pqctoolkit-library

pqctoolkit-library는 양자 안전 암호화 알고리즘을 위한 C 라이브러리입니다.

개요

pqctoolkit-librarys는 다음을 제공합니다:

- 양자 안전 키 캡슐화 메커니즘(KEM) 및 디지털 서명 알고리즘의 구현 모음; 전체 목록은 아래에서 확인할 수 있습니다.
- 이러한 알고리즘에 대한 공통 API

pqctoolkit-library는 실제 환경에서 양자 안전 암호화 기술의 배포와 테스트를 용이하게 하기 위해 애플리케이션에 통합하고 개발하는 것을 목표로 합니다. pqctoolkit-library는 Open Quantum Safe의 `liboqs` 프로젝트를 기반으로 하고 있습니다.

지원되는 알고리즘

아래 목록은 현재 pqctoolkit-library에서 지원하는 모든 알고리즘을 나타내며, NIST 경쟁 과정에서 제외된 Kyber-90s 또는 Dilithium-AES와 같은 알고리즘 변형을 제외한 실험적 알고리즘도 포함하고 있습니다.

`pqctoolkit-library`에서 NIST 표준을 구현하는 유일한 알고리즘은 각각의 비트 강도를 가진 `ML-KEM` (최종 표준) 및 `ML-DSA` (공개 초안) 변형입니다. `pqctoolkit-library`는 NIST 표준화 절차의 마무리 단계에서 NIST가 선택한 이 알고리즘 이름을 유지할 예정이며, NIST가 이 알고리즘의 구현 세부 사항을 변경할 경우, `pqctoolkit-library`는 구현을 조정하여 사용자를 이러한 잠재적 변경으로부터 보호할 것입니다.

Falcon과 SPHINCS+도 표준화 대상으로 선정되었지만, `pqctoolkit-library`의 해당 알고리즘 구현은 현재 NIST 표준 초안을 따르지 않고 라운드 3 제출을 기준으로 하고 있습니다.

또한, KpqC 공모전에 제출된 키 캡슐화 메커니즘을 위한 `SMAUG-T`와 디지털 서명을 위한 `HAETAE` 두 가지 알고리즘이 구현되었습니다.

`ML-KEM` 및 `ML-DSA` 이외의 이름은 변경될 수 있습니다. 기본적으로 `pqctoolkit-library`는 아래 나열된 모든 실험적 PQC 알고리즘을 지원하는 상태로 빌드됩니다.

키교환

- **BIKE**: BIKE-L1, BIKE-L3, BIKE-L5
- **Classic McEliece**: Classic-McEliece-348864†, Classic-McEliece-348864f†, Classic-McEliece-460896†, Classic-McEliece-460896f†, Classic-McEliece-6688128†, Classic-McEliece-6688128f†, Classic-McEliece-6960119†, Classic-McEliece-6960119f†, Classic-McEliece-8192128†, Classic-McEliece-8192128f†
- **FrodoKEM**: FrodoKEM-640-AES, FrodoKEM-640-SHAKE, FrodoKEM-976-AES, FrodoKEM-976-SHAKE, FrodoKEM-1344-AES, FrodoKEM-1344-SHAKE
- **HQC**: HQC-128, HQC-192, HQC-256
- **Kyber**: Kyber512, Kyber768, Kyber1024
- **ML-KEM**: ML-KEM-512, ML-KEM-768, ML-KEM-1024
- **NTRU-Prime**: sntrup761
- **SMAUGT**: SMAUG1, SMAUG3, SMAUG5

서명

- **CROSS**: cross-rsdp-128-balanced, cross-rsdp-128-fast, cross-rsdp-128-small†, cross-rsdp-192-balanced, cross-rsdp-192-fast, cross-rsdp-192-small†, cross-rsdp-256-balanced†, cross-rsdp-256-fast, cross-rsdp-256-small†, cross-rsdpg-128-balanced, cross-rsdpg-128-fast, cross-rsdpg-128-small, cross-rsdpg-192-balanced, cross-rsdpg-192-fast, cross-rsdpg-192-small†, cross-rsdpg-256-balanced, cross-rsdpg-256-fast, cross-rsdpg-256-small†
- **CRYSTALS-Dilithium**: Dilithium2, Dilithium3, Dilithium5
- **Falcon**: Falcon-512, Falcon-1024, Falcon-padded-512, Falcon-padded-1024
- **MAYO**: MAYO-1, MAYO-2, MAYO-3, MAYO-5†
- **ML-DSA**: ML-DSA-44-ipd (alias: ML-DSA-44), ML-DSA-65-ipd (alias: ML-DSA-65), ML-DSA-87-ipd (alias: ML-DSA-87)
- **SPHINCS+-SHA2**: SPHINCS+-SHA2-128f-simple, SPHINCS+-SHA2-128s-simple, SPHINCS+-SHA2-192f-simple, SPHINCS+-SHA2-192s-simple, SPHINCS+-SHA2-256f-simple, SPHINCS+-SHA2-256s-simple
- **SPHINCS+-SHAKE**: SPHINCS+-SHAKE-128f-simple, SPHINCS+-SHAKE-128s-simple, SPHINCS+-SHAKE-192f-simple, SPHINCS+-SHAKE-192s-simple, SPHINCS+-SHAKE-256f-simple, SPHINCS+-SHAKE-256s-simple
- **XMSS**: XMSS-SHA2_10_256, XMSS-SHA2_16_256, XMSS-SHA2_20_256, XMSS-SHAKE_10_256, XMSS-SHAKE_16_256, XMSS-SHAKE_20_256, XMSS-SHA2_10_512, XMSS-SHA2_16_512, XMSS-SHA2_20_512, XMSS-SHAKE_10_512, XMSS-SHAKE_16_512, XMSS-SHAKE_20_512, XMSS-SHA2_10_192, XMSS-SHA2_16_192, XMSS-SHA2_20_192, XMSS-SHAKE256_10_192, XMSS-SHAKE256_16_192, XMSS-SHAKE256_20_192, SHAKE256_10_256, SHAKE256_16_256, SHAKE256_20_256, XMSSMT-SHA2_20/2_256, XMSSMT-SHA2_20/4_256, XMSSMT-SHA2_40/2_256, XMSSMT-SHA2_40/4_256, XMSSMT-SHA2_40/8_256, XMSSMT-SHA2_60/3_256, XMSSMT-SHA2_60/6_256, XMSSMT-SHA2_60/12_256, XMSSMT-SHAKE_20/2_256, XMSSMT-SHAKE_20/4_256, XMSSMT-SHAKE_40/2_256, XMSSMT-SHAKE_40/4_256, XMSSMT-SHAKE_40/8_256, XMSSMT-SHAKE_60/3_256, XMSSMT-SHAKE_60/6_256, XMSSMT-SHAKE_60/12_256
- **LMS**: LMS_SHA256_H5_W1, LMS_SHA256_H5_W2, LMS_SHA256_H5_W4, LMS_SHA256_H5_W8, LMS_SHA256_H10_W1, LMS_SHA256_H10_W2, LMS_SHA256_H10_W4, LMS_SHA256_H10_W8, LMS_SHA256_H15_W1, LMS_SHA256_H15_W2, LMS_SHA256_H15_W4, LMS_SHA256_H15_W8, LMS_SHA256_H20_W1, LMS_SHA256_H20_W2, LMS_SHA256_H20_W4, LMS_SHA256_H20_W8, LMS_SHA256_H25_W1, LMS_SHA256_H25_W2, LMS_SHA256_H25_W4, LMS_SHA256_H25_W8, LMS_SHA256_H5_W8_H5_W8, LMS_SHA256_H10_W4_H5_W8, LMS_SHA256_H10_W8_H5_W8,

LMS_SHA256_H10_W2_H10_W2, LMS_SHA256_H10_W4_H10_W4, LMS_SHA256_H10_W8_H10_W8, LMS_SHA256_H15_W8_H5_W8, LMS_SHA256_H15_W8_H10_W8, LMS_SHA256_H15_W8_H15_W8, LMS_SHA256_H20_W8_H5_W8, LMS_SHA256_H20_W8_H10_W8, LMS_SHA256_H20_W8_H15_W8, LMS_SHA256_H20_W8_H20_W8

- **HAETAE**: HAETAE2, HAETAE3, HAETAE5

참고로, † 기호가 표시된 알고리즘의 경우, pqctoolkit-library에는 많은 스택 공간을 사용하는 구현이 포함되어 있습니다. 이로 인해 스레드 또는 제한된 환경에서 실행할 때 실패가 발생할 수 있습니다.

제한사항 및 보안

이 문서를 작성할 당시 이 라이브러리에서 사용되는 양자 안전 알고리즘에 알려진 취약점은 없지만, 대부분의 알고리즘 및 소프트웨어는 현재 사용 중인 알고리즘과 동일한 수준의 검토를 받지 않았기 때문에 양자 안전 알고리즘을 배포할 때 주의가 필요합니다. 특히 NIST 포스트 양자 암호화 표준화 프로젝트에서 제공하는 지침에 특별한 주의를 기울여야 합니다. 연구가 진행됨에 따라 지원되는 알고리즘의 보안이 급격히 변화할 수 있으며, 고전 및 양자 컴퓨터에 대해 불안정해질 가능성도 있습니다. 또한, `sntrup761`은 상호 운용성 테스트를 위해서만 포함되어 있다는 점에 유의하십시오정

pqctoolkit-library는 특정 알고리즘을 "선정"으로 지정할 의도가 없습니다. 알고리즘 지원은 NIST PQC 표준화와 KpqC 공모전의 방향에 따라 결정됩니다. 양자 안전 암호화를 배포할 때는 이 표준화 프로젝트의 결과에 의존할 것을 강력히 권장합니다.

일부 사용자들은 NIST PQC 표준화와 KpqC 공모전이 완료되기 전에 양자 안전 암호화를 배포하려고 할 수 있습니다. 이러한 경우 **하이브리드 암호화**를 사용하여 양자 안전 공개 키 알고리즘을 기존의 공개 키 알고리즘(예: RSA 또는 타원 곡선)과 함께 사용하는 것을 강력히 권장합니다. 이 방법은 기존의 전통적 암호화와 비교하여 보안성이 저하되지 않도록 보장합니다.

현재 이 라이브러리를 운영 환경이나 민감한 데이터를 보호하는 데 사용하는 것을 권장하지 않습니다. 이 라이브러리는 연구와 프로토타전핑을 돕기 위해 제작되었습니다. 보안 버그를 피하기 위해 최선의 노력을 기울이고 있지만, 고도의 보안 사용을 신뢰할 수 있을 정도의 감사와 분석을 받은 상태는 아닙니다.

플랫폼 제한 사항

지원 노력을 최적화하기 위해,

- 모든 플랫폼에서 모든 알고리즘이 동일하게 잘 지원되는 것은 아니며,
- 모든 컴파일러가 동일하게 잘 지원되는 것은 아닙니다. 예를 들어, 최소 GNU 컴파일러 v7.1.0 이상이 필요하며,
- pqctoolkit-library는 x86_64 프로세서의 Ubuntu 22.04에서 구현되고 테스트되었습니다.

예시 코드

다운로드 받은 두 예제 소스코드로 키교환 및 서명 알고리즘을 실행할 수 있습니다.

- 키교환: `test_kem.c`
- 서명: `test_sig.c`

KEM

pqctoolkit-library의 `include`와 `lib` 파일과 함께 다음의 명령어로 빌드합니다.

Shared library의 경우:

```
export LD_LIBRARY_PATH=/path/to/pqctoolkit-library/lib
gcc -I./include test_sig.c -L./lib -loqs -lcrypto -o test_sig
```

Static library의 경우:

```
 gcc -I./include test_kem.c -L./lib ./lib/liboqs.a -lcrypto -o test_kem
```

키교환 알고리즘을 인자로 지정하여 빌드한 파일을 실행합니다.

```
$ ./test_kem SMAUG1
Testing KEM algorithms using pqctoolkit-library
================================================================================
=====
Sample computation for KEM SMAUG1
================================================================================
=====
shared secrets are equal
$
```

SIG

pqctoolkit-library의 `include`와 `lib` 파일과 함께 다음의 명령어로 빌드합니다.

Shared library의 경우:

```
export LD_LIBRARY_PATH=/path/to/pqctoolkit-library/lib
gcc -I./include test_sig.c -L./lib -loqs -lcrypto -o test_sig
```

Static library의 경우:

```
 gcc -I./include test_sig.c -L./lib ./lib/liboqs.a -lcrypto -o test_sig
```

키교환 알고리즘을 인자로 지정하여 빌드한 파일을 실행합니다.

```
$ ./test_sig HAETAE2
Testing signature algorithms using pqctoolkit-library
================================================================================
=====
Sample computation for signature HAETAE2
================================================================================
```

```
=====
verification passes as expected
$
```