OpenSSLNTRU
Faster post-quantum TLS key exchange

August 10th, 2022 — 31st USENIX Security Symposium

Daniel J. Bernstein¹,²  Billy Bob Brumley³
Ming-Shing Chen²  Nicola Tuveri³

authorcontact-opensslntru@box.crypt.to

¹Department of Computer Science, University of Illinois at Chicago, USA

²Ruhr-Universität Bochum, Germany

³Network and Information Security Group (NISEC), Tampere University, Finland
Outline

Motivation
  Post-Quantum Crypto: why? why now?
  NIST PQC

Background
  Previous PQC KEM experiments

OpenSSL NTRU
  key-exchange performance
  TLS software engineering

Conclusions
Post-Quantum Crypto: why?

► Shor’s algorithm (1994) undermines the security of IFP and DLP, the fundamental assumptions at the core of current asymmetric cryptography.

► We currently don’t know of any sufficiently advanced quantum computer to run Shor against recommended parameter sizes, but, especially for confidentiality, threats are already here:

► Data in transit: what if an attacker records current communications, to later decrypt them once quantum attacks are viable?

► Data at rest: what if an entity must store data, being reasonably sure it can stay confidential for the next 20 or more years?
Time and inertia fight against new cryptographic standards

Let’s consider ECC as an example of a recent cryptographic transition

- it was introduced around 1985;
- only around 2000, SECG published standards for ECDSA and ECDH;
- it became popular only during the last decade, and truly pervasive only with TLS 1.3;
- ECC for WebPKI is still moving past the first deployment steps;
- totaling at least 20 to 30 years to go through development, standardization, hardening, garnering trust, adoption, integration, and deployment.

We can’t wait until powerful quantum computers are available to start the process, if we want to prevent or minimize disruption.
NIST PQC

- In 2016, NIST started a Post-Quantum Cryptography (PQC) Standardization Process.
- It has since gone through 3 rounds of selection and consolidation, now progressing to a standardization phase, alongside a 4th round.
- 2 separate operations are being standardized: PQC encryption (KEM), and PQC digital signatures.

This work

- We focused on PQC confidentiality (KEM), not on PQC authentication (signatures). We selected NTRU Prime, one of the KEM “Alternate Candidates” until the end of Round 3.
- We worked on the Streamlined NTRU Prime (sntrup) variant, a small lattice-based KEM, designed to minimize the complexity of a thorough security review. It has been already integrated in OpenSSH, and enabled by default since the 9.0 release.
TLS Integration experiments: Cloudflare&Google (2019)

NTRU-based (lattices)

faster computation
longer cryptographic material

slower computation
shorter cryptographic material

Supersingular Isogeny Key Encapsulation

CECPQ2 = HRSS + X25519
CECPQ2b = SIKE + X25519

https://blog.cloudflare.com/the-tls-post-quantum-experiment/
https://www.imperialviolet.org/2019/10/30/pqsvssl.html
This work

We present OpenSSLNTRU, an improved integration of PQ KEM into TLS 1.3. We improve on the post-quantum portion of CECPQ2 in two (linked) ways:

▶ key-exchange performance,
▶ TLS software engineering.

Table: Cryptographic features of PQ components of CECPQ2 and OpenSSLNTRU.

<table>
<thead>
<tr>
<th></th>
<th>CECPQ2</th>
<th>OpenSSLNTRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryptosystem</td>
<td>ntruhrss701</td>
<td>sntrup761</td>
</tr>
<tr>
<td>key+ciphertext bytes</td>
<td>2276</td>
<td>2197</td>
</tr>
<tr>
<td>keygen cycles</td>
<td>269191</td>
<td>814608→156317</td>
</tr>
<tr>
<td>enc cycles</td>
<td>26510</td>
<td>48892→46914</td>
</tr>
<tr>
<td>dec cycles</td>
<td>63375</td>
<td>59404→56241</td>
</tr>
<tr>
<td>Core-SVP security</td>
<td>$2^{136}$</td>
<td>$2^{153}$</td>
</tr>
<tr>
<td>cyclotomic concerns</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
OpenSSL NTRU: improving Keygen() performance

The bottleneck in sntrup Keygen() is the computation of certain types of inverses.

- **Adopt “Montgomery’s trick”:** two independent inverses $\frac{1}{a}$ and $\frac{1}{b}$ can be computed as $br$ and $ar$, respectively, from a single inversion $r = \frac{1}{ab}$. This can be repeated, e.g., converting 32 inversions into 1 inversion plus 93 multiplications.

- **Batch Keygen()**, so the batch size is large enough for inversion time to mostly disappear, and yet small enough to avoid creating problems with latency, cache misses, etc.

- **New algorithms and software to optimize sntrup multiplications**, since previously they were all “big $\times$ small” but “Montgomery’s trick” requires “big $\times$ big”.
OpenSSLNTRU: TLS software engineering

**TLS 1.3 key agreement using fast PQ KEM**

- **Web browser** epiphany
- **TLS terminator** stunnel
- **Back-end web server**

**OpenSSL patched for PQ KEM in TLS 1.3**

**New ENGINE engntru**

**New KEM libraries**
- **libsntrup761**
- **libsntrup857**

- **Patched to support**
  - private TLS codepoints for sntrup761 and sntrup857 KEM;
  - KEM key agreements alongside DH NIKE;
  - KEM mapped on available PKE+NIKE primitives;
  (required patches are mostly contained in libssl)

- **Provides EVP methods for sntrup761 and sntrup857 as a combination of PKE+NIKE primitives;**
- **Supports keygen batching (thread-local);**
- **Maps PKE+NIKE operations to libsntrup761 or libsntrup857 KEM backends;**
- **Dynamically loadable in libcrypto.**

**Rapidly evolving software ecosystem**

**Our patches and new software**

- **Patched to support**
  - private TLS codepoints for sntrup761 and sntrup857 KEM;
  - KEM key agreements alongside DH NIKE;
  - KEM mapped on available PKE+NIKE primitives;
  (required patches are mostly contained in libssl)

- **Provides EVP methods for sntrup761 and sntrup857 as a combination of PKE+NIKE primitives;**
- **Supports keygen batching (thread-local);**
- **Maps PKE+NIKE operations to libsntrup761 or libsntrup857 KEM backends;**
- **Dynamically loadable in libcrypto.**

**New KEM libraries**
- **libsntrup761**
- **libsntrup857**

- **Provide new optimized implementations of sntrup761 and sntrup857 operations;**
- **Contain the actual cryptographic functionality.**
OpenSSLNTRU: TLS macro-benchmark

Figure: Cumulative distributions of handshake performance under different cryptosystems in a local network.
Conclusion

- Faster optimized implementation for various sntrup parameter sets, including huge **batching** gains on Keygen().
- **Transparent integration** for existing applications via an OpenSSL ENGINE.
- Decoupling OpenSSL from fast-paced development of optimized PQ implementations, and vice-versa the latter from data types and interfaces specific to OpenSSL.
- Check the paper and its appendices for more math details, micro- and macro-benchmarks, comparisons, and more.
- Try the open source artifact at https://opensslntru.cr.yp.to/.
Questions?

THANKS!

KIITOS!

GRAZIE!

 OpenSSLNTRU: Faster post-quantum TLS key exchange

https://opensslntru.cr.yp.to/

authorcontact-opensslntru@box.cr.yp.to