AuthKEM

draft-celi-wiggers-tls-authkem-00

Sofía Celi, Thom Wiggers

Thom Wiggers
Radboud University

Sofía Celi
Cloudflare

AuthKEM

- 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:

- `enc, ss <- `**`Encap(pkB)`**
- `ss <- `**`Decap(enc, skA)`**

MAC proves to Sofía that Thom has `skT`

```
Sofía                                Thom
Hi                      --------->   Hi, I'm Thom
                        Cert[pkT]
                        <---------
enc, ss <- Encap(pkT)
                           enc
                        --------->
                            ss <- Decap(enc, skT)
                        MAC(ss, msgs)
                        <---------
```

TLS 1.3 vs server-only AuthKEM

TLS 1.3

Client | Server

ClientHello →

ServerHello
EncryptedExtensions
CertificateRequest
Certificate
CertificateVerify
Finished
← Application Data

Certificate
CertificateVerify
Finished
Application Data →

Application Data

TLS 1.3

AuthKEM

Client | Server

ClientHello →

ServerHello
EncryptedExtensions
Certificate
←

KEM Encapsulation
Finished
Application Data →

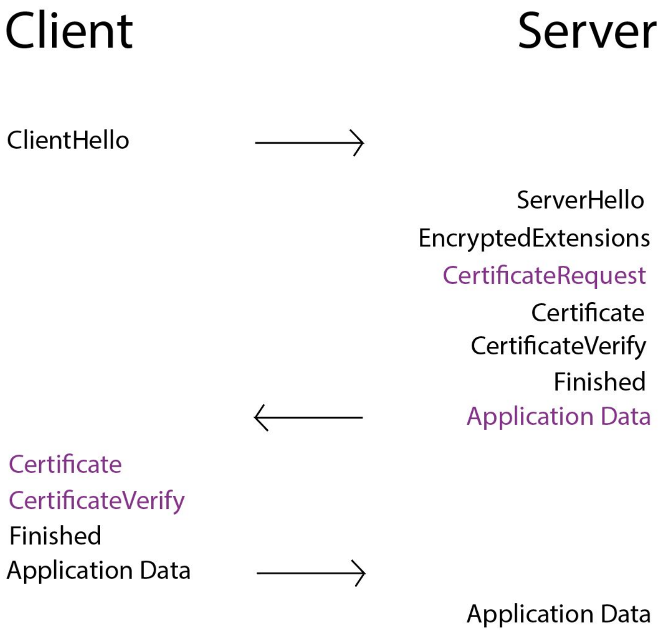
same as TLS 1.3

← Finished
Application Data

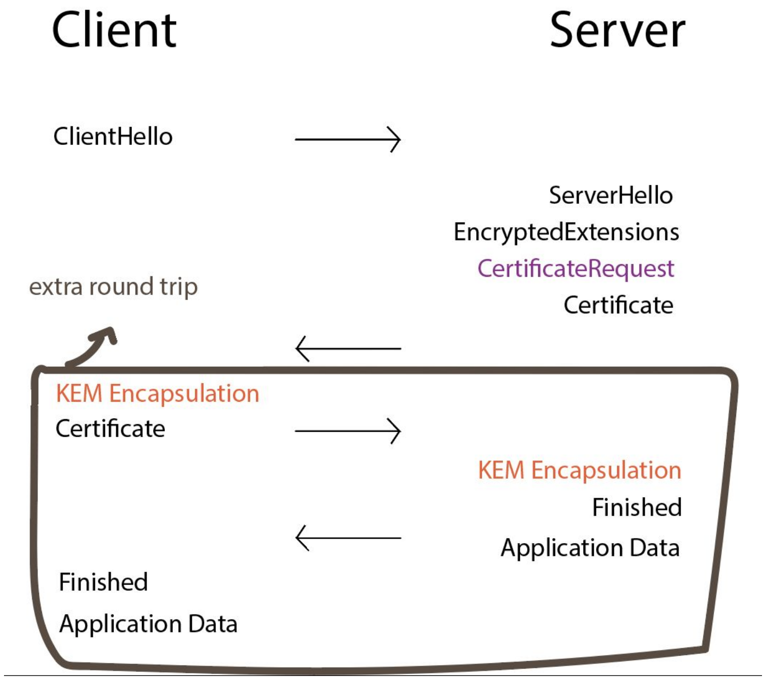
extra half round trip

AuthKEM

TLS 1.3 vs mutual AuthKEM



TLS 1.3



AuthKEM

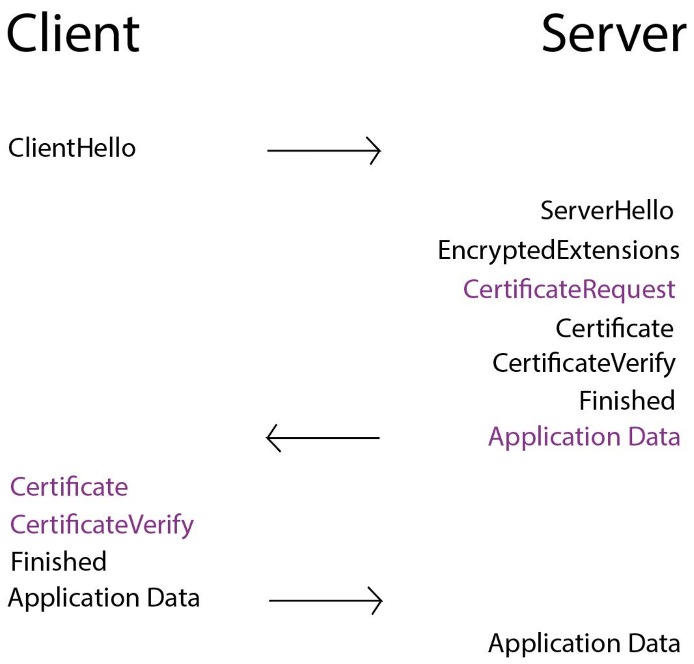
Security considerations

- 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

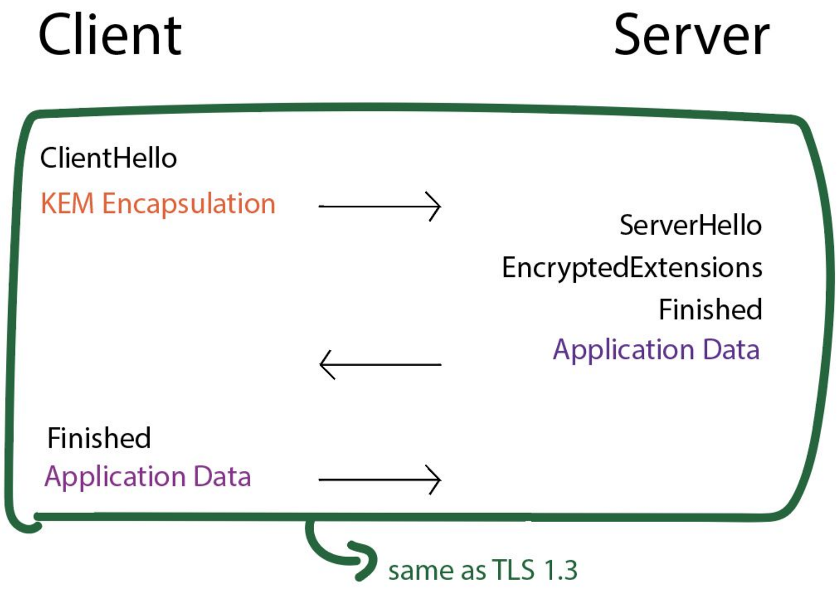
- Receiving Server’s `Finished` message grants explicit authentication
  - Any downgrade attack would be detected at this point
  - **Attacked handshakes will never finish successfully**

- 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



TLS 1.3



AuthKEM

TLS 1.3 vs mutual PDK AuthKEM

TLS 1.3

Client		Server
ClientHello	⟶	
		ServerHello
		EncryptedExtensions
		CertificateRequest
		Certificate
		CertificateVerify
		Finished
	⟵	Application Data
Certificate		
CertificateVerify		
Finished		
Application Data	⟶	
		Application Data

AuthKEM

Client		Server
ClientHello		
KEM Encapsulation		
Certificate (encrypted)	⟶	
		ServerHello
		EncryptedExtensions
		KEMEncapsulation
		Finished
	⟵	Application Data
Finished		
Application Data	⟶	

same as TLS 1.3

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: https://eprint.iacr.org/2020/534.pdf , https://eprint.iacr.org/2021/779.pdf

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
  - PQSigs: few suitable choices (https://eprint.iacr.org/2020/071)
    - 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?

Auth via KEM (pk + enc)		Auth via sig (pk + sig)	
Kyber-512:	1568 bytes	Dilithium-2:	3732 bytes
Kyber-768:	2272 bytes	Dilithium-3:	5952 bytes
NTRU-HPS-2048-509:	1398 bytes	Falcon-512:	1587 bytes

(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

```
Client                            Server

ClientHello        --->
                              ServerHello
                   <---       Certificate

KemEncapsulation   --->
Finished           --->
[HTTP Request]     --->

                   <---          Finished
                   <---[HTTP Response]
```

- 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

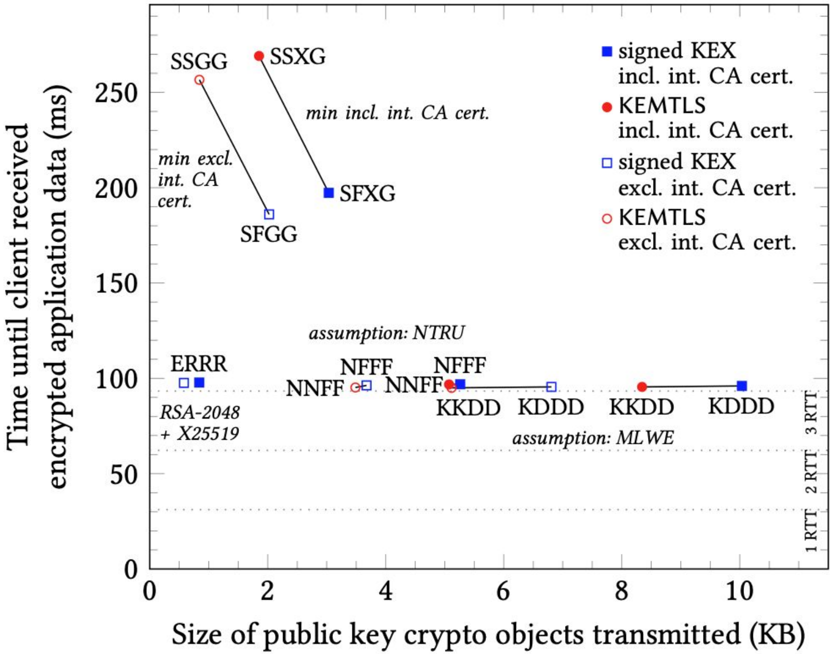
- PSK / 0-RTT should be compatible
- If the client has server public key:
  - Send *KemEncapsulation* as a *ClientHello* extension
- Client auth also possible in 1-RTT instead of 2-RTT

```
Client                              Server

ClientHello          --->
KemEncapsulation     --->
                     <---    ServerHello
                     <---    Finished
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.

Handshake	KEX	Auth	Handshake Flight			
			1^{st}	2^{nd}	3^{th}	4^{th}
TLS 1.3	X25519	Ed25519	0.227	0.436	123.838	180.202
TLS 1.3+DC	X25519	Ed25519	0.243	0.489	156.954	186.868
TLS 1.3+DC	X25519	Ed448	0.242	0.907	165.395	183.124
PQTLS	Kyber512	Dilithium3	0.350	0.701	173.814	198.256
PQTLS	SIKEp434	Dilithium4	2.533	4.856	441.732	212.924
KEMTLS	Kyber512	Kyber512	0.412	0.217	157.123	187.147
KEMTLS	SIKEp434	SIKEp434	3.058	7.215	352.840	291.592
KEMTLS-PDK	Kyber512	Kyber512	0.623	0.327	181.132	189.442
KEMTLS-PDK	SIKEp434	SIKEp434	9.573	12.507	396.818	287.550



(round 2 numbers; K=Kyber, N=Ntru, etc.)

Key Schedule

```
                 v
(EC)DHE -> HKDF-Extract = Handshake Secret
                 |
                 +--> Derive-Secret(., "c hs traffic",
                 |                  ClientHello...ServerHello)
                 |                  = client_handshake_traffic_secret
                 |
                 +--> Derive-Secret(., "s hs traffic",
                 |                  ClientHello...ServerHello)
                 |                  = server_handshake_traffic_secret
                 v
                 Derive-Secret(., "derived", "") = dHS
                 |
                 v
    SSs -> HKDF-Extract = Authenticated Handshake Secret
                 |
                 +--> Derive-Secret(., "c ahs traffic",
                 |                  ClientHello...KEMEncapsulation)
                 |                  = client_handshake_authenticated_traffic_secret
                 |
                 +--> Derive-Secret(., "s ahs traffic",
                 |                  ClientHello...KEMEncapsulation)
                 |                  = server_handshake_authenticated_traffic_secret
                 v
                 Derive-Secret(., "derived", "") = AHS
                 |
                 v
SSc||0 * -> HKDF-Extract = Master Secret
                 |
                 +--> Derive-Secret(., "c ap traffic",
                 |                  ClientHello...server Finished)
                 |                  = client_application_traffic_secret_0
                 |
                 +--> Derive-Secret(., "s ap traffic",
                 |                  ClientHello...server Finished)
                 |                  = server_application_traffic_secret_0
                 |
                 +--> Derive-Secret(., "exp master",
                 |                  ClientHello...server Finished)
                 |                  = exporter_master_secret
                 |
                 +--> Derive-Secret(., "res master",
                                    ClientHello...client Finished)
                                    = resumption_master_secret
```