AuthKEM and AuthKEM-PSK

KEM-based, signature-free
handshake authentication for TLS

https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/
https://datatracker.ietf.org/doc/draft-wiggers-tls-authkem-psk/
https://github.com/kemtls/

Previously on AuthKEM...

Previously on AuthKEM...

- Origin: KEMTLS (ACM CCS 2020) and KEMTLS-PDK (ESORICS 2021)
  - Academic works that proposed KEM-based authentication + KEM ephemeral key exchange
  - We did our homework: we have proofs (pen-and-paper) and models in Tamarin (ESORICS 2022)

Previously on AuthKEM...

- Origin: KEMTLS (ACM CCS 2020) and KEMTLS-PDK (ESORICS 2021)
  - Academic works that proposed KEM-based authentication + KEM ephemeral key exchange
  - We did our homework: we have proofs (pen-and-paper) and models in Tamarin (ESORICS 2022)
- We wrote a draft: AuthKEM-00 (July 2021)
  - The handshake authentication parts from KEMTLS (“Auth via KEM”)
  - Way too long and complicated
  - Got some valuable feedback

Previously on AuthKEM...

- Origin: KEMTLS (ACM CCS 2020) and KEMTLS-PDK (ESORICS 2021)
  - Academic works that proposed KEM-based authentication + KEM ephemeral key exchange
  - We did our homework: we have proofs (pen-and-paper) and models in Tamarin (ESORICS 2022)
- We wrote a draft: AuthKEM-00 (July 2021)
  - The handshake authentication parts from KEMTLS (“Auth via KEM”)
  - Way too long and complicated
  - Got some valuable feedback
- We updated the draft: AuthKEM-01 (March 2022)
  - Presented the AuthKEM Abridged companion FAQ
  - https://thomwiggers.nl/docs/authkem-abridged/

PQSHIELD

Open Access Article

Post-Quantum Authentication in the MQTT Protocol

by Juliet Samandari *,† and Clémentine Gritti †

Department of Computer Science and Software Engineering, University of Canterbury, Christchurch 8014, New Zealand

* Author to whom correspondence should be addressed.

† These authors contributed equally to this work.

J. Cybersecur. Priv. **2023**, *3*(3), 416-434; https://doi.org/10.3390/jcp3030021

Received: 13 May 2023 / Revised: 15 June 2023 / Accepted: 12 July 2023 / Published: 31 July 2023

(This article belongs to the Section **Security Engineering & Applications**)

“ We found that the use of KEM for authentication resulted in a speed increase of 25 ms, a saving of 71%.

The road ahead for AuthKEM

The road ahead for AuthKEM

- A few weeks ago: Split the draft into two proposals
  - AuthKEM-02 (“KEM public keys in certificates”)
  - AuthKEM-PSK-00 (“KEM-based resumption”)

The road ahead for AuthKEM

- A few weeks ago: Split the draft into two proposals
  - AuthKEM-02 (“KEM public keys in certificates”)
  - AuthKEM-PSK-00 (“KEM-based resumption”)
- Greatly improved their presentation and added a comparison/motivation section to AuthKEM

The road ahead for AuthKEM

- A few weeks ago: Split the draft into two proposals
  - AuthKEM-02 (“KEM public keys in certificates”)
  - AuthKEM-PSK-00 (“KEM-based resumption”)
- Greatly improved their presentation and added a comparison/motivation section to AuthKEM
- Today: briefly explain the update and ask for comments and support

PQSHIELD

The road ahead for AuthKEM

- A few weeks ago: Split the draft into two proposals
  - AuthKEM-02 (“KEM public keys in certificates”)
  - AuthKEM-PSK-00 (“KEM-based resumption”)
- Greatly improved their presentation and added a comparison/motivation section to AuthKEM
- Today: briefly explain the update and ask for comments and support
- Soon™: adopt one or both?

Splitting AuthKEM

Splitting AuthKEM

KEM authentication for TLS

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications

PQSHIELD

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

KEM-based PSK-style handshakes

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

KEM-based PSK-style handshakes

- Use KEM ciphertext instead of PSK

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

KEM-based PSK-style handshakes

- Use KEM ciphertext instead of PSK
- KEM public keys are less sensitive than PSKs (storage, device provisioning)

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

KEM-based PSK-style handshakes

- Use KEM ciphertext instead of PSK
- KEM public keys are less sensitive than PSKs (storage, device provisioning)
- KEM public keys probably have fewer tracking issues than opaque session tickets

Splitting AuthKEM

KEM authentication for TLS

- Use KEM public key for handshake auth
- Save lots of handshake traffic
- Intended for “full” TLS handshakes
- Expected savings very significant for non-desktop applications
- Combines well with Merkle Tree Certs / abridged certificates

KEM-based PSK-style handshakes

- Use KEM ciphertext instead of PSK
- KEM public keys are less sensitive than PSKs (storage, device provisioning)
- KEM public keys probably have fewer tracking issues than opaque session tickets
- Cache an AuthKEM certificate

github.com/kemtls/

Now what?

PQSHIELD

github.com/kemtls/

Now what?

- Please submit your love, support, PRs, and comments on the mailing list
    - In particular: “I would like to use…”
    - **Example:** PQShield thinks both of these protocols could be interesting for our customers.

Additional content

Why use AuthKEM instead of HS signatures?

```
Client                                              Server

ClientHello
+ key_share
+ signature_algorithms
                          -------->
                                               ServerHello
                                               + key_share
                                     <EncryptedExtensions>
                          <-------- <Certificate: kem pk>

<KEMEncapsulation>        -------->
{Finished}                -------->
[Application Data]        -------->
  GET /cat.gif HTTP/1.1

                          <--------             {Finished}

[Application Data]        <-------->    [Application Data]
```

Why use AuthKEM instead of HS signatures?

- Reduce bandwidth impact
  - Replace handshake pk/signature by KEM
  - e.g. replace Dilithium-2 by Kyber-768: 3732 → 2272 bytes (-39%) for handshake authentication
  - Note: Falcon is not suitable for online signatures!
  - But: combining AuthKEM with Falcon for offline signatures is possible
    - Using AuthKEM can reuse the KEM implementation from key exchange
    - so don't need Kyber AND Dilithium AND Falcon implementations → reduces code size/complexity

```
Client                                                Server

ClientHello
+ key_share
+ signature_algorithms
                           -------->
                                                 ServerHello
                                                 + key_share
                                       <EncryptedExtensions>
                           <--------   <Certificate: kem pk>

<KEMEncapsulation>         -------->
{Finished}                 -------->
[Application Data]         -------->
  GET /cat.gif HTTP/1.1

                           <--------              {Finished}

[Application Data]         <------->      [Application Data]
```

Why use AuthKEM instead of HS signatures?

- Reduce bandwidth impact
  - Replace handshake pk/signature by KEM
  - e.g. replace Dilithium-2 by Kyber-768: 3732 → 2272 bytes (-39%) for handshake authentication
  - Note: Falcon is not suitable for online signatures!
  - But: combining AuthKEM with Falcon for offline signatures is possible
    - Using AuthKEM can reuse the KEM implementation from key exchange
    - so don't need Kyber AND Dilithium AND Falcon implementations → reduces code size/complexity
- Reduce code size / hardware area
  - Allows re-use of (hardened?) KEM implementation for auth

```
Client                                                  Server

ClientHello
+ key_share
+ signature_algorithms
                              -------->
                                                   ServerHello
                                                   + key_share
                                        <EncryptedExtensions>
                              <-------- <Certificate: kem pk>

<KEMEncapsulation>            -------->
{Finished}                    -------->
[Application Data]            -------->
  GET /cat.gif HTTP/1.1

                              <--------             {Finished}

[Application Data]            <------->     [Application Data]
```

Why use AuthKEM instead of HS signatures?

- Reduce bandwidth impact
  - Replace handshake pk/signature by KEM
  - e.g. replace Dilithium-2 by Kyber-768: 3732 → 2272 bytes (-39%) for handshake authentication
  - Note: Falcon is not suitable for online signatures!
  - But: combining AuthKEM with Falcon for offline signatures is possible
    - Using AuthKEM can reuse the KEM implementation from key exchange
    - so don't need Kyber AND Dilithium AND Falcon implementations → reduces code size/complexity
- Reduce code size / hardware area
  - Allows re-use of (hardened?) KEM implementation for auth
- Kyber decaps is a lot faster than Dilithium signature generation

```
Client                                              Server

ClientHello
+ key_share
+ signature_algorithms
                         -------->
                                               ServerHello
                                               + key_share
                                     <EncryptedExtensions>
                         <--------   <Certificate: kem pk>

<KEMEncapsulation>       -------->
{Finished}               -------->
[Application Data]       -------->
  GET /cat.gif HTTP/1.1

                         <--------              {Finished}

[Application Data]       <------->      [Application Data]
```

Instance labels:
ABCD:
A: KEX
B: HS Auth
C: Intermediate CA
D: Root CA

Kyber-512,
Dilithium-2,
Falcon-512

Table 13.5: Comparison of handshake size and time until the client receives a response from the server (30.9 ms, 1000 Mbps), between unilaterally authenticated post-quantum TLS 1.3 and KEMTLS instances at NIST level I.

Experiment		Handshake size (bytes)				Time until response (ms)			
		No int.	Δ%	With int.	Δ%	No int.	Δ%	With int.	Δ%
TLS	KDDD	7720	−28.0 %	11 452	−18.9 %	94.8	−0.4 %	95.0	−0.3 %
KEMTLS	KKDD	5556		9288		94.4		94.8	
TLS	KFFF	3797	+0.1 %	5360	+0.1 %	95.8	−1.3 %	96.1	−1.2 %
KEMTLS	KKFF	3802		5365		94.5		94.9	
TLS	KDFF	5966	−36.3 %	7529	−28.7 %	94.8	−0.3 %	95.2	−0.3 %
KEMTLS	KKFF	3802		5365		94.5		94.9	

Handshake sizes are shown without ('no int') and with intermediate certificates;
SCTs or OCSP are not included.

Source: Synthetic benchmarks in Chapter 13 of "Post-Quantum TLS", Thom Wiggers, PhD thesis (to appear)

What doesn't AuthKEM solve

PQSHIELD

What doesn’t AuthKEM solve

- Web PKI has too many signatures
  - AuthKEM can only replace the online signature
  - Certificate chain / SCT / OCSP Staples remain a challenge
  - AuthKEM leaves only offline signatures, so might make Falcon easier to use
  - Also, AuthKEM combines well with Merkle Tree Certificates / draft-ietf-tls-cert-abridge-00

What doesn't AuthKEM solve

- Web PKI has too many signatures
  - AuthKEM can only replace the online signature
  - Certificate chain / SCT / OCSP Staples remain a challenge
  - AuthKEM leaves only offline signatures, so might make Falcon easier to use
  - Also, AuthKEM combines well with Merkle Tree Certificates / draft-ietf-tls-cert-abridge-00
- Server can no longer send application data in first reply
  - Generally at most a banner or insensitive configuration information
  - HTTP/2 could use a server-side ALPN equivalent

What doesn’t AuthKEM solve

- Web PKI has too many signatures
  - AuthKEM can only replace the online signature
  - Certificate chain / SCT / OCSP Staples remain a challenge
  - AuthKEM leaves only offline signatures, so might make Falcon easier to use
  - Also, AuthKEM combines well with Merkle Tree Certificates / draft-ietf-tls-cert-abridge-00
- Server can no longer send application data in first reply
  - Generally at most a banner or insensitive configuration information
  - HTTP/2 could use a server-side ALPN equivalent

What doesn’t AuthKEM solve

- Web PKI has too many signatures
  - AuthKEM can only replace the online signature
  - Certificate chain / SCT / OCSP Staples remain a challenge
  - AuthKEM leaves only offline signatures, so might make Falcon easier to use
  - Also, AuthKEM combines well with Merkle Tree Certificates / draft-ietf-tls-cert-abridge-00
- Server can no longer send application data in first reply
  - Generally at most a banner or insensitive configuration information
  - HTTP/2 could use a server-side ALPN equivalent
- Client authentication in AuthKEM requires a full additional round-trip
  - Client certificate can not be sent before receiving server certificate
  - KEM authentication requires receiving and processing a response to client certificate message
  - Client authentication seems irrelevant to the human-facing Web [Birghan & Van der Merwe 2022]
  - Machine-to-machine services using mTLS might be able to mitigate via AuthKEM-PSK

What doesn't AuthKEM solve

- Web PKI has too many signatures
  - AuthKEM can only replace the online signature
  - Certificate chain / SCT / OCSP Staples remain a challenge
  - AuthKEM leaves only offline signatures, so might make Falcon easier to use
  - Also, AuthKEM combines well with Merkle Tree Certificates / draft-ietf-tls-cert-abridge-00
- Server can no longer send application data in first reply
  - Generally at most a banner or insensitive configuration information
  - HTTP/2 could use a server-side ALPN equivalent
- Client authentication in AuthKEM requires a full additional round-trip
  - Client certificate can not be sent before receiving server certificate
  - KEM authentication requires receiving and processing a response to client certificate message
  - Client authentication seems irrelevant to the human-facing Web [Birghan & Van der Merwe 2022]
  - Machine-to-machine services using mTLS might be able to mitigate via AuthKEM-PSK
- Web browsers seem less sensitive to the code size / bandwidth / complexity arguments than some other environments

AuthKEM-PSK

```
       Client                                              Server
        Knows server's authkem-psk pk

Key   ^ ClientHello
Exch  | + key_share
&     | + stored_auth_key (ciphertext)
Auth  | + signature_algorithms
      | + early_auth*
      | + early_data*
      | (Certificate*)
      | (Application Data*)    -------->         ServerHello  ^
      |                                          + key_share  |
      |                                    + stored_auth_key  | Key
      |                                        + early_auth*  | Exch,
      |                                        + early_data*  | Auth &
      |                                {EncryptedExtensions}  | Server
      |                                 {KEMEncapsulation*}   | Params
      |                        <--------          {Finished}  v
      |                        <-------- [Application Data*]
      | (EndOfEarlyData)
      v {Finished}             -------->

        [Application Data]     <------->  [Application Data]
```

AuthKEM-PSK

- Use a KEM public key to abbreviate the TLS handshake

```
       Client                                           Server
        Knows server's authkem-psk pk

Key   ^ ClientHello
Exch  | + key_share
&     | + stored_auth_key (ciphertext)
Auth  | + signature_algorithms
      | + early_auth*
      | + early_data*
      | (Certificate*)
      | (Application Data*)    -------->         ServerHello  ^
      |                                          + key_share  |
      |                                    + stored_auth_key  | Key
      |                                        + early_auth*  | Exch,
      |                                        + early_data*  | Auth &
      |                                {EncryptedExtensions}  | Server
      |                                 {KEMEncapsulation*}   | Params
      |                        <--------          {Finished}  v
      |                        <-------- [Application Data*]
      | (EndOfEarlyData)
      v {Finished}             -------->

        [Application Data]    <-------->  [Application Data]
```


AuthKEM-PSK

- Use a KEM public key to abbreviate the TLS handshake
- Send a ciphertext in the ClientHello message

```
       Client                                               Server
        Knows server's authkem-psk pk

Key   ^ ClientHello
Exch  | + key_share
&     | + stored_auth_key (ciphertext)
Auth  | + signature_algorithms
      | + early_auth*
      | + early_data*
      | (Certificate*)
      | (Application Data*)    -------->          ServerHello  ^
      |                                           + key_share  |
      |                                     + stored_auth_key  | Key
      |                                         + early_auth*  | Exch,
      |                                         + early_data*  | Auth &
      |                                 {EncryptedExtensions}  | Server
      |                                   {KEMEncapsulation*}  | Params
      |                        <--------           {Finished}  v
      |                        <-------- [Application Data*]
      | (EndOfEarlyData)
      v {Finished}             -------->

        [Application Data]     <------->  [Application Data]
```

AuthKEM-PSK

- Use a KEM public key to abbreviate the TLS handshake
- Send a ciphertext in the ClientHello message
- Potentially allow sending client certificate in first message

```
        Client                                                Server
         Knows server's authkem-psk pk

Key    ^ ClientHello
Exch   | + key_share
&      | + stored_auth_key (ciphertext)
Auth   | + signature_algorithms
       | + early_auth*
       | + early_data*
       | (Certificate*)
       | (Application Data*)     -------->          ServerHello  ^
       |                                            + key_share  |
       |                                      + stored_auth_key  | Key
       |                                           + early_auth*  | Exch,
       |                                           + early_data*  | Auth &
       |                                   {EncryptedExtensions}  | Server
       |                                     {KEMEncapsulation*}  | Params
       |                        <--------             {Finished}  v
       |                        <-------- [Application Data*]
       | (EndOfEarlyData)
       v {Finished}             -------->

         [Application Data]    <------->   [Application Data]
```

AuthKEM-PSK

- Use a KEM public key to abbreviate the TLS handshake
- Send a ciphertext in the ClientHello message
- Potentially allow sending client certificate in first message
- Selected TODO items:
  - Should we try to integrate into PSK or set up a different extension (currently)?
    - See appendix / Issue #25
  - Should/how do we handle early data / "early authentication" exactly

```
       Client                                                 Server
        Knows server's authkem-psk pk

Key  ^ ClientHello
Exch | + key_share
&    | + stored_auth_key (ciphertext)
Auth | + signature_algorithms
     | + early_auth*
     | + early_data*
     | (Certificate*)
     | (Application Data*)    -------->          ServerHello  ^
     |                                           + key_share  |
     |                                     + stored_auth_key  | Key
     |                                         + early_auth*  | Exch,
     |                                         + early_data*  | Auth &
     |                                 {EncryptedExtensions}  | Server
     |                                  {KEMEncapsulation*}   | Params
     |                       <--------             {Finished} v
     |                       <--------  [Application Data*]
     | (EndOfEarlyData)
     v {Finished}            -------->

       [Application Data]    <------->   [Application Data]
```

PQSHIELD

AuthKEM-PSK

- Use a KEM public key to abbreviate the TLS handshake
- Send a ciphertext in the ClientHello message
- Potentially allow sending client certificate in first message
- Selected TODO items:
  - Should we try to integrate into PSK or set up a different extension (currently)?
    - See appendix / Issue #25
  - Should/how do we handle early data / “early authentication” exactly
- Cryptographic side note: handshake is “fresh”, unlike PSK resumptions!

```
       Client                                              Server
        Knows server's authkem-psk pk

Key  ^ ClientHello
Exch | + key_share
&    | + stored_auth_key (ciphertext)
Auth | + signature_algorithms
     | + early_auth*
     | + early_data*
     | (Certificate*)
     | (Application Data*)     -------->          ServerHello  ^
     |                                            + key_share  |
     |                                      + stored_auth_key  | Key
     |                                          + early_auth*  | Exch,
     |                                          + early_data*  | Auth &
     |                                  {EncryptedExtensions}  | Server
     |                                    {KEMEncapsulation*}  | Params
     |                         <--------           {Finished}  v
     |                         <--------  [Application Data*]
     | (EndOfEarlyData)
     v {Finished}              -------->

       [Application Data]      <------->   [Application Data]
```

Why AuthKEM-PSK

Why AuthKEM-PSK

- Suitable for those places where you would use pre-shared symmetric keys

Why AuthKEM-PSK

- Suitable for those places where you would use pre-shared symmetric keys
- No pre-shared symmetric key headaches
  - E.g. all IoT devices can be set up with the same KEM public key
  - If KEM public key is extracted from a device, you're fine
    - Recall: symmetric PSK compromise allows server/client MITM and full impersonation

Why AuthKEM-PSK

- Suitable for those places where you would use pre-shared symmetric keys
- No pre-shared symmetric key headaches
  - E.g. all IoT devices can be set up with the same KEM public key
  - If KEM public key is extracted from a device, you're fine
    - Recall: symmetric PSK compromise allows server/client MITM and full impersonation
- If “early authentication” is used, we get a 1-RTT mutually KEM-authenticated handshake

PQSHIELD

Why AuthKEM-PSK

- Suitable for those places where you would use pre-shared symmetric keys
- No pre-shared symmetric key headaches
  - E.g. all IoT devices can be set up with the same KEM public key
  - If KEM public key is extracted from a device, you're fine
    - Recall: symmetric PSK compromise allows server/client MITM and full impersonation
- If "early authentication" is used, we get a 1-RTT mutually KEM-authenticated handshake
- Avoid transmitting big (post-quantum) certificates
  - And avoid signature verification (code) altogether

Why AuthKEM-PSK

- Suitable for those places where you would use pre-shared symmetric keys
- No pre-shared symmetric key headaches
  - E.g. all IoT devices can be set up with the same KEM public key
  - If KEM public key is extracted from a device, you're fine
    - Recall: symmetric PSK compromise allows server/client MITM and full impersonation
- If "early authentication" is used, we get a 1-RTT mutually KEM-authenticated handshake
- Avoid transmitting big (post-quantum) certificates
  - And avoid signature verification (code) altogether

We think AuthKEM-PSK allows all sorts of interesting setups in (possibly extremely restricted) embedded or IoT applications