

# Implementing Post-quantum Cryptography for Developers

**Authors** Julius Hekkala, Kimmo Halunen (kimmo.halunen@oulu.fi), Visa Vallivaara

## Introduction

- The security of the commonly used public key algorithms is threatened by the development of quantum computers.
- NIST is standardizing post-quantum algorithms to replace the vulnerable public key algorithms in key exchange and digital signatures.
- The availability of post-quantum algorithms in open-source libraries is often poor.

## Dangers when using open-source cryptographic libraries

- Errors in implementations
- Mistakes when using the library
  - Clear interfaces important

## Algorithm integration

- We forked Crypto++, which is a C++ open-source cryptographic library.
- Chosen algorithms: CRYSTALS-Kyber, SABER and CRYSTALS-Dilithium
- Goals in the integration process:

Usability

Conformity  
with the  
library

Reliability

Preventing  
unnecessary  
code

Performance

- Implementing the algorithms from scratch using only specification as a basis would be really difficult
  - Reference implementations were used as a basis
  - C code to C++
- Interfaces for the user to change between security levels

## Challenges in implementing post-quantum algorithms

- Mathematical complexity
- Increased key and/or ciphertext sizes

Challenges in our work:

- Handling the parameters
- Debugging
- Differences between compilers
- Changes in algorithms

Function	CPU cycles (Reference impl.)	CPU cycles (C++ default)	Runtime Increase (%)	CPU cycles (C++ with optim.)	Runtime increase (%)
Key gen.	80317	158330	97	80856	1
Encaps.	100271	193770	93	91811	-8
Decaps.	109088	218243	100	97211	-11

Performance of security level 3 SABER. The performance of the reference implementation is compared against the fork, when compiling the fork with and without the `-march=native` compiler option.

## Results

- Performance of the algorithms in the fork was generally satisfactory
- The runtime of SABER was at first double the runtime of the reference implementation
  - Reason: different compiler options in the fork and the reference implementation
  - When adding the `-march=native` compiler option, the performance of SABER was even better in the fork than in the reference implementation.
- With limited testing, we were not able to find any memory leaks.

- Because of the complexity of the algorithms, making the implementation efficient but secure can prove challenging.
- Debugging is not easy either, as the algorithms are complex and not deterministic.
- SABER has a simpler structure than Kyber
  - Probably easier to implement
- We generally achieved our goals
  - Adding exception handling would improve the usability and the code structure could be improved.
- The implemented algorithms can be used to test them in different environments and protocols.