## F Artifact Appendix

### F.1 Abstract

This demo was

- announced 2020.04.16 on the pqc-forum mailing list,
- updated 2020.04.23 from OpenSSL `1.1.1f` to OpenSSL `1.1.1g`,
- updated 2021.06.08 from OpenSSL `1.1.1g` to OpenSSL `1.1.1k`, including additional support for sntrup857,
- updated 2021.09.30 from OpenSSL `1.1.1k` to OpenSSL `1.1.1l`, alongside an update of the instructions to use `stunnel 5.60` and `glib-networking 2.60.4`, and
- updated 2021.11.02 to cover usage of `tls_timer` and suggestions regarding its use for experiments, and
- updated 2021.12.14 from OpenSSL `1.1.1l` to OpenSSL `1.1.1m`.

Our patches work for versions of OpenSSL from `1.1.1f` to `1.1.1m`.

This is a demo of OpenSSLNTRU web browsing taking just 156317 Haswell cycles to generate a new one-time `sntrup761` public key for each TLS 1.3 session. This demo uses: (i) the Gnome web browser (client) and `stunnel` (server) using (ii) a patched version of OpenSSL `1.1.1m` using (iii) a new OpenSSL `ENGINE` using (iv) a fast new `sntrup761` library.

The TLS 1.3 integration in OpenSSLNTRU uses the same basic data flow as the CECPQ2 experiment carried out by Google and Cloudflare. Compared to the cryptography in CECPQ2, the cryptography in OpenSSLNTRU has a higher security level and better performance. Furthermore, OpenSSLNTRU's new software layers decouple the fast-moving post-quantum software ecosystem from the TLS software ecosystem. OpenSSLNTRU also supports a second NTRU Prime parameter set, `sntrup857`, optimizing computation costs at an even higher security level.

### F.2 Artifact check-list (meta-information)

- **How much time is needed to prepare workflow (approximately)?:** 60 min
- **How much time is needed to complete experiments (approximately)?:** 5–60 min
- **Publicly available?:** Y
- **Archived (provide DOI)?:** 10.5281/zenodo.5833729

### F.3 Description

#### F.3.1 How to access

Visit `https://opensslntru.cr.yp.to/demo.html`.

Additionally, we provide an archived version on Zenodo. The instructions in this appendix apply to the latter using, in place of the online URLs at `https://opensslntru.cr.yp.to/`, the contents extracted from the downloaded archive.

#### F.3.2 Hardware dependencies

1. AVX2 support

#### F.3.3 Software dependencies

1. Linux
2. OpenSSL `1.1.1`

### F.4 Installation

`https://opensslntru.cr.yp.to/demo.html`

### F.5 Evaluation and expected results

We claim the artifact at `https://opensslntru.cr.yp.to/demo.html` reproduces two of the paper claims.

#### F.5.1 Reaching applications transparently

Following the provided instructions, the artifact allows to reproduce Section 4.3. By the end of the demo, you should achieve:

1. Setting up a TLS server with a custom TLS 1.3 cipher suite supporting `sntrup`.
2. Setting up a TLS client with a custom TLS 1.3 cipher suite supporting `sntrup`.
3. The user sees this upon issuing the last command `epiphany https://test761.cr.yp.to` as listed in the demo instructions.

The server side is optional, depending on if you want to talk to your own webserver or `https://test761.cr.yp.to`.

#### F.5.2 Macrobenchmarks: TLS handshakes

Additionally, the last part of the artifact covers the use of `tls_timer` to measure the wall-clock execution time of sequential TLS connections using different TLS groups, as described in Section 4.4.

Using `tls_timer` you can evaluate that

1. our specialized batch implementation (provided via `engNTRU` by `libsntrup761`) is faster than the reference code included in the optional patch to embed support for `sntrup761` operations in `libcrypto`;
2. with the caveats mentioned in Section 4.4, we achieve new records, in terms of computational costs, when compared with `X25519` and `NIST P-256`, the fastest pre-quantum implementations of TLS 1.3 key-exchange groups included in OpenSSL `1.1.1`, while providing higher pre-quantum security levels and much higher post-quantum security levels against all known attacks.

## F.6 Notes

To reproduce the results summarized in Figure 5 in terms of absolute values, you would have to replicate the setup described in terms of hardware in footnote 6, and also take care of setting up both systems as detailed in the paragraph directly above the footnote to avoid biases due to CPU contention. It should also be noted that 100 experiments for each group, each performing 8192 connections, will require several hours, and that for the entire duration of the experiment you should ensure low network traffic and that no other processes (automated updates and other scheduled processes in particular) are executed on the machines running the experiments.

A simpler alternative, that would prove consistent with the results presented in Figure 5 in terms of sorting the groups according to the average connections per second, but not necessarily in absolute values, would be to install the server side and the client side of the demo on the same host, and then use `tls_timer` over the loopback interface, lowering the number of sequential connections (e.g., to 1024) to reduce the execution time of each experiment.

In any case, it is important to take care of the details listed in Section 4.4 regarding disabling frequency scaling, Turbo boost, concurrent services, and scheduled processes, isolating physical cores exclusively to each of the 3 processes (i.e., `tls_timer`, `stunnel`, and `apache2`) involved in each experiment run, and disabling/reducing logging to console or files, in order to minimize external causes of noise and achieve consistent results.