Tampere University

OpenSSLNTRU

Faster post-quantum TLS key exchange

August 10th, 2022 — 31st USENIX Security Symposium

Daniel J. Bernstein[1,2] Billy Bob Brumley[3]
Ming-Shing Chen[2] Nicola Tuveri[3]

authorcontact-opensslntru@box.cr.yp.to

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

[2]Ruhr-Universität Bochum, Germany

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

Outline

- Motivation
  - Post-Quantum Crypto: why? why now?
  - NIST PQC
- Background
  - Previous PQC KEM experiments
- OpenSSLNTRU
  - 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 ara 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.

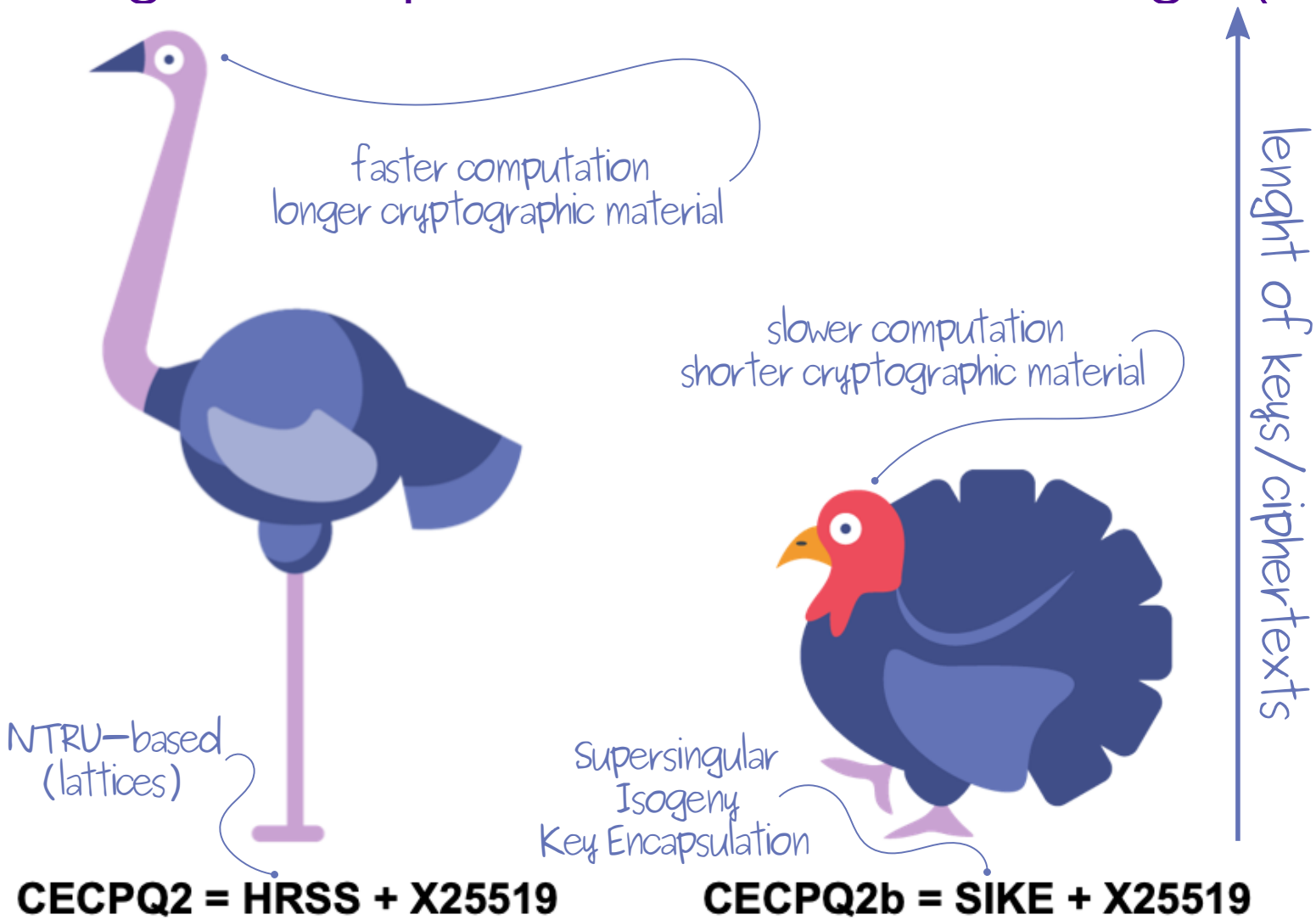
NIST PQC

- 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 $\mathrm{OpenSSH}$, and enabled by default since the 9.0 release.

TLS Integration experiments: Cloudflare&Google (2019)



https://blog.cloudflare.com/towards-post-quantum-cryptography-in-tls/

https://blog.cloudflare.com/the-tls-post-quantum-experiment/

https://www.imperialviolet.org/2019/10/30/pqsivssl.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.

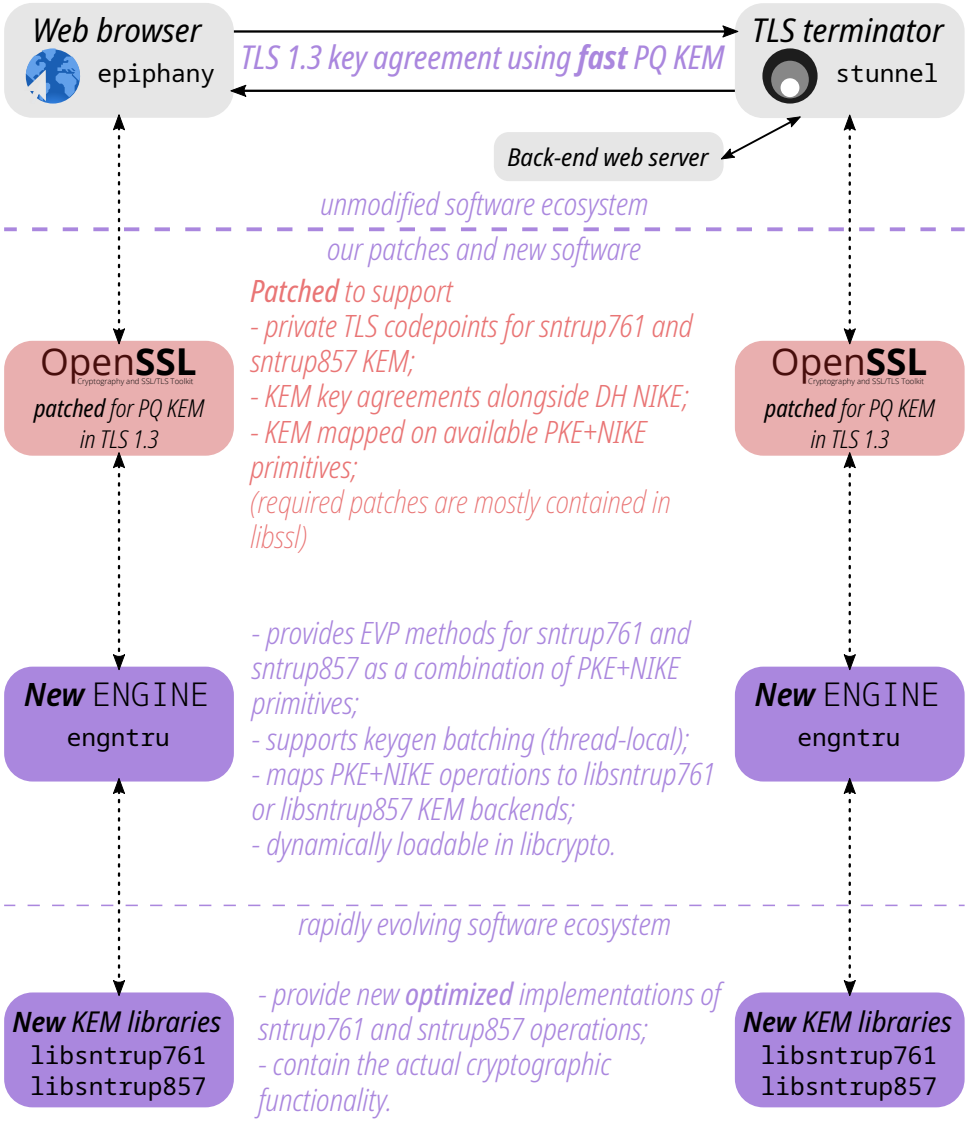
	CECPQ2	OpenSSLNTRU
cryptosystem	ntruhrss701	sntrup761
key+ciphertext bytes	2276	2197
keygen cycles	269191	814608→156317
enc cycles	26510	48892→46914
dec cycles	63375	59404→56241
Core-SVP security	2^{136}	2^{153}
cyclotomic concerns	yes	no

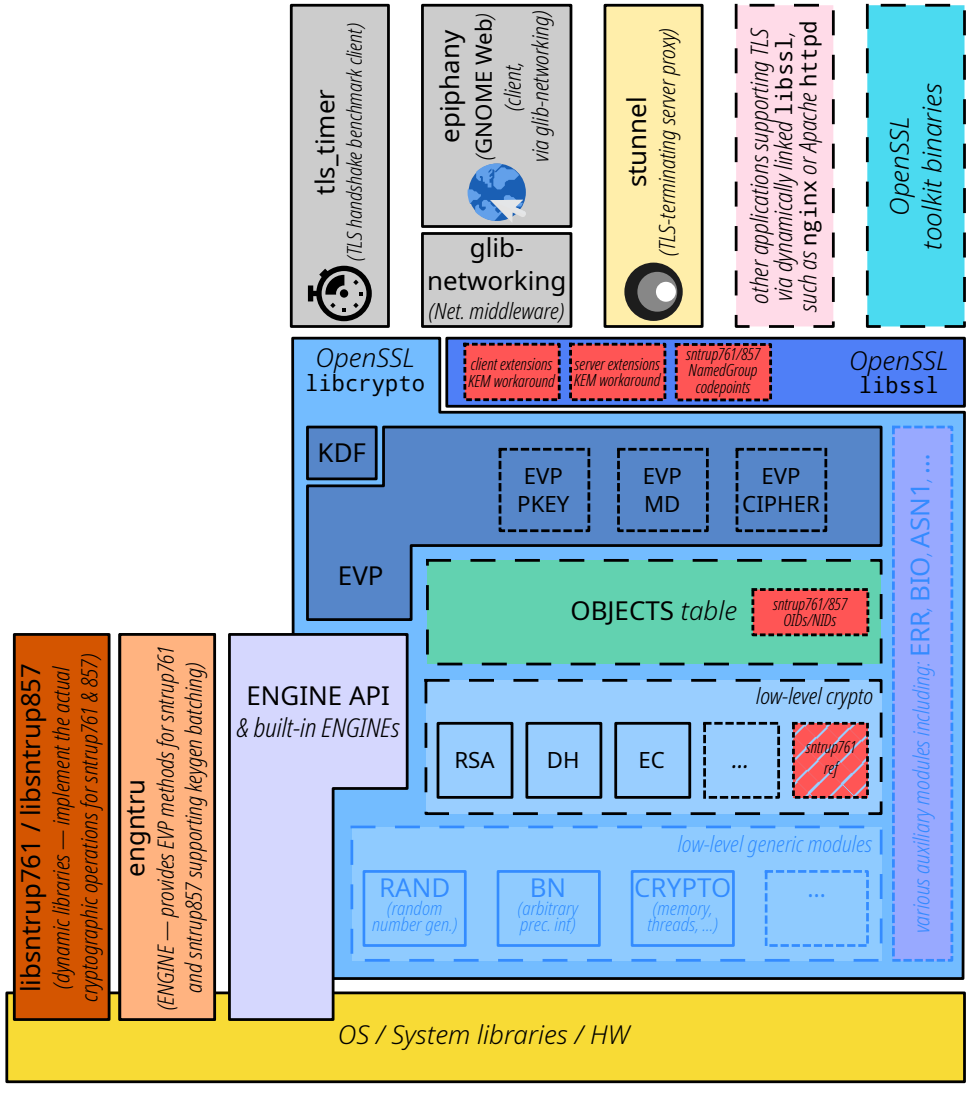
OpenSSLNTRU: improving Keygen() performance

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*×*small*" but "Montgomery's trick" requires "*big*×*big*".

OpenSSLNTRU: TLS software engineering





OpenSSLNTRU: TLS macro-benchmark

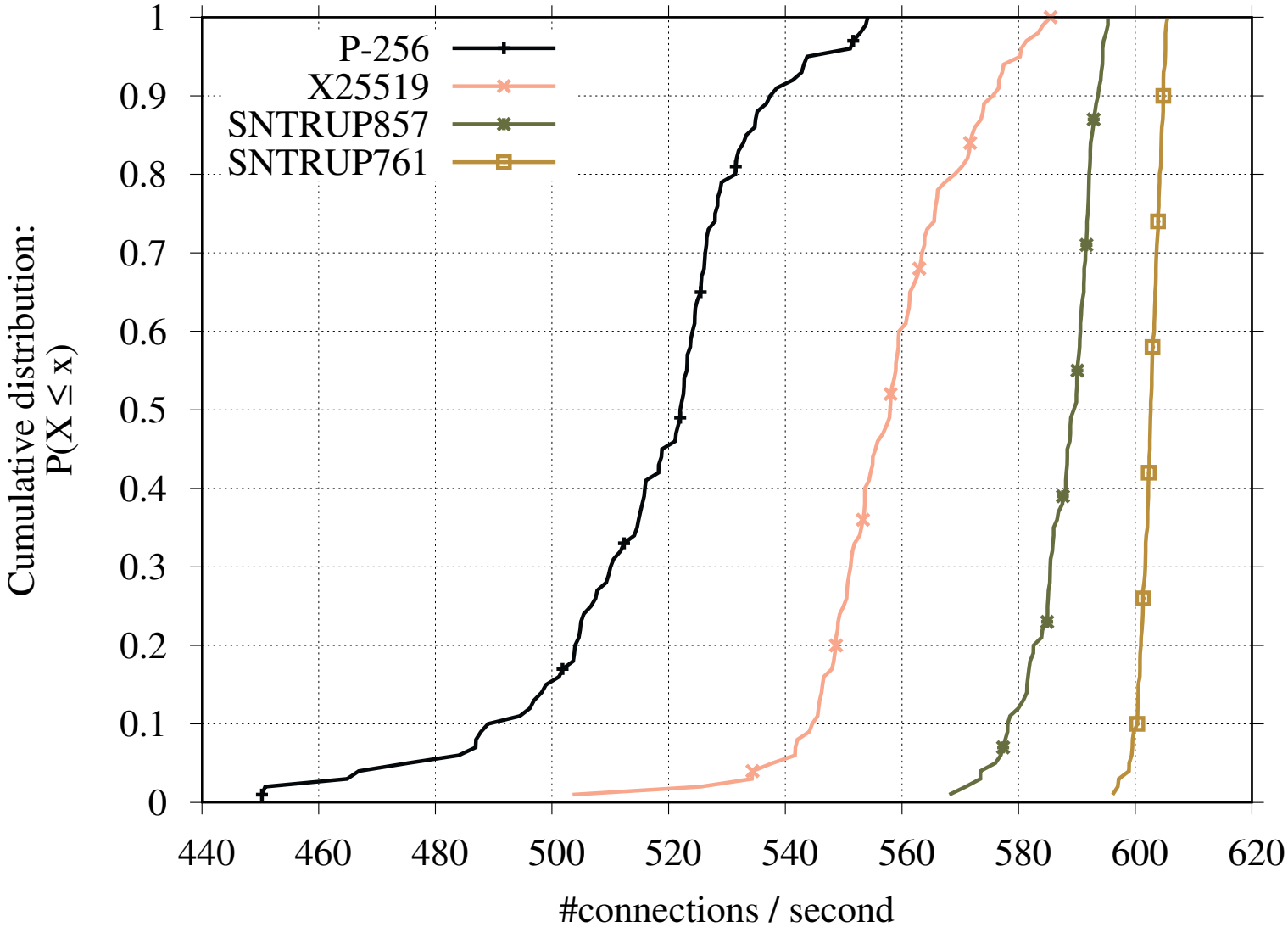


Figure: Cumulative distributions of handshake performance under different cryptosystems in a local network.

Conclusion

- Faster optimized implementation for various sntrup parameter sets, including huge batching gains on $\mathrm{Keygen}()$.
- Transparent integration for existing applications via an OpenSSL ENGINE.
- Decoupling OpenSSL from fast-paced development of optimized PQ implementations, and viceversa 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 $\mathrm{https://opensslntru.cr.yp.to/}$.

Questions?

Thanks!
Kiitos!
Grazie!

OpenSSLNTRU: Faster post-quantum TLS key exchange

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

authorcontact-opensslntru@box.cr.yp.to