microsoft / **PQCrypto-SIDH**



master Go to file Code

**patricklonga** Merge pull req… 29 days ago **171**

| | | |
|---|---|---|
| KAT | Removing old KATs | 3 months ago |
| Visual St… | Fix potential timing leakag… | 3 months ago |
| src | Fix issue related to table l… | last month |
| tests | Some reorg of header files | 3 months ago |
| .gitignore | Initial commit | 4 years ago |
| CONTRI… | Create CONTRIBUTING.md | 4 years ago |
| LICENSE | Initial commit | 4 years ago |
| Makefile | Add compressed SIDH an… | 5 months ago |
| READM… | Add compressed SIDH an… | 5 months ago |

**README.md**

# SIDH v3.3 (C Edition)

The **SIDH** library is an efficient supersingular isogeny-based cryptography library written in C language. **Version v3.3** of the library includes the ephemeral Diffie-Hellman key exchange scheme "SIDH" [1,2], and the CCA-secure key encapsulation mechanism "SIKE" [3]. These schemes are conjectured to be secure against quantum computer attacks.

Concretely, the SIDH library includes the following KEM schemes:

- SIKEp434: matching the post-quantum security of AES128 (level 1).
- SIKEp503: matching the post-quantum security of SHA3-256 (level 2).
- SIKEp610: matching the post-quantum security of AES192 (level 3).
- SIKEp751: matching the post-quantum security of AES256 (level 5).

And the following ephemeral key exchange schemes:

- SIDHp434: matching the post-quantum security of AES128 (level 1).
- SIDHp503: matching the post-quantum security of SHA3-256 (level 2).
- SIDHp610: matching the post-quantum security of AES192 (level 3).
- SIDHp751: matching the post-quantum security of AES256 (level 5).

It also includes the following compressed KEM schemes:

- SIKEp434_compressed: matching the post-quantum security of AES128 (level 1).

## About

SIDH Library is a fast and portable software library that implements state-of-the-art supersingular isogeny cryptographic schemes. The chosen parameters aim to provide security against attackers running a large-scale quantum computer, and security against classical algorithms.

Readme

MIT License

## Releases 5

**SIDH ...** Latest
on Apr 17

+ 4 releases

## Packages

No packages published

## Contributors 8

- SIKEp503_compressed: matching the post-quantum security of SHA3-256 (level 2).
- SIKEp610_compressed: matching the post-quantum security of AES192 (level 3).
- SIKEp751_compressed: matching the post-quantum security of AES256 (level 5).

And the following compressed ephemeral key exchange schemes:

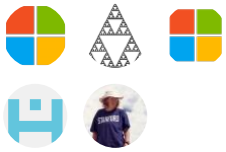
- SIDHp434_compressed: matching the post-quantum security of AES128 (level 1).
- SIDHp503_compressed: matching the post-quantum security of SHA3-256 (level 2).
- SIDHp610_compressed: matching the post-quantum security of AES192 (level 3).
- SIDHp751_compressed: matching the post-quantum security of AES256 (level 5).

The compressed schemes exhibit reduced public keys at the expense of longer computing times. Their implementation is based on [11,12], which in turn are based on and improves upon [9] and [10].

The library was developed by Microsoft Research for experimentation purposes.

## Contents

- `KAT folder` : Known Answer Test (KAT) files for the KEM.
- `src folder` : C and header files. Public APIs can be found in `P434_api.h` , `P503_api.h` , `P610_api.h` and `P751_api.h` .
- `Optimized x64 implementation for p434` : optimized implementation of the field arithmetic over the prime p434 for x64 platforms.
- `Optimized x64 implementation for p503` : optimized implementation of the field arithmetic



### Languages

- **C** 99.4%
- **Other** 0.6%

over the prime p503 for x64 platforms.

- `Optimized x64 implementation for p610` : optimized implementation of the field arithmetic over the prime p610 for x64 platforms.
- `Optimized x64 implementation for p751` : optimized implementation of the field arithmetic over the prime p751 for x64 platforms.
- `Optimized ARMv8 implementation for p434` : optimized implementation of the field arithmetic over the prime p434 for 64-bit ARMv8 platforms.
- `Optimized ARMv8 implementation for p503` : optimized implementation of the field arithmetic over the prime p503 for 64-bit ARMv8 platforms.
- `Optimized ARMv8 implementation for p610` : optimized implementation of the field arithmetic over the prime p610 for 64-bit ARMv8 platforms.
- `Optimized ARMv8 implementation for p751` : optimized implementation of the field arithmetic over the prime p751 for 64-bit ARMv8 platforms.
- `Generic implementation for p434` : implementation of the field arithmetic over the prime p434 in portable C.
- `Generic implementation for p503` : implementation of the field arithmetic over the prime p503 in portable C.
- `Generic implementation for p610` : implementation of the field arithmetic over the prime p610 in portable C.
- `Generic implementation for p751` : implementation of the field arithmetic over the prime p751 in portable C.
- `compression folder` : main C files of the compressed variants.
- `random folder` : randombytes function using the system random number generator.
- `sha3 folder` : SHAKE256 implementation.
- `Test folder` : test files.

- `Visual Studio folder` : Visual Studio 2015 files for compilation in Windows.
- `Makefile` : Makefile for compilation using the GNU GCC or clang compilers on Linux.
- `License` : MIT license file.
- `Readme` : this readme file.

## Main Features

- Supports IND-CCA secure key encapsulation mechanism.
- Supports ephemeral Diffie-Hellman key exchange.
- Includes compressed variants that feature reduced public key sizes.
- Supports four security levels matching the post-quantum security of AES128, SHA3-256, AES192 and AES256.
- Protected against timing and cache-timing attacks through regular, constant-time implementation of all operations on secret key material.
- Support for Windows OS using Microsoft Visual Studio, Linux OS and Mac OS X using GNU GCC and clang.
- Provides basic implementation of the underlying arithmetic functions using portable C to enable support on a wide range of platforms including x64, x86, ARM and s390x.
- Provides optimized implementations of the underlying arithmetic functions for x64 platforms with optional, high-performance x64 assembly for Linux and Mac OS X.
- Provides an optimized implementation of the underlying arithmetic functions for 64-bit ARM platforms using assembly for Linux.
- Includes Known Answer Tests (KATs), and

testing/benchmarking code.

## New in Version 3.3

- Improved versions of the four parameter sets for compressed SIDH and compressed SIKE [11,12].
- Optimized implementations of the field arithmetic for 64-bit ARMv8 processors for Linux.
- General optimizations to the field arithmetic.
- Support for Mac OS X for the optimized x64 assembly implementations.
- Support for big endian platforms, specifically IBM s390x processors.

## Supported Platforms

**SIDH v3.3** is supported on a wide range of platforms including x64, x86, ARM and s390x processors running Windows, Linux or Mac OS X. We have tested the library with Microsoft Visual Studio 2015, GNU GCC v5.4, and clang v3.8. See instructions below to choose an implementation option and compile on one of the supported platforms.

## Implementation Options

The following implementation options are available:

- Portable implementations enabled by setting `OPT_LEVEL=GENERIC`.
- Optimized x64 assembly implementations for Linux\Mac OS X enabled by setting `ARCH=x64` and `OPT_LEVEL=FAST`.
- Optimized ARMv8 assembly implementation for Linux enabled by setting `ARCH=ARM64` and `OPT_LEVEL=FAST`.

Follow the instructions in the sections "*Instructions for Linux*" or "*Instructions for Windows*" below to configure these different implementation options.

## Instructions for Linux

By simply executing:

```
$ make
```

the library is compiled for x64 using clang, optimization level `FAST`, and using the special instructions MULX and ADX. Optimization level `FAST` enables the use of assembly, which in turn is a requirement to enable the optimizations using MULX/ADX.

Other options for x64:

```
$ make ARCH=x64 CC=[gcc/clang] OPT_LEVEL=[FAS
```

When `OPT_LEVEL=FAST` (i.e., assembly use enabled), the user is responsible for setting the flags MULX and ADX according to the targeted platform (for example, MULX/ADX are not supported on Sandy or Ivy Bridge, only MULX is supported on Haswell, and both MULX and ADX are supported on Broadwell, Skylake and Kaby Lake architectures). Note that USE_ADX can only be set to `TRUE` if `USE_MULX=TRUE`.

Options for x86/ARM/s390x:

```
$ make ARCH=[x86/ARM/s390x] CC=[gcc/clang]
```

Options for ARM64:

```
$ make ARCH=[ARM64] CC=[gcc/clang] OPT_LEVEL=
```

As in the x64 case, `OPT_LEVEL=FAST` enables the use of assembly optimizations on ARMv8 platforms.

Different tests and benchmarking results are obtained by running:

```
$ ./arith_tests-p434
$ ./arith_tests-p503
$ ./arith_tests-p610
$ ./arith_tests-p751
$ ./sike434/test_SIKE
$ ./sike503/test_SIKE
$ ./sike610/test_SIKE
$ ./sike751/test_SIKE
$ ./sidh434/test_SIDH
$ ./sidh503/test_SIDH
$ ./sidh610/test_SIDH
$ ./sidh751/test_SIDH
$ ./sike434_compressed/test_SIKE
$ ./sike503_compressed/test_SIKE
$ ./sike610_compressed/test_SIKE
$ ./sike751_compressed/test_SIKE
$ ./sidh434_compressed/test_SIDH
$ ./sidh503_compressed/test_SIDH
$ ./sidh610_compressed/test_SIDH
$ ./sidh751_compressed/test_SIDH
```

To run the KEM implementations against the KATs, execute:

```
$ ./sike434/PQCtestKAT_kem
$ ./sike503/PQCtestKAT_kem
$ ./sike610/PQCtestKAT_kem
$ ./sike751/PQCtestKAT_kem
$ ./sike434_compressed/PQCtestKAT_kem
$ ./sike503_compressed/PQCtestKAT_kem
$ ./sike610_compressed/PQCtestKAT_kem
$ ./sike751_compressed/PQCtestKAT_kem
```

The program tries its best at auto-correcting unsupported configurations. For example, since the `FAST` implementation is currently only available for x64 and ARMv8 doing `make ARCH=x86 OPT_LEVEL=FAST` is actually processed using `ARCH=x86 OPT_LEVEL=GENERIC`.

# Instructions for Windows

## Building the library with Visual Studio:

Open the solution file `SIDH.sln` in Visual Studio, choose either x64 or Win32 from the platform menu and then choose either `Fast` or `Generic` from the configuration menu (as explained above, the option `Fast` is not currently available for x86). Finally, select "Build Solution" from the "Build" menu.

## Running the tests:

After building the solution file, there should be the following executable files: `arith_tests-P434.exe`, `arith_tests-P503.exe`, `arith_tests-P610.exe` and `arith_tests-P751.exe`, to run tests for the underlying arithmetic, `test-SIDHp[SET].exe` to run tests for the key exchange, and `test-SIKEp[SET].exe` to run tests for the KEM, where SET = {434, 503, 610, 751, 434_compressed, 503_compressed, 610_compressed, 751_compressed}.

## Using the library:

After building the solution file, add the generated `P434.lib`, `P503.lib`, `P610.lib` and `P751.lib` library files to the set of References for a project, and add `P434_api.h`, `P503_api.h`, `P610_api.h`, `P751_api.h`, `P434_compressed_api.h`, `P503_compressed_api.h`, `P610_compressed_api.h` and `P751_compressed_api.h` to the list of header files of a project.

## License

**SIDH** is licensed under the MIT License; see `License` for details.

The library includes some third party modules that are licensed differently. In particular:

- `tests/aes/aes_c.c` : public domain
- `tests/rng/rng.c` : copyrighted by Lawrence E. Bassham
- `tests/PQCtestKAT_kem<#>.c` : copyrighted by Lawrence E. Bassham
- `src/sha3/fips202.c` : public domain

## Contributors

- Basil Hess.
- Geovandro Pereira.
- Joost Renes.

# References

[1] Craig Costello, Patrick Longa, and Michael Naehrig, "Efficient algorithms for supersingular isogeny Diffie-Hellman". Advances in Cryptology - CRYPTO 2016, LNCS 9814, pp. 572-601, 2016. The extended version is available `here`.

[2] David Jao and Luca DeFeo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". PQCrypto 2011, LNCS 7071, pp. 19-34, 2011. The extended version is available here .

[3] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik, "Supersingular Isogeny Key Encapsulation". Submission to the NIST Post-Quantum Standardization project, 2017.
The round 2 submission package is available here .

[4] Craig Costello, and Huseyin Hisil, "A simple and compact algorithm for SIDH with arbitrary degree isogenies". Advances in Cryptology - ASIACRYPT 2017, LNCS 10625, pp. 303-329, 2017. The preprint version is available here .

[5] Armando Faz-Hernández, Julio López, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez, "A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol". IEEE Transactions on Computers, Vol. 67(11), 2018. The preprint version is available here .

[6] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes and Francisco Rodríguez-Henríquez, "On the cost of computing isogenies between supersingular elliptic curves". SAC 2018, LCNS 11349, pp. 322-343, 2018. The preprint version is available here .

[7] Samuel Jaques and John M. Schanck, "Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE". Advances in Cryptology - CRYPTO 2019 (to appear), 2019. The preprint version is available here .

[8] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes and Fernando Virdia, "Improved Classical Cryptanalysis of the Computational Supersingular Isogeny Problem", 2019. The preprint version is available `here` .

[9] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes and David Urbanik, "Efficient compression of SIDH public keys". Advances in Cryptology - EUROCRYPT 2017, LNCS 10210, pp. 679-706, 2017. The preprint version is available `here` .

[10] Gustavo H.M. Zanon, Marcos A. Simplicio Jr, Geovandro C.C.F. Pereira, Javad Doliskani and Paulo S.L.M. Barreto, "Faster key compression for isogeny-based cryptosystems". IEEE Transactions on Computers, Vol. 68(5), 2019. The preprint version is available `here` .

[11] Michael Naehrig and Joost Renes, "Dual Isogenies and Their Application to Public-key Compression for Isogeny-based Cryptography". Advances in Cryptology - ASIACRYPT 2019, LNCS 11922, pp. 243-272, 2019. The preprint version is available `here` .

[12] Geovandro C.C.F. Pereira, Javad Doliskani and David Jao, "x-only point addition formula and faster torsion basis generation in compressed SIKE". The preprint version is available `here` .

# Contributing

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.