqs-provider/build/lib"

(3) TLS 서버 터미널 동작

openssl s_server \

-accept 8443 \

-cert server.crt

-key server.key \

-www -tls1_3 \

-groups p256_mlkem512 \

-provider default -provider base -provider oqsprovider \

-provider-path "\$OPENSSL_MODULES"

```
(base) minjoo@simminjuui-iMac ~ % openssl s_server \
-accept 8443 \
-cert server.crt -key server.key \
-www -tls1_3 \
-groups p256_mlkem512 \
-provider default -provider base -provider oqsprovider \
-provider-path "$OPENSSL_MODULES"
```

```
s_server: unable to load provider oqsprovider
Hint: use -provider-path option or OPENSSL_MODULES environment variable.
80CC4BEF01000000:error:12800067:DSO support routines:dlfcn_load:could not load the shared library
:crypto/dso/dso_dlfcn.c:118:filename(/opt/homebrew/Cellar/openssl@3/3.4.0/lib/openssl-modules/oqspro
vider.dylib): dlopen(/opt/homebrew/Cellar/openssl@3/3.4.0/lib/openssl-modules/oqsprovider.dylib, 0x0
002): tried: '/opt/homebrew/Cellar/openssl@3/3.4.0/lib/openssl-modules/oqsprovider.dylib' (no such f
ile), '/System/Volumes/Preboot/Cryptexes/OS/opt/homebrew/Cellar/openssl@3/3.4.0/lib/openssl-modules/
oqsprovider.dylib' (no such file), '/opt/homebrew/Cellar/openssl@3/3.4.0/lib/openssl-modules/oqsprov
ider.dylib' (no such file)
80CC4BEF01000000:error:12800067:DSO support routines:DSO_load:could not load the shared library:c
rypto/dso/dso_lib.c:147:
80CC4BEF01000000:error:07880025:common libcrypto routines:provider_init:reason(37):crypto/provide
r_core.c:950:name=oqsprovider
```

```
(base) minjoo@simminjuui-iMac ~ % openssl s_server \
-accept 8443 \
-cert server.crt -key server.key \
-www -tls1_3 \
-groups p256_mlkem512 \
-provider default -provider base -provider oqsprovider \
-provider-path "$OPENSSL_MODULES"
```

Using default temp DH parameters

ACCEPT

Case 1) 기존 인증서 사용+ 하이브리드 TLS

5) Handshake 벤치마크 실행(클라이언트 터미널_기존 터미널)
./bench.sh

ECDSA P-256 키·인증서 생성

P-256+ML-KEM512 키 교환(TLS 1.3)

```
(base) minjoo@simminjuui-iMac ~ % openssl s_server \  
-accept 8443 \  
-cert server.crt -key server.key \  
-www -tls1_3 \  
-groups p256_mlkem512 \  
-provider default -provider base -provider oqsprovider \  
-provider-path "$OPENSSL_MODULES"
```

```
Using default temp DH parameters  
ACCEPT  
█
```

서버

```
echo "> Avg handshake time: $(echo "$ELAPSED / $N" | bc -l)s"  
echo "> Throughput: $(echo "$N / $ELAPSED" | bc -l) handshakes/sec"  
EOF
```

```
chmod +x bench.sh
```

```
(base) minjoo@simminjuui-iMac ~ % ./bench.sh
```

```
→ 200 handshakes in 7.571191000s  
→ Avg handshake time: .03785595500000000000s  
→ Throughput: 26.41592320151479470006 handshakes/sec
```

```
(base) minjoo@simminjuui-iMac ~ % █
```

클라이언트

Case 2) PQC 인증서 + 하이브리드 TLS

Case1의 쉘환경 설정 동일하게 수행

0) 실험용 디렉터리 생성 및 이동

```
mkdir -p ~/tls-test
```

```
cd ~/tls-test
```

1) PQC 인증서와 키 생성(mldsa44)

2) TLS 서버 실행

```
openssl s_server \  
-accept 8443 \  
-cert server.crt \  
-key server.key \  
-tls1_3 \  
-groups p256_mlkem512 \  
-provider default \  
-provider base \  
-provider oqsprovider
```

① 개인키(server.key) 생성

```
openssl genpkey \  
-provider default \  
-provider base \  
-provider oqsprovider \  
-algorithm mldsa44 \  
-out server.key
```

② Self-signed 인증서(server.crt) 발급 (유효기간 1년)

```
openssl req -new -x509 \  
-provider default \  
-provider base \  
-provider oqsprovider \  
-key server.key \  
-out server.crt \  
-days 365 -nodes \  
-subj "/C=KR/ST=Seoul/L=Seoul/O=MyOrg/OU=IT/CN=localhost"
```

-accept 8443 : 8443 포트 대기

-groups p256_mlkem512 : P-256 + Kyber512 하이브리드 KEM
oqsprovider 로딩으로 Kyber·mldsa44 동작

Case 2) PQC 인증서 + 하이브리드 TLS

[클라이언트 터미널]

3) 핸드셰이크 기능 검증

cd ~/tls-test

openssl s_client \

-connect localhost:8443 \

-tls1_3 \

-groups p256_mlkem512 \

-provider default \

-provider base \

-provider oqsprovider \

-CAfile server.crt \

-msg

```
-----
Certificate chain
 0 s:C=KR, ST=Seoul, L=Seoul, O=MyOrg, OU=IT, CN=localhost
  i:C=KR, ST=Seoul, L=Seoul, O=MyOrg, OU=IT, CN=localhost
  a:PKEY: UNDEF, 128 (bit); sigalg: mldsa44
  v:NotBefore: Apr 27 14:02:58 2025 GMT; NotAfter: Apr 27 14:02:58 2026 GMT
-----
```

tls-test -- openssl s_server -accept 8443 -cert server.crt -key server.key -tl...

```
ple:sphincsshake128fsimple:p256_sphincsshake128fsimple:rsa3072_sphincsshake128fs
imple:mayo1:p256_mayo1:mayo2:p256_mayo2:mayo3:p384_mayo3:mayo5:p521_mayo5:CROSSr
sdp128balanced:OV_Is_pkc:p256_OV_Is_pkc:OV_Ip_pkc:p256_OV_Ip_pkc:OV_Is_pkc_sk:p
256_OV_Is_pkc_sk:OV_Ip_pkc_sk:p256_OV_Ip_pkc_sk
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:Ed25519:Ed44
8:ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+S
HA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA
512:mldsa44:p256_mldsa44:rsa3072_mldsa44:mldsa44_pss2048:mldsa44_rsa2048:mldsa44
_ed25519:mldsa44_p256:mldsa44_bp256:mldsa65:p384_mldsa65:mldsa65_pss3072:mldsa65
_rsa3072:mldsa65_p256:mldsa65_bp256:mldsa65_ed25519:mldsa87:p521_mldsa87:mldsa87
_p384:mldsa87_bp384:mldsa87_ed448:falcon512:p256_falcon512:rsa3072_falcon512:fal
conpadded512:p256_falconpadded512:rsa3072_falconpadded512:falcon1024:p521_falcon
1024:falconpadded1024:p521_falconpadded1024:sphincsha2128fsimple:p256_sphincssh
a2128fsimple:rsa3072_sphincsha2128fsimple:sphincsha2128ssimple:p256_sphincsha
2128ssimple:rsa3072_sphincsha2128ssimple:sphincsha2192fsimple:p384_sphincsha2
192fsimple:sphincsshake128fsimple:p256_sphincsshake128fsimple:rsa3072_sphincsha
ke128fsimple:mayo1:p256_mayo1:mayo2:p256_mayo2:mayo3:p384_mayo3:mayo5:p521_mayo5
:CROSSrSDP128balanced:OV_Is_pkc:p256_OV_Is_pkc:OV_Ip_pkc:p256_OV_Ip_pkc:OV_Is_pk
c_sk:p256_OV_Is_pkc_sk:OV_Ip_pkc_sk:p256_OV_Ip_pkc_sk
Supported groups: p256_mlkem512
Shared groups: p256_mlkem512
CIPHER is TLS_AES_256_GCM_SHA384
This TLS version forbids renegotiation.
```

tls-test -- openssl s_client -connect localhost:8443 -tls1_3 -groups p256_...

```
TLS session ticket lifetime hint: 7200 (seconds)
TLS session ticket:
0000 - aa 3f 52 f0 be 9e 4f 74-03 01 de 5f fd 61 27 b3 .?R...Ot...a'.
0010 - 62 4c 85 e5 f1 1a 94 94-cb 4d fa c5 36 f6 43 48 bL.....M..6.CH
0020 - 66 92 35 46 61 e7 da 09-f5 5b 4b a8 c8 d8 aa b7 f.5Fa....[K....
0030 - 6d 7a cf fe 88 c2 95 f9-76 d9 39 5e 6a 9b 78 d8 mz.....v.9^j.x.
0040 - 10 83 15 29 2e 83 f6 78-c3 8d 64 b8 00 d6 ab f7 ...)....x.d....
0050 - e6 b9 4c 8e 1b 41 50 47-1c 35 04 ca 2c 3f 1e 51 ..L..APG.5...?..Q
0060 - 17 f2 32 c6 08 6a 58 4e-68 f4 aa 2e 38 28 dc 1c ..2..jXNh...8(..
0070 - b2 d3 bb 99 a7 aa 39 cf-27 22 8b 9c fe 0e 7d d1 .....9.'"...}.
0080 - 5d 26 53 56 82 f6 d0 c9-be 05 72 72 9a 8b f8 40 ]&SV.....rr...@
0090 - b2 8f 11 ba a2 e2 a8 b0-1d 27 e6 bf df 52 2a 09 .....R*.
00a0 - 1e d3 ed 1b e5 eb 14 fc-18 81 cc 01 96 f8 91 92 .....
00b0 - 64 8e 7c 36 9c 47 56 ff-0a d7 fb 82 8f 5c d3 4a d.|6.GV.....\J
00c0 - f5 9e 56 54 5e 35 51 8b-c1 cc 93 57 7b 0b 78 01 ..VT^5Q....W{x.

Start Time: 1745762590
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0

read R BLOCK
```

-groups p256_mlkem512 : ClientHello 에 P-256+Kyber512 제안
(키교환만 하이브리드)
-CAfile server.crt : Self-signed 인증서 신뢰
-msg : 메시지 레벨 로그 출력

Case 2) PQC 인증서 + 하이브리드 TLS

- 10초 동안 몇번 핸드셰이크가 돌아가는지 확인

```
minjoo ~ openssl s_server -accept 8443 -cert server.crt -key server.key -tl...
256_OV_Is_pkc_skc:OV_Is_pkc_skc:p256_OV_Is_pkc_skc
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:Ed25519:Ed44
8:ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+S
HA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA
512:mldsa44:p256_mldsa44:rsa3072_mldsa44:mldsa44_pss2048:mldsa44_rsa2048:mldsa44
_ed25519:mldsa44_bp256:mldsa44_bp256:mldsa65:p384_mldsa65:mldsa65_pss3072:mldsa65
_rsa3072:mldsa65_p256:mldsa65_bp256:mldsa65_ed25519:mldsa87:p521_mldsa87:mldsa87
_p384:mldsa87_bp384:mldsa87_ed448:falcon512:p256_falcon512:rsa3072_falcon512:fal
conpadded512:p256_falconpadded512:rsa3072_falconpadded512:falcon1024:p521_falcon
1024:falconpadded1024:p521_falconpadded1024:sphincssha2128fsimple:p256_sphincssh
a2128fsimple:rsa3072_sphincssha2128fsimple:sphincssha2128ssimple:p256_sphincssh
a2128ssimple:rsa3072_sphincssha2128ssimple:sphincssha212fsimple:p384_sphincssh
a212fsimple:sphincsshake128fsimple:p256_sphincsshake128fsimple:rsa3072_sphincssh
ake128fsimple:mayo1:p256_mayo1:mayo2:p256_mayo2:mayo3:p384_mayo3:mayo5:p521_mayo5
:CROSSrsdp128balanced:OV_Is_pkc:p256_OV_Is_pkc:OV_Is_pkc:p256_OV_Is_pkc:OV_Is_pk
c_skc:p256_OV_Is_pkc_skc:OV_Is_pkc_skc:p256_OV_Is_pkc_skc
Supported groups: p256_mlkem512
Shared groups: p256_mlkem512
CIPHER is TLS_AES_256_GCM_SHA384
This TLS version forbids renegotiation.
DONE
shutting down SSL
CONNECTION CLOSED
[]
```

```
cat > ~/tls-test/time-bench.sh << 'EOF'
#!/usr/bin/env bash
HOST=localhost
PORT=8443
DURATION=10 # 측정 시간(초)
PROV_PATH="$HOME/oqs-provider/build/lib"
```

```
# 환경 체크
export PATH="$(brew --prefix openssl@3)/bin:$PATH"
export OPENSSL_MODULES="$PROV_PATH"
```

```
START_TS=$(date +%s)
END_TS=$((START_TS + DURATION))
COUNT=0
```

```
while [ "$(date +%s)" -lt "$END_TS" ]; do
# 빈 줄 입력 → 핸드셰이크. 출력 모두 버림
printf " | openssl s_client \
  -connect ${HOST}:${PORT} \
  -tls1_3 \
  -groups p256_mlkem512 \
  -provider default -provider base -provider oqsprovider \
  -provider-path "${PROV_PATH}" \
  -CAfile server.crt \
  > /dev/null 2>&1
COUNT=$((COUNT + 1))
done
```

```
echo "→ $COUNT handshakes in ${DURATION}s"
printf "→ %.2f handshakes/sec\n" "$(bc -l <<< "$COUNT / $DURATION")"
printf "→ Avg handshake time: %.4fs\n" "$(bc -l <<< "$DURATION / $COUNT")"
EOF
```

```
chmod +x ~/tls-test/time-bench.sh
```

Case 3) PQC 인증서 + PQC TLS

0. Case1의 쉘환경 설정 동일하게 수행

Cas2의 실험용 디렉터리로 이동

1. PQCC 서명용 개인키(Falcon512 생성) 및 인증서 생성

`openssl genpkey \`

`-provider default \`

`-provider base \`

`-provider oqsprovider \`

`-algorithm falcon512 \`

`-out server.key`

`openssl req -new -x509 \`

`-provider default \`

`-provider base \`

`-provider oqsprovider \`

`-key server.key \`

`-out server.crt \`

`-days 365 \`

`-subj "/CN=localhost"`

Case 3) PQC 인증서 + PQC TLS

2. 서버 실행(순수 ML-KEM512-only)

```
-accept 8443 \  
-cert server.crt \  
-key server.key \  
-tls1_3 \  
-groups mlkem512 \  
-provider default \  
-provider base \  
-provider oqsprovider \  
-provider-path $PROV_PATH \  
-www
```

키 교환(KEM): Kyber512 (mlkem512)
인증서 서명: Falcon512

Case 3) PQC 인증서 + PQC TLS

[서버]

3. 단일 핸드셰이크

openssl s_client \

-connect localhost:8443 \

-tls1_3 \

**-groups mlkem512 **

-provider default \

-provider base \

-provider oqsprovider \

-provider-path \$PROV_PATH

```
read R BLOCK
GET / HTTP/1.0
Host: localhost
<빈 줄>
HTTP/1.0 200 ok
Content-type: text/html
```

입력하면 아래와 같이 나옴

```
<HTML><BODY BGCOLOR="#ffffff">
<pre>
```

```
s_server -accept 8443 -cert server.crt -key server.key -tls1_3 -groups mlkem512
-provider default -provider base -provider oqsprovider -provider-path /Users/min
joo/oqs-provider/build/lib -www
This TLS version forbids renegotiation.
```

```
Supported groups: mlkem512
Shared groups: mlkem512
```

```
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 3BC113F5764913F22BDA09945AC63AC75EC399FABFCBEFB78B2E63524FADD384
    Session-ID-ctx: 01000000
    Resumption PSK: DFB39D651032A1A0CEDF987431DAE2B3B9D3B57CBB7640D9A9C6239B0441
    6F2CA0FC02CAB4398233FEF9FD78EFA5093B
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1745764946
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
```

Case 3) PQC 인증서 + PQC TLS

[서버]

4. 100회 핸드셰이크

```
chmod +x bench_loop_100.sh
bash bench_loop_100.sh
```

✓ 성공 연결 : 100 / 100

🕒 총 소요 시간 : 2.968213000s

⚡ 처리량 : 33.69 connections/sec

🕒 평균 Latency: 20.00 ms

```
cat > bench_loop_100.sh << 'EOF'
#!/usr/bin/env bash
```

```
export PATH="$(brew --prefix openssl@3)/bin:$PATH"
export OPENSSL_MODULES=~/.oqs-provider/build/lib
export PROV_PATH=~/.oqs-provider/build/lib
```

```
N=100
START=$(date +%s.%N)
SUCCESS=0
```

```
# POSIX 방식 for 루프
for i in $(seq 1 $N); do
  if openssl s_client \
    -connect localhost:8443 \
    -tls1_3 \
    -groups mlkem512 \
    -provider default -provider base -provider oqsprovider \
    -provider-path $PROV_PATH \
    < /dev/null \
    > /dev/null 2>&1; then
    SUCCESS=$((SUCCESS+1))
  fi
done
```

```
END=$(date +%s.%N)
ELAPSED=$(echo "$END - $START" | bc)
TPS=$(echo "scale=2; $SUCCESS / $ELAPSED" | bc)
AVG_MS=$(echo "scale=2; ($ELAPSED / $SUCCESS) * 1000" | bc)
```

```
echo "✓ 성공 연결: $SUCCESS / $N"
echo "🕒 총 소요 시간: ${ELAPSED}s"
echo "⚡ 처리량: ${TPS} connections/sec"
echo "🕒 평균 Latency: ${AVG_MS} ms"
EOF
```

```
chmod +x bench_loop_100.sh
bash bench_loop_100.sh
```

향후 진행 계획

- 현재 liboqs에 KpqC 알고리즘 4종 포팅 중
- 이번 세미나에서 진행한 테스트를 수행하기 위해서는 openssl3에도 KpqC를 포팅시켜야함
- 따라서,
 - 1) KpqC 알고리즘 포팅 수행
 - 2) liboqs 알고리즘 성능 측정하는 방법을 사용해서 포팅의 정확도 확인 먼저 수행
 - 3) OpenSSL 3에 KpqC 포팅 수행
 - 4) 이번 세미나에서 수행한 테스트를 동일하게 적용 예정

Q & A