

pqctoolkit-boringssl

pqctoolkit-boringssl은 양자내성 키 교환 및 인증 알고리즘을 추가하기 위해 OQS-BoringSSL을 포크한 프로젝트로, HAETAЕ와 SMAUG-T라는 두 가지 KpqC 제출 알고리즘을 지원하여 프로토타이핑 및 평가 목적에 사용됩니다. 이 포크는 Google의 승인을 받은 프로젝트가 아닙니다.

- [pqctoolkit-boringssl](#)
- [개요](#)
- [상태](#)
 - [제한 사항 및 보안](#)
 - [지원되는 알고리즘](#)
 - [키 교환](#)
 - [서명](#)
 - [TLS Demo](#)
- [예제 코드](#)
 - [Server](#)
 - [client](#)
 - [빌드 및 실행](#)
- [지원환경](#)

개요

pqctoolkit-library는 양자 내성 암호화 알고리즘을 위한 C 라이브러리입니다. **pqctoolkit-boringssl**은 OQS-BoringSSL에 pqctoolkit-library를 통합하여 TLS 1.3 프로토콜에서 양자내성 암호화를 평가할 수 있도록 하는 포크입니다.

상태

pqctoolkit-boringssl은 다음을 지원합니다:

- 양자내성 키 교환
- 하이브리드 키 교환 (양자내성 + 타원 곡선)
- 양자내성 디지털 서명
- 하이브리드 디지털 서명 (양자내성 + RSA / 타원 곡선)

BoringSSL에서 기본적으로 지원하는 암호화 알고리즘은 Google의 구현을 사용하며, 그렇지 않은 경우에는 pqctoolkit-library의 구현을 사용합니다.

이 포크를 운영 환경이나 민감한 데이터를 보호하기 위해 사용하는 것을 권장하지 않습니다. 이 포크는 실험적인 단계에 있으며, BoringSSL은 API 또는 ABI 안정성을 보장하지 않습니다. 자세한 내용은 아래 [제한사항 및 보안](#) 섹션을 참조하십시오.

십시오.

pqctoolkit-library와 이 통합은 어떠한 보증 없이 "있는 그대로" 제공됩니다.

제한 사항 및 보안

연구가 진전됨에 따라 지원되는 알고리즘의 보안성이 급격히 변화할 수 있으며, 고전 컴퓨터와 양자 컴퓨터 모두에 대해 불안정해질 가능성이 있습니다.

우리는 현재 NIST 포스트 양자 암호화 표준화 프로젝트가 양자 내성 알고리즘을 식별하는 가장 신뢰할 수 있는 방법이라고 믿으며, 양자내성 암호화를 배포할 때 NIST 표준화 프로젝트의 결과를 따를 것을 강력히 권장합니다.

이 문서를 작성할 당시, 이 포크에서 사용된 양자내성 알고리즘에는 알려진 취약점이 없지만, NIST 표준화 프로젝트를 포함한 표준화 커뮤니티로부터 추가적인 지침이 제공될 때까지 양자내성 알고리즘을 배포하지 않고 기다리는 것이 좋습니다.

일부 사용자들은 표준화 프로젝트가 완료되기 전에 양자내성 암호화를 배포하려 할 수 있습니다. 이러한 경우 **하이브리드 암호화**를 사용하는 것을 강력히 권장합니다. 하이브리드 암호화는 양자내성 공개 키 알고리즘을 기존 공개 키 알고리즘(예: RSA 또는 타원 곡선)과 결합하여 솔루션이 기존의 전통적 암호화보다 보안성이 떨어지지 않도록 보장합니다. 이 포크는 하이브리드 암호화를 사용할 수 있는 기능을 제공합니다.

TLS에 대한 보안 증명(예: [JKSS12] 및 [KPW13])은 PRF-ODH 가정 또는 IND-CCA KEM 형태의 능동적 보안이 필요한 키 교환 메커니즘을 요구합니다. pqctoolkit-library에서 제공되는 일부 KEM은 IND-CCA 보안을 제공하지만, 일부는 그렇지 않습니다([이 데이터 시트](#)에서 각 알고리즘이 제공하는 보안 수준을 확인할 수 있습니다). IND-CCA 보안을 제공하지 않는 경우, TLS에 대한 기존 능동 공격자 보안 증명은 적용되지 않습니다.

또한, pqctoolkit-boringssl 프로젝트는 API 또는 ABI 안정성을 보장하지 않습니다.

지원되는 알고리즘

키 교환

BoringSSL이 Google의 구현을 통해 지원하는 X25519MLKEM768 및 X25519Kyber768Draft00과 함께, 이 포크는 pqctoolkit-library에서 추가로 제공하는 양자내성 알고리즘도 지원합니다(단, 해당 알고리즘이 pqctoolkit-library에서 활성화된 경우에 한함):

- **BIKE:** `bikel1`, `p256_bikel1`, `x25519_bikel1`, `bikel3`, `p384_bikel3`, `bikel5`, `p521_bikel5`
- **CRYSTALS-Kyber:** `kyber512`, `p256_kyber512`, `x25519_kyber512`, `kyber768`, `p384_kyber768`, `kyber1024`, `p521_kyber1024`
- **FrodoKEM:** `frodo640aes`, `p256_frodo640aes`, `x25519_frodo640aes`, `frodo640shake`, `p256_frodo640shake`, `x25519_frodo640shake`, `frodo976aes`, `p384_frodo976aes`, `frodo976shake`, `p384_frodo976shake`, `frodo1344aes`, `p521_frodo1344aes`, `frodo1344shake`, `p521_frodo1344shake`
- **HQC:** `hqc128`, `p256_hqc128`, `x25519_hqc128`, `hqc192`, `p384_hqc192`, `hqc256`, `p521_hqc256`†
- **ML-KEM:** `mlkem768`, `p384_mlkem768`, `mlkem1024`, `p521_mlkem1024`
- **SMAUG:** `smaug1`, `p521_smaug1`, `smaug3`, `p521_smaug3`, `smaug5`, `p521_smaug5`

참고로, † 기호가 표시된 알고리즘은 많은 스택 사용량을 가지며, 스레드에서 실행하거나 제한된 환경에서 실행할 경우 실패가 발생할 수 있습니다.

서명

다음의 pqctoolkit-library에서 제공하는 양자내성 디지털 서명 알고리즘이 지원됩니다(해당 알고리즘이 pqctoolkit-library에서 활성화된 경우에 한함):

- **CRYSTALS-DILITHIUM:** `dilithium2`, `dilithium3`, `dilithium5`
- **Falcon:** `falcon512`, `p256_falcon512`, `falconpadded512`, `falcon1024`, `falconpadded1024`
- **HAETAETAE:** `haetae2`, `haetae3`, `haetae5`
- **MAYO:** `mayo1`, `mayo2`, `mayo3`, `mayo5`
- **ML-DSA:** `rsa3072_mldsa44`, `mldsa65`, `p384_mldsa65`, `mldsa87`
- **SPHINCS-SHA2:** `sphincssha2128fsimple`, `sphincssha2128ssimple`,
`sphincssha2192fsimple`, `sphincssha2192ssimple`, `sphincssha2256fsimple`,
`sphincssha2256ssimple`
- **SPHINCS-SHAKE:** `sphincsshake128fsimple`, `sphincsshake128ssimple`,
`sphincsshake192fsimple`, `sphincsshake192ssimple`, `sphincsshake256fsimple`,
`sphincsshake256ssimple`

TLS Demo

pqctoolkit-boring에는 기본적인 TLS 서버(`server`)와 TLS 클라이언트(`client`)가 포함되어 있어 TLS 연결을 시연하고 테스트할 수 있습니다.

pqctoolkit-library의 모든 알고리즘이 활성화된 상태에서 기본 TLS 서버를 실행하려면, `build` 디렉토리에서 다음 명령을 실행하십시오:

```
tool/bssl server -accept 4433 -sig-alg <SIG> -loop
```

여기서 `<SIG>`는 위의 [지원되는 알고리즘](#) 섹션에 나열된 양자내성 또는 하이브리드 서명 알고리즘 중 하나입니다. `sig-alg` 옵션을 생략하면 기본 고전 알고리즘 `ecdhe`와 소수 곡선 `X9_62_prime256v1`이 사용됩니다.

다른 터미널 창에서 지원되는 키 교환 알고리즘 중 하나를 요청하는 TLS 클라이언트를 실행할 수 있습니다:

```
tool/bssl client -curves <KEX> -connect localhost:4433
```

여기서 `<KEX>`는 위의 [지원되는 알고리즘](#) 섹션에 나열된 양자내성 또는 하이브리드 키 교환 알고리즘 중 하나입니다.

예제 코드

`example` 디렉토리의 예제 코드는 pqctoolkit-boringssl 라이브러리를 이용하여 양자내성암호 알고리즘을 적용한 TLS 서버와 클라이언트를 생성합니다.

Server

`ssl_utils.c`의 155번 라인의 `make_key_pair`의 함수 전달 인자로 서버의 서명 알고리즘을 결정할 수 있습니다. `include/openssl/nid.h` 파일을 참조하여 사용하고자 하는 서명 알고리즘의 NID 매크로 이름으로 변경할 수 있습니다.

`ssl_util.c` 파일 내용

```
EVP_PKEY *evp_pkey = make_key_pair(NID_haetae5);
```

client

`client.c` 파일의 56번 라인의 `SSL_CTX_set1_curves_list`의 함수 전달인자로 사용하고자 하는 키교환 알고리즘의 목록을 우선순위에 따라 지정할 수 있습니다.

```
if (SSL_CTX_set1_curves_list(ctx, "kyber512:smaug1:smaug3:smaug5") != 1) {
```

빌드 및 실행

`Makefile`을 이용해서 빌드합니다.

```
make
```

빌드 된 `server`를 실행하여 클라이언트 연결을 기다립니다.

```
$ ./server
=====
TLS server listening on port 4433
=====
Connection Information
  TLS Version: TLSv1.3
  Cipher: TLS_AES_128_GCM_SHA256
  ECDHE group: kyber512
Client sent message: Hello message from client
Closed the connection

=====
TLS server listening on port 4433
=====
```

`client`를 실행하여 TLS 연결이 이루어지고 hello 메시지를 수신함을 확인합니다.

```
$ ./client
Connecting to 127.0.0.1:4433
Connection Information
```

```
TLS Version: TLSv1.3
Cipher: TLS_AES_128_GCM_SHA256
ECDHE group: kyber512
Signature algorithm: haetae5
Cert subject: /C=KR/O=Crypto Lab
Cert issuer: /C=KR/O=Crypto Lab
Server sent message : Hello from TLS server!
Closed connection
$
```

지원환경

지원 노력을 최적화하기 위해 pqctoolkit-boringssl은 x86_64 프로세서의 Ubuntu 22.04에서 구현되고 테스트되었습니다.