AuthKEM

draft-celi-wiggers-tls-authkem-00

Sofía Celi, Thom Wiggers

Thom Wiggers
Radboud University

Sofía Celi
Cloudflare
What is authentication, really?
- Proving who you are
- Proving possession of a private key

Authentication in TLS
- Signature with certificate key in a *cert-based* context
- Knowledge of PSK

**draft-celi-wiggers-tls-authkem**: Authentication via Key Encapsulation Mechanisms (KEMs)
Authentication via KEM

KEM:

- $\text{enc, ss} \leftarrow \text{Encap}(pk_B)$
- $\text{ss} \leftarrow \text{Decap}(\text{enc, skA})$

Sofía

Hi

Thom

---------> Hi, I’m Thom

Cert[pkT]<--------

$\text{enc, ss} \leftarrow \text{Encap}(pk_T)$

$\text{enc}$

---------> $\text{ss} \leftarrow \text{Decap}(\text{enc, skT})$

MAC(ss, msgs)<--------

MAC proves to Sofía that Thom has $sk_T$
TLS 1.3 vs server-only AuthKEM

Client

- ClientHello
- Certificate
- CertificateVerify
- Finished
- Application Data

Server

- ServerHello
- EncryptedExtensions
- CertificateRequest
- Certificate
- CertificateVerify
- Finished
- Application Data

TLS 1.3

Client

- ClientHello
- ServerHello
- EncryptedExtensions
- Certificate
- CertificateVerify
- Finished
- Application Data

Server

- KEM Encapsulation
- Application Data
- Finished

AuthKEM

same as TLS 1.3

extra half round trip
TLS 1.3 vs mutual AuthKEM

**Client**
- ClientHello
- Certificate
- CertificateVerify
- Finished
- Application Data

**Server**
- ServerHello
- EncryptedExtensions
- CertificateRequest
- Certificate
- CertificateVerify
- Finished
- Application Data

**TLS 1.3**

**AuthKEM**
Security considerations

- Client sends application data on second flight, but:
  - Server’s ciphersuites not yet authenticated
  - Server only implicitly authenticated
  - Client **MUST** be confident in its selected ciphersuites

- Receiving Server’s *Finished* message grants explicit authentication
  - Any downgrade attack would be detected at this point
  - **Attacked handshakes will never finish successfully**

- Any application data sent **before and after** the Server’s *Finished* message is received:
  - (retroactive) strong downgrade resilience and forward secrecy
TLS 1.3 vs server-only PDK AuthKEM

Client:
- ClientHello
- Certificate
- CertificateVerify
- Finished

Server:
- ServerHello
- EncryptedExtensions
- CertificateRequest
- Certificate
- CertificateVerify
- Finished
- Application Data

Client:
- ClientHello
- KEM Encapsulation
- Finished

Server:
- ServerHello
- EncryptedExtensions
- Finished
- Application Data

TLS 1.3

AuthKEM
TLS 1.3 vs mutual PDK AuthKEM

Client

- ClientHello
- Certificate
- CertificateVerify
- Finished
- Application Data

Server

- ServerHello
- EncryptedExtensions
- CertificateRequest
- Certificate
- CertificateVerify
- Finished
- Application Data

Client

- ClientHello
- KEM Encapsulation
- Certificate (encrypted)

Server

- ServerHello
- EncryptedExtensions
- KEMEncapsulation
- Finished
- Application Data

Finished
- Application Data

same as TLS 1.3

TLS 1.3

AuthKEM
Security considerations for PDK mutual authentication

- The encrypted client certificate:
  - Not encrypted under a forward-secure key. Similar considerations and trade-offs as 0-RTT data.
  - MUST be sent encrypted with a ciphersuite that the server will accept.

- Only <80% of traffic (as noted by Cloudflare) is cached/resumption mode.
Implementation considerations

- New messages, new authentication algorithms
- Handshake state machine closer to TLS 1.2 (Client’s `Finished` is sent first)
- New authenticated handshake secret added to the key schedule
  - Necessary for client authentication
Why use it?

- Same algorithms for KeyExchange and Auth:
  - Push signing algorithm out of the TLS stack
  - In some situations, a signed DH exchange is not appropriate:
    - Delegated Credential with DH key
    - Certificate with an (EC)DH key, as in ietf-curdle-pkix

- The academic works proposing AuthKEM contain a in-depth technical discussion of and a proof of the security of the handshake protocol:
Why use it?

Why not just use draft-ietf-tls-semistatic-dh?

- Requires a non-interactive key exchange; incompatible with PQ KEMs
- PQ NIKE (CSIDH) is very slow (tens of ms)
- CSIDH-512 security level still uncertain (too optimistic?)
Why use it?

- **Post-quantum** KEMs and signature schemes are coming
  - Authentication via KEM saves bytes
    - Large public keys and signatures, and/or;
    - Slow(er) operations, and/or;
    - Special hardware requirements for acceptable perf
- AuthKEM is ideal of constrained environments or servers that support many clients
**Why use it?**

<table>
<thead>
<tr>
<th>Auth via KEM (pk + enc)</th>
<th>Auth via sig (pk + sig)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber-512: 1568 bytes</td>
<td>Dilithium-2: 3732 bytes</td>
</tr>
<tr>
<td>Kyber-768: 2272 bytes</td>
<td>Dilithium-3: 5952 bytes</td>
</tr>
<tr>
<td>NTRU-HPS-2048-509: 1398 bytes</td>
<td>Falcon-512: 1587 bytes</td>
</tr>
</tbody>
</table>

(we use pre-quantum HPKE in the draft as that’s currently standardized)
What about the increased round trips?

- Client can send application data at the same point as in TLS 1.3
- Caching / pdk mechanism avoids this round-trip

Initial experiments at Cloudflare and simulations show (experiments using KEMs for KEX and only post-quantum algorithms):
  - AuthKEM performs as fast as using pq signature algorithms
  - AuthKEM with cached long-term key performs the best

We need more experiments in regards to low latency, low bandwidth, caching parts of the certificate chain, and more.
Thank you!

https://www.ietf.org/id/draft-celi-wiggers-tls-authkem-00.html

(and see the draft for the nitty-gritty details)
## High-level overview of AuthKEM

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientHello</td>
<td>--- &gt;</td>
<td>ServerHello</td>
</tr>
<tr>
<td>Certificate</td>
<td></td>
<td>Finished</td>
</tr>
</tbody>
</table>

- Send over `KemEncapsulation` in reply to `Certificate`.
- Mix in shared secret in key schedule so traffic keys are authenticated.
- Traffic secret can’t be derived without server secret key.
- Client doesn’t have to wait until server sends `Finished` before sending data.
- Client requests are sent in same place as TLS 1.3.
- Client’s `Finished` is sent before server’s.

Unfortunately, mutual auth requires a full extra round-trip.
AuthKEM special scenarios and tricks

- PSK / 0-RTT should be compatible
- If the client has server public key:
  - Send `KemEncapsulation` as a `ClientHello` extension
- Client auth also possible in 1-RTT instead of 2-RTT

---

ClientHello --->
KemEncapsulation --->

ServerHello <<<
Finished <<<

[HTTP Request] --->

[HTTP Response] <<<
Table 1. Average time in $10^{-3}$ seconds of messages for server-only authentication. Note that timings are measured per-client and per-server: each one has its own timer. The 'KEX' label refers to the Key Exchange and the 'Auth' label refers to authentication.

<table>
<thead>
<tr>
<th>Handshake</th>
<th>KEX</th>
<th>Auth</th>
<th>Handshake Flight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1st</td>
</tr>
<tr>
<td>TLS 1.3</td>
<td>X25519</td>
<td>Ed25519</td>
<td>0.227</td>
</tr>
<tr>
<td>TLS 1.3+DC</td>
<td>X25519</td>
<td>Ed25519</td>
<td>0.243</td>
</tr>
<tr>
<td>TLS 1.3+DC</td>
<td>X25519</td>
<td>Ed448</td>
<td>0.242</td>
</tr>
<tr>
<td>PQTLS</td>
<td>Kyber512</td>
<td>Dilithium3</td>
<td>0.350</td>
</tr>
<tr>
<td>PQTLS</td>
<td>SIKEp434</td>
<td>Dilithium4</td>
<td>2.533</td>
</tr>
<tr>
<td>KEMTLS</td>
<td>Kyber512</td>
<td>Kyber512</td>
<td>0.412</td>
</tr>
<tr>
<td>KEMTLS</td>
<td>SIKEp434</td>
<td>SIKEp434</td>
<td>3.058</td>
</tr>
<tr>
<td>KEMTLS-PDK</td>
<td>Kyber512</td>
<td>Kyber512</td>
<td>0.623</td>
</tr>
<tr>
<td>KEMTLS-PDK</td>
<td>SIKEp434</td>
<td>SIKEp434</td>
<td>9.573</td>
</tr>
</tbody>
</table>
(round 2 numbers; K=Kyber, N=Ntru, etc.)
Key Schedule

(EC)DHE $\rightarrow$ HKDF-Extract = Handshake Secret

  $\rightarrow$ Derive-Secret(., "c hs traffic", ClientHello...ServerHello)
  $\rightarrow$ client_handshake_traffic_secret

  $\rightarrow$ Derive-Secret(., "s hs traffic", ClientHello...ServerHello)
  $\rightarrow$ server_handshake_traffic_secret

V Derive-Secret(., "derived", ") = dHS

SSs $\rightarrow$ HKDF-Extract = Authenticated Handshake Secret

  $\rightarrow$ Derive-Secret(., "c ahs traffic", ClientHello...KEMEncapsulation)
  $\rightarrow$ client_handshake_authenticated_traffic_secret

  $\rightarrow$ Derive-Secret(., "s ahs traffic", ClientHello...KEMEncapsulation)
  $\rightarrow$ server_handshake_authenticated_traffic_secret

V Derive-Secret(., "derived", ") = AHS

SSc||0 $\ast$ $\rightarrow$ HKDF-Extract = Master Secret

  $\rightarrow$ Derive-Secret(., "c ap traffic", ClientHello...server Finished)
  $\rightarrow$ client_application_traffic_secret_0

  $\rightarrow$ Derive-Secret(., "s ap traffic", ClientHello...server Finished)
  $\rightarrow$ server_application_traffic_secret_0

  $\rightarrow$ Derive-Secret(., "exp master", ClientHello...server Finished)
  $\rightarrow$ exporter_master_secret

  $\rightarrow$ Derive-Secret(., "res master", ClientHello...client Finished)
  $\rightarrow$ resumption_master_secret