Updates from the Open Quantum Safe Project

Open Quantum Safe core team:

Michael Baentsch
Vlad Gheorghiu, *evolutionQ & University of Waterloo*
Basil Hess, *IBM Research*
Christian Paquin, *Microsoft Research*
John Schanck, *University of Waterloo*
Douglas Stebila, *University of Waterloo*
Goutam Tamvada, *University of Waterloo*

April 23, 2021

Abstract

The Open Quantum Safe (OQS) project is an open-source project that aims to support the development and prototyping of quantum-resistant cryptography. This short note provides an update on the tools OQS makes available.

1 Introduction

The Open Quantum Safe (OQS) project[1] is an open-source project that aims to support the development and prototyping (of applications) of quantum-resistant cryptography.

OQS consists of the following main lines of work: liboqs, an open source C library for quantum-resistant cryptographic algorithms, and prototype integrations into protocols and applications, including a fork of the widely used OpenSSL library. These tools support research by ourselves and others. To reduce the hurdle for getting started and to aid the uptake and use of these components, our tools are also available as ready-to-use binaries in the form of Docker images and test servers.

In this short note, we provide an update on the Open Quantum Safe project.

2 liboqs

liboqs is an open source C library for quantum-safe cryptographic algorithms. liboqs makes accessible a collection of open-source implementations of quantum-safe key encapsulation mechanism (KEM) and digital signature algorithms through a common API. liboqs builds on Linux, macOS, and Windows, on Intel, AMD, and ARM platforms. Some of the implementations of these algorithms have been directly contributed to liboqs by members of the NIST submission teams; others are incorporated from the PQClean project.

In March 2021 we released version 0.5.0 of liboqs, which contains implementations of all algorithms that have advanced to NIST Round 3 as finalists or alternate candidates (except GeMSS). This version also includes optimized versions of most of these algorithms for Intel platforms, which

[1] `https://openquantumsafe.org`, `https://github.com/open-quantum-safe`

can be compiled to run optimized for a specific architecture or in a portable executable with CPU extensions detected at runtime.

Testing. We run a battery of tests on the cryptographic primitives that we provide: we test the functionality of each primitive on random inputs and on its Known Answer Tests; we test for memory errors and undefined behaviour using LLVM's ASan and UBSan tools; and we test for secret-dependent branching using Valgrind.

The basic functionality tests are run by a continuous integration system, and we are currently exploring ways to run the ASan, UBSan, and Valgrind tests more regularly.

We are also in the process of expanding our tests with coverage-guided fuzzing. We have developed a test harness that allows libFuzzer to manipulate the outputs of random bit sources and random oracles. Using this harness, the fuzzer can quickly explore code paths that are rarely executed in random tests and which are not executed at all by the Known Answer Tests. The fuzzer can also check that different implementations of the same primitive behave identically on these rarely executed code paths.

2.1 Language wrappers

We provide a set of language wrappers that aid in using the liboqs C API safely from within different different programming languages, including C++, Go, Python, C#, Java, and Rust. All wrappers use the same API (or as similar as possible, up to programming language conventions and constraints), regardless of the programming language.

Low-level programming chores such as deallocating memory or zero-ing hot memory (e.g., memory that used to store secret keys) are automatically managed by the wrappers, allowing the programmer to switch focus from the system code to the application code. All wrappers have zero overhead, aside from the intrinsic overhead of calling C code from within the corresponding programming language.

Unit testing suites and continuous integration for liboqs and its language wrappers are provided via CircleCI, AppVeyor, and GitHub's workflows.

3 TLS

We've integrated liboqs into forks of BoringSSL and OpenSSL to provide prototype post-quantum key exchange and authentication in the TLS protocol. With respect to hybrid key exchange, these implementations follow an Internet-Draft currently under consideration by the IETF TLS working group.[2]

Our OpenSSL 1.1.1 fork implements post-quantum and hybrid key exchange and post-quantum public key authentication in TLS 1.3, and also supports post-quantum algorithms in X.509 certificate generation and S/MIME / CMS message handling. This post-quantum-enabled OpenSSL fork can be used in many applications that rely on OpenSSL, and we have successfully done so with the Apache and nginx web servers, HAProxy (an HTTP load balancer), and the curl command-line HTTP client.

The new architecture of the forthcoming OpenSSL 3.0 aims to make easy the integration of new cryptographic mechanisms. We have made available an OpenSSL 3 provider that adds post-quantum key exchange to TLS via a simple binary add-on (shared library). This demonstrates the possibility to add post-quantum cryptography without the need to change the internal logic of the TLS code within OpenSSL.

[2] https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/

Our BoringSSL fork implements post-quantum and hybrid key exchange and post-quantum-only public key authentication in TLS 1.3, and is interoperable with our OpenSSL 1.1.1 fork. We also provide a binary version of the Chromium web browser that uses post-quantum algorithms with the help of the BoringSSL fork.

3.1 TLS interop server

To facilitate simple tests between post-quantum-enabled TLS clients and the OQS implementation, we have made available a public interoperability test server at `https://test.openquantumsafe.org`.

This is an nginx server running the OQS-enabled OpenSSL stack using TLS 1.3, currently offering 3651 ports with all supported combinations of post-quantum certificates and key exchange algorithms, both with simple post-quantum algorithms as well as hybrid PQ+classic cryptography.

We welcome implementers of PQ-enabled TLS implementations to run their implementations against our implementations and interop server, and report any interoperabiltity issues on our Github issue tracker.

4 Other protocols

SSH. We provide a fork of OpenSSH that implements post-quantum and hybrid key exchange and authentication in the SSH protocol.

CMS and S/MIME. Our OpenSSL 1.1.1 fork includes support for post-quantum and hybrid signing operations in the CMS and S/MIME secure mail protocols.

5 Profiling

We recently launched a new dashboard at `https://openquantumsafe.org/benchmarking` visualizing the performance of the implementations at different levels of the software stack on different architectures. Currently, measurements on x86_64 (Intel/AMD) and aarch64 (ARM64) are obtained at the core algorithm level in liboqs and OpenSSL, and at the level of TLS handshakes (both PQ-only and hybrid) in OpenSSL. Also shown are the results of executing the pure reference (C-only) code as well as optimized code versions, and memory consumption figures (stack and heap).

These tests are meant to complement the much more detailed SUPERCOP tests with virtual-machine-based, application-level performance numbers that are more indicative of the performance to be seen in actual (cloud) deployments.

6 Getting started quickly with Docker images

To enable researchers and developers to get started quickly with post-quantum cryptography, we provide ready-to-run Docker images containing PQ-enabled versions of many of the applications described above, including:

- curl
- Apache httpd
- OpenSSH

- nginx
- HAProxy

7 Third-party usage of OQS tools

Since the second NIST PQC workshop, the following uses of OQS tools by third parties have come to our attention:

- Cisco: Post-quantum TLS 1.3 and SSH performance (preliminary results)
- IBM: IBM Cloud delivers quantum-safe cryptography and Hyper Protect Crypto Services to help protect data in the hybrid era
- Microsoft Research: Post-quantum cryptography VPN
- strongSwan: Post-quantum cryptography in IKEv2 using strongSwan

OQS tools were also used in the following research papers:

- PQFabric: A permissioned blockchain secure from both classical and quantum attacks, by Bhargav Das, Amelia Holcomb, Michele Mosca, and Geovandro C. C. F. Pereira. arXiv:2010.06571.
- Post-quantum TLS without handshake signatures, by Peter Schwabe, Douglas Stebila, and Thom Wiggers. *ACM CCS 2020*.
- Benchmarking post-quantum cryptography in TLS, by Christian Paquin, Douglas Stebila, and Goutam Tamvada. *PQCrypto 2020*.
- Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH, by Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. *CoNEXT 2020*.
- Post-quantum authentication in TLS 1.3: A performance study, by Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. *NDSS 2020*.
- Towards quantum-safe VPNs and Internet, by Maran van Heesch, Niels van Adrichem, Thomas Attema, and Thijs Veugen.
- Two PQ signature use-cases: Non-issues, challenges and potential solutions, by Panos Kampanakis and Dimitrios Sikeridis. *7th ETSI/IQC Quantum Safe Cryptography Workshop 2019*.

7.1 IBM Cloud: QSC-enabled Kubernetes ingress controller & QSC-enabled OpenShift router

To enable clients with quantum-safe cryptography (QSC) protected access to clusters in the IBM Cloud, IBM Research implemented a custom ingress controller for IBM Cloud Kubernetes Service (IKS) and a custom router for Red Hat OpenShift on IBM Cloud (managed OpenShift), with QSC provided by Open Quantum Safe. Both enable QSC access to the related clusters in the IBM Cloud. With that, clients can access their clusters benefitting from QSC protected TLS session key establishment, while not having to change anything for the services inside their clusters.

The custom ingress controller for Kubernetes and custom router for ROKS respectively are terminating TLSv1.3 connections from a QSC-enabled application client and feature full backward compatibility for non-QSC operation. This approach enables network connections to use QSC KEM algorithms for session key establishment, and also offer the possibility to use hybrid QSC/non-QSC session key establishment. This hybrid mode of QSC enablement in TLS offers a way to prepare for the future and take a staged transition to QSC operation. The implementation is based on the community NGINX ingress controller and HAProxy ingress controller, with QSC-enabled OpenSSL 1.1.1g libraries provided by Open Quantum Safe.

More information as well as performance testing with concurrent requests can be found under `https://github.com/IBM/qsc-ingress`.

8 Getting involved

All our work is done as open source via our GitHub project. We welcome all types of contributions: new algorithms, source code, code review, bug reports, new integrations, and documentation. Feel free to begin participating on GitHub, or reach out to any of our core team members for more information.

9 Acknowledgements

The Open Quantum Safe project incorporates and adapts a variety of open source cryptographic software. See the individual project pages for lists of contributors and external software. We especially acknowledge algorithm implementations via the PQClean project.

Direct financial support for the development of Open Quantum Safe has been provided by Amazon Web Services, the Canadian Centre for Cyber Security, and Unitary Fund.

Major in-kind contributions of developer time have been made by Amazon Web Services, evolutionQ, IBM Research, and Microsoft Research.

Research projects which developed specific components of OQS have been supported by various research grants, including Australian Research Council (ARC) Discovery Project grant DP130104304, Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2016-05146, NSERC Discovery Accelerator Supplement grant RGPAS-2016-05146, and NSERC Alliance grant ALLRP-556330-20.