Post-Quantum TLS without handshake signatures

Sofía Celi\textsuperscript{1}, Armando Faz Hernández\textsuperscript{1}, Peter Schwabe\textsuperscript{2,4}, Douglas Stebila\textsuperscript{3}, Thom Wiggers\textsuperscript{4}

\textsuperscript{1} Cloudflare \quad \textsuperscript{2} MPI Security and Privacy
\textsuperscript{3} University of Waterloo \quad \textsuperscript{4} Radboud University
TLS 1.3 Handshake

Key exchange: Diffie–Hellman
Authentication: Signatures
Post-Quantum TLS 1.3 Handshake

- Key exchange: Post-Quantum Key-Encapsulation Mechanisms
- Authentication: Post-Quantum Signatures
Problem

Post-Quantum signatures are...
Problem

Post-Quantum signatures are... 
- quite a bit bigger than KEMs
- quite a bit slower than KEMs
- quite a bit of extra code
Use PQ KEMs for authentication instead
**KEM**

**Definition (Key Encapsulation Mechanism (KEM))**

- (pk, sk) ← KEM.Keygen()
- (ss, ct) ← KEM.Encapsulate(pk)
- ss ← KEM.Decapsulate(ct, sk)

**Example**

To authenticate **Douglas** to **Peter**

Peter

(ss, ct) ← KEM.Encapsulate(pk_{Douglas})

ct

ss ← KEM.Decapsulate(ct, sk_{Douglas})

K ← KDF(ss)

MAC_{K}(⋯)

Douglas

K ← KDF(ss)
Definition (Key Encapsulation Mechanism (KEM))

- (pk, sk) ← KEM.Keygen()
- (ss, ct) ← KEM.Encapsulate(pk)
- ss ← KEM.Decapsulate(ct, sk)

Example

To authenticate Douglas to Peter

\[
\begin{align*}
\text{Peter} & \quad \text{Douglas} \\
(ss, ct) & \leftarrow \text{KEM.Encapsulate}(pk_{\text{Douglas}}) \\
ct & \\
ss & \leftarrow \text{KEM.Decapsulate}(ct, sk_{\text{Douglas}}) \\
K & \leftarrow \text{KDF}(ss) \\
\text{MAC}_K(\cdots) & \leftarrow K \leftarrow \text{KDF}(ss)
\end{align*}
\]
Definition (Key Encapsulation Mechanism (KEM))

- \((pk, sk) \leftarrow \text{KEM.Keygen}()\)
- \((ss, ct) \leftarrow \text{KEM.Encapsulate}(pk)\)
- \(ss \leftarrow \text{KEM.Decapsulate}(ct, sk)\)

Example

To authenticate Douglas to Peter

\[
\begin{align*}
\text{Peter} &\quad \text{Douglas} \\
(ss, ct) &\leftarrow \text{KEM.Encapsulate}(pk_{Douglas}) \\
ct & \quad K \leftarrow \text{KDF}(ss) \\
\quad ss &\leftarrow \text{KEM.Decapsulate}(ct, sk_{Douglas}) \\
K & \leftarrow \text{KDF}(ss) \\
\text{MAC}_K(\cdots) &\quad T_{thom}\end{align*}
\]
**KEM**

**Definition (Key Encapsulation Mechanism (KEM))**

- \( (pk, sk) \leftarrow \text{KEM.Keygen}() \)
- \( (ss, ct) \leftarrow \text{KEM.Encapsulate}(pk) \)
- \( ss \leftarrow \text{KEM.Decapsulate}(ct, sk) \)

**Example**

To authenticate **Douglas** to **Peter**

\[
\begin{align*}
\text{Peter} & \quad \text{Douglas} \\
(ss, ct) & \leftarrow \text{KEM.Encapsulate}(pk_{Douglas}) \\
ct & \\
ss & \leftarrow \text{KEM.Decapsulate}(ct, sk_{Douglas}) \\
K & \leftarrow \text{KDF}(ss) \\
K & \leftarrow \text{KDF}(ss) \\
\text{MAC}_K(\cdots) & \\
\end{align*}
\]
KEM authentication in TLS

Problem

- In TLS, the client doesn’t already have the public key of the server!
- To put this in TLS 1.3, we need an extra roundtrip!
- TLS 1.3 tried very hard to finish the handshake a single roundtrip.
KEM authentication in TLS

Problem
- In TLS, the client doesn’t already have the public key of the server!
- To put this in TLS 1.3, we need an extra roundtrip!
- TLS 1.3 tried very hard to finish the handshake a single roundtrip.

Solution
Implicitly authenticated key exchange: the client encapsulates to the server’s long-term public key but does not wait until they get the MAC before sending data!
KEM authentication in TLS

Problem

- In TLS, the client doesn’t already have the public key of the server!
- To put this in TLS 1.3, we need an extra roundtrip!
- TLS 1.3 tried very hard to finish the handshake a single roundtrip.

Solution

Implicitly authenticated key exchange: the client encapsulates to the server’s long-term public key *but does not wait until they get the MAC before sending data!*

Seen in HMQV (DH), BCGP09 & FSXY12 (KEMs), . . . , Signal, Noise, Wireguard, . . .
KEMTLS

- Ephemeral key exchange
- Static-KEM authentication
- Combine shared secrets
- Allow client to send application data before receiving server’s key confirmation
Ephemeral key exchange

Static-KEM authentication

Combine shared secrets

Allow client to send application data before receiving server's key confirmation

Thom Wiggers

PQ TLS without handshake signatures

RWC 2021
Thom Wiggers

PQ TLS without handshake signatures

RWC 2021
KEMTLS

- Ephemeral key exchange
- Static-KEM authentication
- Combine shared secrets
- Allow client to send application data before receiving server’s key confirmation

Thom Wiggers
PQ TLS without handshake signatures
RWC 2021
## Choosing algorithms

### Ephemeral Key Exchange

- ~ IND-CCA KEM
- Ideally fast with small pk + ct

### KEM for server authentication

- IND-CCA KEM
- Ideally fast with small pk + ct
## Choosing algorithms

<table>
<thead>
<tr>
<th><strong>Ephemeral Key Exchange</strong></th>
<th><strong>KEM for server authentication</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>~ IND-CCA KEM</td>
<td>IND-CCA KEM</td>
</tr>
<tr>
<td>Ideally fast with small pk + ct</td>
<td>Ideally fast with small pk + ct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Root CA certificate</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Already present on client</td>
</tr>
<tr>
<td>Only care about signature size</td>
</tr>
</tbody>
</table>
Choosing algorithms

Ephemeral Key Exchange
- ~ IND-CCA KEM
- Ideally fast with small pk + ct

KEM for server authentication
- IND-CCA KEM
- Ideally fast with small pk + ct

Intermediate CA certificate
- Small public key + signature size

Root CA certificate
- Already present on client
- Only care about signature size
Comparison

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Dilithium
Falcon
GeMSS
Kyber
NTRU
SIKE
XMSS$_{MT}$

---

1Rustls with AVX2 implementations. Emulated network: latency 31.1 ms, 1000 Mbps, no packet loss. Average of 100000 iterations.
Observations on emulated experiments

- Size-optimized KEMTLS requires \(<\frac{1}{2}\) communication of size-optimized PQ signed-KEM
- Speed-optimized KEMTLS uses 90\% fewer server CPU cycles and still reduces communication
  - NTRU KEX (27 µs) 10x faster than Falcon signing (254 µs)
- No extra round trips required until client starts sending application data
- Smaller trusted code base (no signature generation on client/server)
Real world measurements
ft. Cloudflare
Experimenting in the real world

- Experimental implementation in Go standard library TLS
- Branch https://github.com/cloudflare/go/tree/cf-pq-kemtls
- Based on Delegated Credentials (draft-ietf-tls-subcerts)
- Also implements client authentication
Experimenting in the real world

- Experimental implementation in Go standard library TLS
- Branch https://github.com/cloudflare/go/tree/cf-pq-kemtls
- Based on Delegated Credentials (draft-ietf-tls-subcerts)
- Also implements client authentication

We intend to do measurements on real networks

- between Cloudflare DCs
- Measure more aspects than just handshake time
- ???

Hope to report more soon
Post-Quantum TLS without Handshake signatures

- Implicit authentication via KEMs
- Preserve client ability to do request after 1RTT
- Saves bytes on the wire and server CPU cycles
- ACM CCS 2020 doi: 10.1145/3372297.3423350
- Full version with proofs: ia.cr/2020/534

Cloudflare is helping us investigate KEMTLS in the real world. Experimental implementation in branch cf-pq-kemtls at github.com/cloudflare/go. Hopefully results soon™ — keep an eye on the Cloudflare Research Blog.
Appendix
FAQ

- Client authentication?
  - We provide a sketch in Appendix D, but mostly leave it for future work.
  - Naive way does require a full additional round-trip.

- What about TLS 1.3 0-RTT?
  - 0-RTT is for resumption. You can do the same thing in KEMTLS.
  - We see opportunities for more efficient handshakes when resuming or in scenarios with pre-distributed KEM public keys.

- Server can’t send application data in its first TLS flow. Will that break HTTP/3 where the server sends a SETTINGS frame?
  - Could be included in an extension as a server-side variant of ALPN.

- How do you do certificate lifecycle management (issuance, revocation) with KEM public keys?
  - At first glance many of these issues seem non-trivial: currently these assume the public key can be used for signatures in some way.
  - Another good direction for future work.
## Communications sizes

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>KEX (pk+ct)</th>
<th>Excluding intermediate CA certificate (incl. int. CA cert.)</th>
<th>Including intermediate CA certificate (incl. int. CA cert.)</th>
<th>Sum TCP payloads of TLS HS (incl. int. CA cert.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TLS 1.3</strong></td>
<td><strong>Errr</strong></td>
<td>ECDH (X25519) 64</td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Min. incl. int. CA cert.</strong></td>
<td><strong>Sfxg</strong></td>
<td>SIKE 405</td>
<td>Falcon 690</td>
<td>Falcon 897</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Min. excl. int. CA cert.</strong></td>
<td><strong>Sfgg</strong></td>
<td>SIKE 405</td>
<td>Falcon 690</td>
<td>Falcon 897</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Assumption: MLWE+MSIS</strong></td>
<td><strong>Kddd</strong></td>
<td>Kyber 1536</td>
<td>Dlithium 2044</td>
<td>Dlithium 1184</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Assumption: Ntru</strong></td>
<td><strong>Nff</strong></td>
<td>Ntru 1398</td>
<td>Falcon 690</td>
<td>Falcon 897</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Min. incl. int. CA cert.</strong></td>
<td><strong>Ssxg</strong></td>
<td>SIKE 405</td>
<td>SIKE 209</td>
<td>SIKE 196</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Min. excl. int. CA cert.</strong></td>
<td><strong>Ssgg</strong></td>
<td>SIKE 405</td>
<td>SIKE 209</td>
<td>SIKE 196</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Assumption: MLWE+MSIS</strong></td>
<td><strong>Kkdd</strong></td>
<td>Kyber 1536</td>
<td>Kyber 736</td>
<td>Kyber 800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td><strong>Assumption: Ntru</strong></td>
<td><strong>Nff</strong></td>
<td>Ntru 1398</td>
<td>Ntru 699</td>
<td>Ntru 699</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 256</td>
<td>RSA-2048 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RSA-2048 272</td>
<td>RSA-2048 256</td>
</tr>
</tbody>
</table>

Thom Wiggers  PQ TLS without handshake signatures  RWC 2021
## Time measurements

### Computation time for asymmetric crypto

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
</tr>
<tr>
<td>ERRR</td>
<td>0.134</td>
<td>0.629</td>
</tr>
<tr>
<td>SFXG</td>
<td>40.058</td>
<td>21.676</td>
</tr>
<tr>
<td>SFGG</td>
<td>34.104</td>
<td>21.676</td>
</tr>
<tr>
<td>KDDD</td>
<td>0.080</td>
<td>0.087</td>
</tr>
<tr>
<td>NFFF</td>
<td>0.141</td>
<td>0.254</td>
</tr>
</tbody>
</table>

### Handshake time (31.1 ms latency, 1000 Mbps bandwidth)

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRR</td>
<td>66.4</td>
<td>97.6</td>
</tr>
<tr>
<td>SFXG</td>
<td>165.8</td>
<td>196.9</td>
</tr>
<tr>
<td>SFGG</td>
<td>154.9</td>
<td>186.0</td>
</tr>
<tr>
<td>KDDD</td>
<td>64.3</td>
<td>95.5</td>
</tr>
<tr>
<td>NFFF</td>
<td>65.1</td>
<td>96.3</td>
</tr>
</tbody>
</table>

### Handshake time (195.6 ms latency, 10 Mbps bandwidth)

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRR</td>
<td>305.8</td>
<td>593.3</td>
</tr>
<tr>
<td>SFXG</td>
<td>482.1</td>
<td>678.4</td>
</tr>
<tr>
<td>SFGG</td>
<td>473.7</td>
<td>669.8</td>
</tr>
<tr>
<td>KDDD</td>
<td>411.6</td>
<td>852.4</td>
</tr>
<tr>
<td>NFFF</td>
<td>398.1</td>
<td>662.2</td>
</tr>
</tbody>
</table>

### KMTLS

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSXG</td>
<td>202.1</td>
<td>268.8</td>
</tr>
<tr>
<td>SSGG</td>
<td>190.4</td>
<td>256.6</td>
</tr>
<tr>
<td>KKDD</td>
<td>63.4</td>
<td>95.0</td>
</tr>
<tr>
<td>NNFF</td>
<td>63.6</td>
<td>95.2</td>
</tr>
</tbody>
</table>

### KPMTLS

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSXG</td>
<td>505.8</td>
<td>732.0</td>
</tr>
<tr>
<td>SSGG</td>
<td>496.8</td>
<td>723.0</td>
</tr>
<tr>
<td>KKDD</td>
<td>399.2</td>
<td>835.1</td>
</tr>
<tr>
<td>NNFF</td>
<td>396.2</td>
<td>593.4</td>
</tr>
</tbody>
</table>

---

Thom Wiggers  | PQ TLS without handshake signatures  | RWC 2021
<table>
<thead>
<tr>
<th></th>
<th>Computation time for asymmetric crypto</th>
<th>Handshake time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Excl. int. CA cert.</td>
<td>Incl. int. CA cert.</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
</tr>
<tr>
<td><strong>ERRR</strong></td>
<td>0.134</td>
<td>0.629</td>
</tr>
<tr>
<td><strong>SFXG</strong></td>
<td>40.058</td>
<td>21.676</td>
</tr>
<tr>
<td><strong>SFGG</strong></td>
<td>34.104</td>
<td>21.676</td>
</tr>
<tr>
<td><strong>KDDD</strong></td>
<td>0.080</td>
<td>0.087</td>
</tr>
<tr>
<td><strong>NFFF</strong></td>
<td>0.141</td>
<td>0.254</td>
</tr>
<tr>
<td><strong>KEMTLS</strong></td>
<td><strong>SSXG</strong></td>
<td>61.456</td>
</tr>
<tr>
<td></td>
<td><strong>SSGG</strong></td>
<td>55.503</td>
</tr>
<tr>
<td></td>
<td><strong>KKDD</strong></td>
<td>0.060</td>
</tr>
<tr>
<td></td>
<td><strong>NNFF</strong></td>
<td>0.118</td>
</tr>
<tr>
<td>Proto</td>
<td>Handshake time (31.1 ms latency, 1000 Mbps bandwidth)</td>
<td>Excl. int. CA cert.</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>629</td>
<td>66.4 97.6 35.4</td>
<td>66.6 97.8 35.6</td>
</tr>
<tr>
<td>676</td>
<td>165.8 196.9 134.0</td>
<td>166.2 197.3 134.4</td>
</tr>
<tr>
<td>676</td>
<td>154.9 186.0 123.1</td>
<td>259.0 290.2 227.1</td>
</tr>
<tr>
<td>7087</td>
<td>64.3 95.5 33.3</td>
<td>64.8 96.0 33.8</td>
</tr>
<tr>
<td>254</td>
<td>65.1 96.3 34.1</td>
<td>65.6 96.9 34.7</td>
</tr>
<tr>
<td>712</td>
<td>202.1 268.8 205.6</td>
<td>202.3 269.1 205.9</td>
</tr>
<tr>
<td>712</td>
<td>190.4 256.6 193.4</td>
<td>293.3 359.5 296.3</td>
</tr>
<tr>
<td>7021</td>
<td>63.4 95.0 32.7</td>
<td>63.9 95.5 33.2</td>
</tr>
<tr>
<td>7027</td>
<td>63.6 95.2 32.9</td>
<td>64.2 95.8 33.5</td>
</tr>
</tbody>
</table>
### Handshake time (195.6 ms latency, 10 Mbps bandwidth)

<table>
<thead>
<tr>
<th></th>
<th>Excl. int. CA cert.</th>
<th>Incl. int. CA cert.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Server</td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>sent req.</td>
<td>recv. resp.</td>
</tr>
<tr>
<td>35.6</td>
<td>397.1</td>
<td>593.3</td>
</tr>
<tr>
<td>134.4</td>
<td>482.1</td>
<td>678.4</td>
</tr>
<tr>
<td>227.1</td>
<td>473.7</td>
<td>669.8</td>
</tr>
<tr>
<td>33.8</td>
<td>411.6</td>
<td>852.4</td>
</tr>
<tr>
<td>34.7</td>
<td>398.1</td>
<td>662.2</td>
</tr>
<tr>
<td>205.9</td>
<td>505.8</td>
<td>732.0</td>
</tr>
<tr>
<td>296.3</td>
<td>496.8</td>
<td>723.0</td>
</tr>
<tr>
<td>33.2</td>
<td>399.2</td>
<td>835.1</td>
</tr>
<tr>
<td>33.5</td>
<td>396.2</td>
<td>593.4</td>
</tr>
</tbody>
</table>
KEMTLS in more detail

---

Phase 1: ephemeral key exchange

- \((pk_e, sk_e) \leftarrow \text{KEM}_{e}.\text{Keygen}()\)

Client Hello:
- \(pk_e, r_c \leftarrow \{0, 1\}^{256}\)
- supported algs.

\(ES \leftarrow \text{HKDF}.\text{Extract}(0, 0)\)
\(dES \leftarrow \text{HKDF}.\text{Extract}(ES, "derived", \emptyset)\)

\((ss_e, ct_e) \leftarrow \text{KEM}_e.\text{Encapsulate}(pk_e)\)

Server Hello:
- \(ct_e, r_s \leftarrow \{0, 1\}^{256}\), selected algs.

\(ss_e \leftarrow \text{KEM}_e.\text{Decapsulate}(ct_e, sk_e)\)

HS \leftarrow \text{HKDF}.\text{Extract}(dES, ss_e)

- accept CHTS \leftarrow \text{HKDF}.\text{Expand}(HS, "c hs traffic", CH..SH)
- stage 1
- accept SHTS \leftarrow \text{HKDF}.\text{Expand}(HS, "s hs traffic", CH..SH)
- stage 2

\(dHS \leftarrow \text{HKDF}.\text{Expand}(HS, "derived", \emptyset)\)

\{EncryptedExtensions\}_stage2
\{ServerCertificate\}_stage2: \text{cert}[pk_S], int. CA cert.

Phase 2: Implicitly authenticated key exchange

\((ss_S, ct_S) \leftarrow \text{KEM}_s.\text{Encapsulate}(pk_S)\)

\{ClientKemCiphertext\}_stage1: ct_S

\(ss_S \leftarrow \text{KEM}_s.\text{Decapsulate}(ct_S, sk_S)\)

AHS \leftarrow \text{HKDF}.\text{Extract}(dHS, ss_S)

- accept CAHTS \leftarrow \text{HKDF}.\text{Expand}(AHS, "c ahs traffic", CH..SH)
- stage 3

\(dAHS \leftarrow \text{HKDF}.\text{Expand}(AHS, "derived", \emptyset)\)

\{ClientFinished\}_stage3: CF \leftarrow \text{HMAC}(f_k_c, CH..CF)

- abort if \(CF \neq \text{HMAC}(f_k_c, CH..CF)\)

Phase 3: Confirmation / explicit authentication

\{ServerFinished\}_stage5: SF \leftarrow \text{HMAC}(f_k_s, CH..CF)

- abort if \(SF \neq \text{HMAC}(f_k_s, CH..SF)\)

---

Thom Wiggers  PQ TLS without handshake signatures  RWC 2021
Sending application data before FIN

The client sends data before receiving ServerFinished.

*Does this mean you can downgrade to weak crypto to aid future (quantum) decryption?*
Sending application data before FIN

The client sends data before receiving `ServerFinished`.

*Does this mean you can downgrade to weak crypto to aid future (quantum) decryption?*

- Can’t downgrade to signed TLS 1.3
  - TLS 1.3 handshake signature stops the attack
Sending application data before FIN

The client sends data before receiving `ServerFinished`.

*Does this mean you can downgrade to weak crypto to aid future (quantum) decryption?*

- Can’t downgrade to signed TLS 1.3
  - TLS 1.3 handshake signature stops the attack
- Active adversary might try to downgrade first client-to-server flow
  - Only to whatever algorithms the client advertised in `ClientHello`
    - Don’t support pre-quantum in KEMTLS
Sending application data before FIN

The client sends data before receiving `ServerFinished`.

*Does this mean you can downgrade to weak crypto to aid future (quantum) decryption?*

- Can’t downgrade to signed TLS 1.3
  - TLS 1.3 handshake signature stops the attack
- Active adversary might try to downgrade first client-to-server flow
- Only to whatever algorithms the client advertised in `ClientHello`
  - Don’t support pre-quantum in KEMTLS
- The handshake will no longer successfully complete
  - `ServerFinished` reveals the downgrade unless MAC, KEM, KDF or hash are broken *at time of attack*
  - Once `SF` is received: retroactive full downgrade resilience
  - You also get upgraded from weak to full forward secrecy at this stage
TLS 1.3 Handshake

- **Key exchange:** Diffie–Hellman
- **Authentication:** Signatures
Post-Quantum TLS 1.3 Handshake

Key exchange: Post-Quantum Key-Encapsulation Mechanisms

Authentication: Post-Quantum Signatures
KEMTLS

- Ephemeral key exchange
- Static-KEM authentication
- Combine shared secrets
- Allow client to send application data before receiving server’s key confirmation
Post-Quantum TLS without Handshake signatures

- Implicit authentication via KEMs
- Preserve client ability to do request after 1RTT
- Saves bytes on the wire and server CPU cycles
- ACM CCS 2020 doi: 10.1145/3372297.3423350
- Full version with proofs: ia.cr/2020/534

Cloudflare is helping us investigate KEMTLS in the real world. Experimental implementation in branch cf-pq-kemtls at github.com/cloudflare/go. Hopefully results soon™ — keep an eye on the Cloudflare Research Blog.