KEM-based Authentication for IKEv2 with Post-quantum Security

draft-wang-ipsecme-kem-auth-ikev2-03

Guilin Wang (Wang.Guilin@huawei.com)
Valery Smyslov (svan@elvis.ru)

IETF 125

Motivation and Related Works

- ML-KEM data is shorter than ML-DSA
  - for Levels 1, 3, and 5 is shorter by 2164, 2989, and 4083 bytes, respectively
- ML-KEM is faster than ML-DSA
  - KeyGen 4 times faster, encap 7 times faster than sign, and decap 3 times faster than verify
- KEM-based authentication can provide additional security properties, not available with signature authentication
- Prior works: MQV, HMQV, (R)PKE authentication for IKEv1
- Related works:
  - PQuAKE protocol (presented at IETF 123)
  - draft-celi-wiggers-tls-authkem (KEM-based Authentication for TLS 1.3)
  - draft-wiggers-tls-authkem-psk (KEM-based pre-shared-key handshakes for TLS 1.3)

KEM-based Authentication Outline

- Peers perform ephemeral (hybrid) key exchange that protects subsequent communications
- Peers exchange their long-term KEM certificates
- Peers perform two key exchanges using long-term keys from KEM certificates, resulting in keys SSi and SSr
- Peers confirm to each other that they received the right SS keys by using these keys in computing authentication data, that authenticate the protocol transcript (key confirmation)
- Peers stir SS keys into the session keys

Use in IKEv2

- Rely on `IKE_SA_INIT` and `IKE_INTERMEDIATE` (as per RFC 9370) for ephemeral hybrid PQ/T key exchange
- Rely on RFC 9867 for optional additional protection of peers' identities with group PPK
- Rely on `Hash&URL` certificate encoding to allow peers to provide KEM certificates by reference
- Use `IKE_INTERMEDIATE` (RFC 9242) for exchange of KEM certificates and for exchange of encapsulated SS keys
- Use `IKE_AUTH` for key confirmation, modifying the `AUTH` payload calculation
- Use of draft-ietf-ipsecme-ikev2-downgrade-prevention is (highly) recommended
- RFC 9593 can be used for announcement supported KEMs

Changes in -03 Version

- Initiator's identity protection against active attacker is added
  - responder sends its KEM certificate first
  - initiator sends its KEM certificate encrypted with a key that only the responder can compute
- New payload `Encrypted Certificate` (`ECERT`)
- New Certificate Encoding `X.509 bundle`
- Changes in Notifications
  - new `AUTHKEM_CT`, `AUTHKEM_SSCONF`
  - gone `AUTHKEM_DATA`

Protocol Exchanges

Initiator — **Exchange** (RFCs), `Session Keys` — Responder

IKE_SA_INIT (7296)

```
HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(INTERMEDIATE_EXCHANGE_SUPPORTED),
N(USE_AUTHKEM)                          →
                                        ←  HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
                                           N(INTERMEDIATE_EXCHANGE_SUPPORTED),
                                           N(USE_AUTHKEM)
```

Protocol Exchanges

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867]), $SK_1 = f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)]}

Protocol Exchanges

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867]), $SK_1=f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)]}

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

HDR,SK_2{[CERTREQ]} →

← HDR,SK_2{CERT}

Protocol Exchanges

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867]), $SK_1=f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)]}

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

HDR,SK_2{[CERTREQ]} →

← HDR,SK_2{CERT}

IKE_INTERMEDIATE (AUTHKEM)

HDR,SK_2{N(**AUTHKEM_CT,ECERT**)} →

← HDR,SK_2{N(**AUTHKEM_CT,AUTHKEM_SSCONF**)}

Protocol Exchanges

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867]), $SK_1=f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)]}

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

HDR,SK_2{[CERTREQ]} →

← HDR,SK_2{CERT}

IKE_INTERMEDIATE (AUTHKEM)

HDR,SK_2{N(**AUTHKEM_CT,ECERT**)} →

← HDR,SK_2{N(**AUTHKEM_CT,AUTHKEM_SSCONF**)}

IKE_AUTH (7296), **$SK_3 = f(SK_s,SSi,SSr)$**

HDR,SK_3{IDi,AUTH(**KEM Auth**),SAi2,TSi,TSr} →

← HDR,SK_3{IDr,AUTH(**KEM Auth**),SAr2,TSi,TSr}

Computing Session Keys

- Once last `IKE_INTERMEDIATE` completes, session keys are updated:

  SKEYSEED' = prf(SK_d, SSi | SSr)

  (SK_* keys are then re-computed as per RFC 7296 using SKEYSEED')
- Thus the session keys for IKE SA and IPsec SAs depend not only on ephemeral key exchange(s), but also on SSi and SSr

New Notifications

- `USE_AUTHKEM` – to negotiate the use of KEM-based authentication
  - contains no data
- `AUTHKEM_CT` – to exchange SS values
  - data is ciphertext (either CTi or CTr)
- `AUTHKEM_SSCONF` – to confirm SSr value
  - data is computed as
    SSCONF = prf(SK_pr, SSr)

Role of SSCONF

- SSCONF value allows initiator to detect possible misconfiguration when public key from responder's KEM certificate doesn't match its private key
  - since SSr key is immediately stirred into the session keys for the next exchange, the result of such misconfiguration is inability for responder to decrypt messages
- Initiator should check received SSCONF value. If it is correct, then initiator can stir SSr into session keys
  - otherwise, peers will be unable to continue communication since session keys for the next exchange would be different, but at least the initiator knows this fact and can log the problem (hopefully to be fixed in case of misconfiguration)
- SSi is checked by responder when decrypting the initiator's certificate, no need for SSCONF

AUTH Payload

- Document defines new Authentication Method KEM Authentication. Authentication data is computed as

For initiator:

```
AUTH = prf( prf(SK_pi, "Key Pad for IKEv2"),
                    <InitiatorSignedOctets>)
```

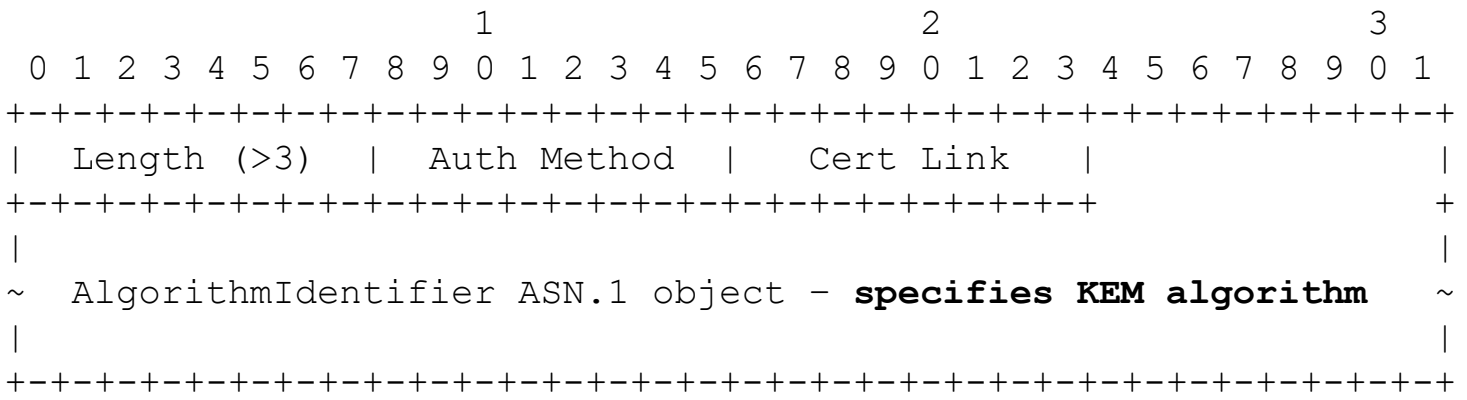
For responder:

```
AUTH = prf( prf(SK_pr, "Key Pad for IKEv2"),
                    <ResponderSignedOctets>)
```

- Do we need a different string here? For example `"KEM Auth for IKEv2"`?

Announcing New Auth Method

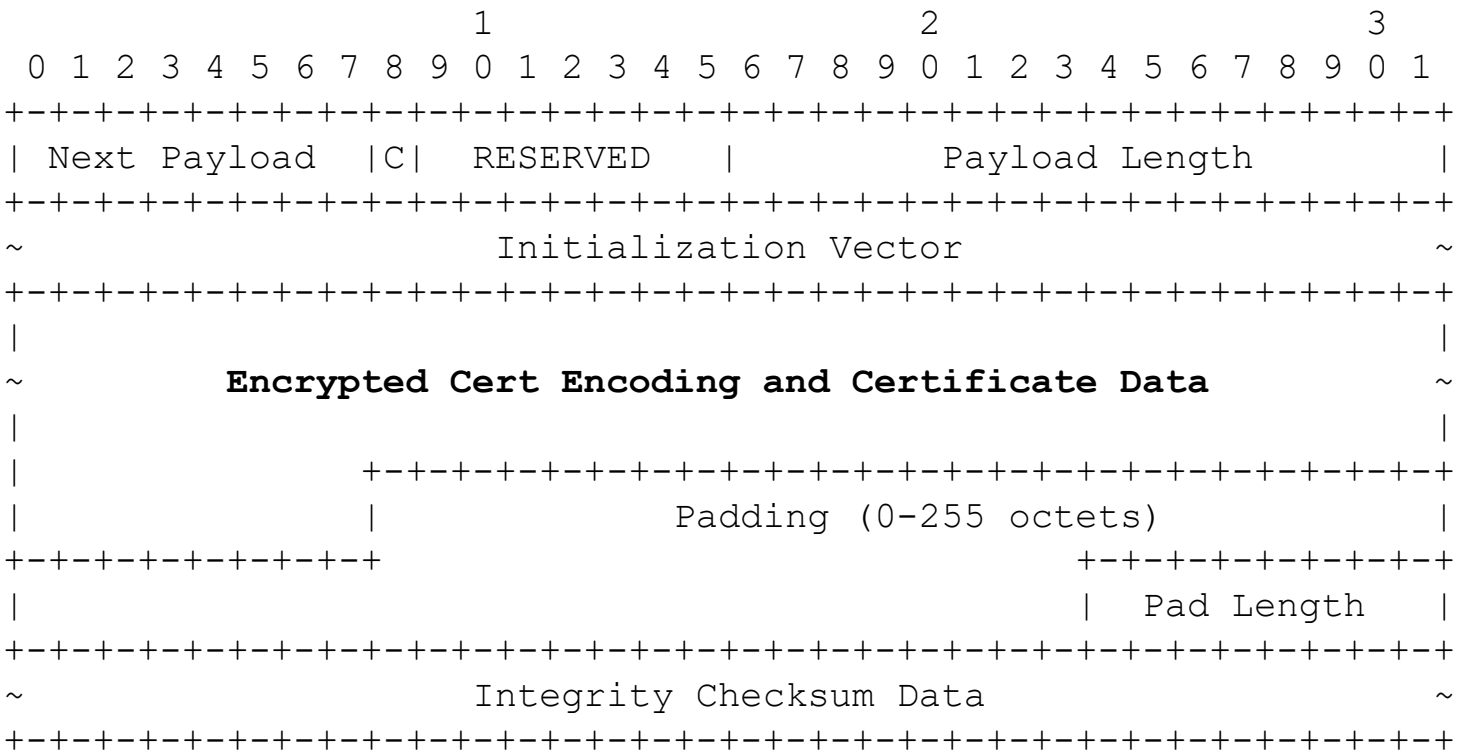
- RFC 9593 allows peers to announce supported authentication methods in `SUPPORTED_AUTH_METHODS` notification
  - format of announcement for KEM Authentication method is the same as for “Digital Signature”



- While IKEv2 allows authentication methods to differ in different directions, with KEM-based authentication this is not possible
  - if KEM-based authentication is negotiated, peers **should not** announce support for methods other than KEM Authentication
  - they can announce support for **several** KEMs and actually use **different** KEMs

Encrypted Certificate Payload

- Content is the same as for Certificate payload, but in encrypted form. Format is similar to the Encrypted payload. Algorithms are the same as for IKE SA. Algorithm keys are computed as EC_a | EC_e = prf+ (SK_d, K), where K is the protection key (e.g. SSi)



“X.509 bundle” Cert Encoding

- Allows several certificates and CRLs to be sent in a single payload
- Defined in RFC 7296 (with `Hash and URL of X.509 bundle` cert encoding)

```
CertBundle
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-cert-bundle(34) }

  DEFINITIONS EXPLICIT TAGS ::=
  BEGIN

  IMPORTS
    Certificate, CertificateList
    FROM PKIX1Explicit88
       { iso(1) identified-organization(3) dod(6)
         internet(1) security(5) mechanisms(5) pkix(7)
         id-mod(0) id-pkix1-explicit(18) } ;

  CertificateOrCRL ::= CHOICE {
    cert [0] Certificate,
    crl  [1] CertificateList }

  CertificateBundle ::= SEQUENCE OF CertificateOrCRL
```

Security Properties

- Existing security properties of IKEv2 are preserved (we believe)
- New security properties
  - repudiation: since no data on the wire depend on peers' long-term private keys, each peer can repudiate to 3rd party the fact of exchange
  - enhanced PQ security for session keys: the keys depend not only on ephemeral key exchange(s), but also on keys resulted from exchanges using long-term KEM certificates

Possible Protocol Optimization

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

```
HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(INTERMEDIATE_EXCHANGE_SUPPORTED),
N(USE_AUTHKEM)                                  →
                                                ←   HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
                                                        N(INTERMEDIATE_EXCHANGE_SUPPORTED),
                                                        N(USE_AUTHKEM)
```

IKE_INTERMEDIATE (9370 [,9867]), $SK_1=f(KE)$

```
HDR,SK1{KEi1[,N(PPK_IDENTITY_KEY)]}             →
                                                ←   HDR,SK1{KEr1[,N(PPK_IDENTITY)]}
```

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

```
HDR,SK2{[CERTREQ]}                              →
                                                ←   HDR,SK2{CERT}
```

IKE_INTERMEDIATE (AUTHKEM)

```
HDR,SK2{N(AUTHKEM_CT,ECERT)}                    →
                                                ←   HDR,SK2{N(AUTHKEM_CT,AUTHKEM_SSCONF)}
```

IKE_AUTH (7296), **$SK_3 = f(SK_s,SSi,SSr)$**

```
HDR,SK3{IDi,AUTH(KEM Auth),SAi2,TSi,TSr}        →
                                                ←   HDR,SK3{IDr,AUTH(KEM Auth),SAr2,TSi,TSr}
```

Can we Remove this Exchange?

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867]), $SK_1=f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)]}

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

HDR,SK_2{[CERTREQ]} →

← HDR,SK_2{CERT}

IKE_INTERMEDIATE (AUTHKEM)

HDR,SK_2{N(**AUTHKEM_CT,ECERT**)} →

← HDR,SK_2{N(**AUTHKEM_CT,AUTHKEM_SSCONF**)}

IKE_AUTH (7296), $SK_3 = f(SK_s,SSi,SSr)$

HDR,SK_3{IDi,AUTH(**KEM Auth**),SAi2,TSi,TSr} →

← HDR,SK_3{IDr,AUTH(**KEM Auth**),SAr2,TSi,TSr}

We can, but...

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867], AUTHKEM), $SK_1=f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]
[,CERTREQ]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)],
CERT}

IKE_INTERMEDIATE (AUTHKEM), $SK_2=f(SK_1,AKE_1[,PPK])$

HDR,SK_2{N(**AUTHKEM_CT,ECERT**)} →

← HDR,SK_2{N(**AUTHKEM_CT,AUTHKEM_SSCONF**)}

IKE_AUTH (7296), $SK_3 = f(SK_s,SSi,SSr)$

HDR,SK_3{IDi,AUTH(**KEM Auth**),SAi2,TSi,TSr} →

← HDR,SK_3{IDr,AUTH(**KEM Auth**),SAr2,TSi,TSr}

This is better!

Initiator — **Exchange** (RFCs), Session Keys — Responder

IKE_SA_INIT (7296)

HDR,SAi1,KEi,Ni,[N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**) →

← HDR,SAr1,KEr,Nr,[CERTREQ,][N(USE_PPK_ALT),]
N(**INTERMEDIATE_EXCHANGE_SUPPORTED**),
N(**USE_AUTHKEM**)

IKE_INTERMEDIATE (9370 [,9867], AUTHKEM), $SK_1 = f(KE)$

HDR,SK_1{KEi1[,N(PPK_IDENTITY_KEY)]
[,CERTREQ]} →

← HDR,SK_1{KEr1[,N(PPK_IDENTITY)],
ECERT}

Encrypted with SK_2!

IKE_INTERMEDIATE (AUTHKEM), $\mathbf{SK_2} = f(SK_1, AKE_1[, PPK])$

HDR,SK_2{N(**AUTHKEM_CT,ECERT**)} →

← HDR,SK_2{N(**AUTHKEM_CT,AUTHKEM_SSCONF**)}

IKE_AUTH (7296), $SK_3 = f(SK_s, SSi, SSr)$

HDR,SK_3{IDi,AUTH(**KEM Auth**),SAi2,TSi,TSr} →

← HDR,SK_3{IDr,AUTH(**KEM Auth**),SAr2,TSi,TSr}

Thanks!

Comments?

WG adoption?