LAKE Lightweight Authenticated Key Exchange
Internet-Draft
Intended status: Standards Track
Expires: 24 October 2025

U. Blumenthal
B. Luo
S. O'Melia
G. Torres
D. Wilson
MIT

22 April 2025

PQuAKE - Post-Quantum Authenticated Key Exchange
draft-uri-lake-pquake-00

Abstract

This document defines the Post-Quantum Authenticated Key Exchange (PQuAKE) protocol that addresses the needs of bandwidth- and/or power-constrained environments, while maintaining strong security guarantees. It accomplishes that by minimizing the number of bits that need to be exchanged and by utilizing an implicit peer authentication approach similar to Menezes-Qu-Vanstone (MQV) design. This protocol is suitable for integration into protocols that establish dynamic secure sessions, such as Extensible Authentication Protocol (EAP), Internet Key Exchange Version 2 (IKEv2), or Secure COmmunications Interoperability Protocol (SCIP). This protocol has proofs in the verifiers Verifpal and CryptoVerif for security properties such as secrecy of the session key, mutual authentication, identity hiding with a preshared secret, and forward secrecy of the session key. The authors are in the process of publishing the proofs.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://mouse07410.github.io/pquake-draft/draft-uri-lake-pquake.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-uri-lake-pquake/.

Discussion of this document takes place on the Lightweight Authenticated Key Exchange Working Group mailing list (mailto:lake@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/lake/. Subscribe at https://www.ietf.org/mailman/listinfo/lake/.

Source for this draft and an issue tracker can be found at https://github.com/mouse07410/pquake-draft.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Compliance requirements for the components 4
1.2. Mandatory-To-Implement algorithms 5
2. Conventions and Definitions 5
3. Protocol Description 5
3.1. Protocol Diagram 6
3.2. Exchange certificates 8
3.2.1. Key Derivation of k_hid with preshared secret 8
3.2.2. Key Derivation of k_hid without preshared secret . . 8
3.2.3. Initiator 8
3.2.4. Responder 8
3.3. Encapsulate shared secrets 8
3.3.1. Initiator 8
3.3.2. Responder 9
3.4. Decapsulate ciphertexts and key derivation 9

3.4.1. Initiator 9
3.4.2. Responder 9
3.5. Key Confirmation 9
3.5.1. Initiator 9
3.5.2. Responder 9
3.6. Error Handling 10
4. Protocol Messages 10
4.1. Message Format 10
4.2. Hello Messages 11
4.2.1. Initiator Hello 11
4.2.2. Responder Hello 12
4.3. Certificate Exchange 12
4.4. Certificate Format 12
4.5. Encapsulation 13
4.6. Key Confirmation 13
5. Integration into IKEv2 13
5.1. IKE_SA_INIT 14
5.2. IKE_INTERMEDIATE 14
5.3. IKE_AUTH 14
6. Security Considerations 15
7. IANA Considerations 15
8. References 15
8.1. Normative References 15
8.2. Informative References 16
Acknowledgments 16
Authors' Addresses 16

1. Introduction

The primary goal of PQuAKE is to minimize the communication overhead of Post-Quantum (PQ) public-key cryptography during a key exchange. Bandwidth or power limited devices may not realistically use alternative PQ key exchanges such as the TLS handshake protocol to derive a shared symmetric key, as PQ digital signatures and keys are drastically larger. This protocol minimizes the communication overhead by reducing the number of signatures transmitted by each party to one offline-generated certificate signature. It is designed to be an add-on to such protocols as EAP [EAP], IKEv2, and others. Since computing a PQ digital signature typically is more expensive than performing a PQ KEM, there is a benefit of reduced computational costs.

The overall idea of the implicit authentication of the peers comes from the MQV protocol.

Both parties MAY have a pre-shared symmetric secret key, usually distributed among all the members of the given network or Community of Interest (COI). Adding a pre-shared symmetric key to the key

derivation ensures confidentiality of the peers' identities (certificates) against active attackers that do not have that pre-shared key.

The protocol maintains the following security guarantees:

* The secure channel between the Initiator and Responder is mutually authenticated

* Key freshness, aka a new key is generated for this channel, and it hasn't been reused or stale;

* If two parties properly follow the protocol, both parties will compute the same shared keys that are known only to them;

* Perfect Forward Secrecy, aka after the channel is closed, there is no way for an adversary to break security properties associated with the shared key established via this protocol;

* Security against replay attacks;

* Confidentiality of peers' identities against passive adversary;

* Confidentiality of peers' identities against active adversary (aka, Man-In-The-Middle) when both peers utilize long-term pre-shared key.

This protocol has proofs in the verifiers Verifpal and CryptoVerif for security properties such as secrecy of the session key, mutual authentication, identity hiding with a preshared secret, and forward secrecy of the session key. The authors are in the process of publishing the proofs.

It is important to note that PQuAKE does not replace protocols like the TLS record protocol, only the handshake protocol. Other protocols such as IKEv2, SCIP, or EAP may integrate PQuAKE into their key exchange phase.

1.1. Compliance requirements for the components

The building blocks of this protocol SHALL have the following security properties:

* Symmetric Key Cryptosystem - IND-CCA2

* Key Encapsulation Mechanism (KEM) - Implicit, IND-CCA2 and IK-CCA2

* Key Derivation Function 1 - Random Oracle Hash Function

* Key Derivation Function 2 - Random Oracle Hash Function

* Message Authentication Code (MAC) - Pseudorandom Function

1.2. Mandatory-To-Implement algorithms

While this protocol has been formally proven to work with any KEM, MAC, and symmetric cipher that exhibit the above properties -- interoperability requires that a Mandatory-To-Implement (MTI) set of algorithms is specified for the Version 1 of the protocol:

* Enc: AES-GCM-256

* KEM: ML-KEM-1024

* Hash: SHA384

* MAC: HMAC

* KDF: HKDF

* Signature: ML-DSA-87

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Protocol Description

The PQuAKE protocol consists of four steps. Within each step, the exchanges can happen asynchronously, but one step (aka, all of the exchanges of that step) MUST conclude before the peers can proceed to the next one. If any step results in an error, the protocol SHOULD abort. That includes receiving an out-of-order or corrupted message, or not receiving an expected message within the deployer-configured time interval. However, the protocol SHOULD only abort at the end of the protocol if the peer’s identity does not match an out-of-band verification, and an implicit KEM without aborts SHOULD be used. Ideally, the protocol SHOULD only abort after the key confirmation step if the reason for aborting is related to the identities of the two parties.

Currently, no notification to the other party, such as information about an abnormal completion and/or giving details of the error, is included in the protocol.

The four steps are the following:

1. Establish an ephemeral symmetric key via hello messages and exchange encrypted certificates (one MUST use an encryption scheme with IND-CCA2 security and an implicit KEM with IND-CCA2 security and IK-CCA2 security).

2. Encapsulate shared secrets and exchange the ciphertexts.

3. Decapsulate shared secrets, derive key confirmation keys and a session key.

4. Perform Key Confirmation.

Note that both peers take full transcripts (chain-hashes) of all the messages they send and receive, and include the resulting hash outputs among the inputs of the Key Derivation Function (KDF) that gets invoked to generate shared secrets (first for encrypting certificates, and the next time - to provide the shared secret that is the purpose of this protocol).

While both parties have to share their certificates or identities for authentication, we assume the identities of each party shall remain confidential to those outside of this exchange. They encrypt their certificates with a shared symmetric ephemeral key that they generate via a ephemeral KEM. This key is used to encrypt the certificate and is an input to the KDF when generating the shared key. The final KDF that provides the negotiated shared secret, will also include this symmetric key in its input.

Instead of generating a signature over the handshake transcript like TLS, PQuAKE performs an implicit authentication of the handshake. It does this by making the protocol’s session key dependent on not only a series of KEM calculated shared secrets but also dependent on the hashes of the sent and received messages. Since only their corresponding party, who they have authenticated, can know those secrets, deriving the same session key implicitly authenticates each other while creating a shared secret.

3.1. Protocol Diagram

```
   Initiator                                                  Responder
   ------------------------------------------------------------------

   Establish confidential link and exchange certificates
   -----------------------------------------------------
   (sk_e, pk_e) <- KEM.keygen()

   { pk_e } -->
                                      (ct_e, ss_e) <- KEM.encap(pk_e)

                                                        <-- { ct_e }

   ss_e <- KEM.decap(ct_e, sk_e)

   k_hid <- kdf_1(ss_pre, ss_e || "HID")
                                    k_hid <- kdf(ss_pre, ss_e || "HID")

   e_cert_i <- Enc(k_hid, cert_i)       e_cert_r <- Enc(k_hid, cert_r)

   { e_cert_i } -->                                    <-- { e_cert_r }


   Encapsulate and send shared secrets
   -----------------------------------
   cert_r <- Dec(k_hid, e_cert_r)       cert_i <- Dec(k_hid, e_cert_i)

   if cert_r is not valid, abort         if cert_i is not valid, abort

   (ct_i, ss_i) <- KEM.encap(pk_r)     (ct_r, ss_r) <- KEM.encap(pk_i)

   { ct_i } -->                                        <-- { ct_r }


   Decapsulate shared secrets and derive session keys
   --------------------------------------------------
   ss_r <- KEM.decap(sk_i, ct_r)         ss_i <- KEM.decap(sk_r, ct_i)

   send_hash <- H(hf, pk_e, e_cert_i, ct_i)
                              send_hash <- H(hf, ct_e, e_cert_r, ct_r)

   recv_hash <- H(hf, ct_e, e_cert_r, ct_r)
                              recv_hash <- H(hf, pk_e, e_cert_i, ct_i)

   S <- ss_e||ss_i||ss_r||send_hash||recv_hash
                           S <- ss_e||ss_i||ss_r||recv_hash||send_hash

   k_C_i, k_C_r, ... <- kdf_2(hf2, S)
                                    k_C_i, k_C_r, ... <- kdf_2(hf2, S)
```

```
Key Confirmation
----------------
{ HMAC(k_C_i, recv_hash || send_hash) } -->
                        <-- { HMAC(k_C_r, send_hash || recv_hash) }
```

3.2. Exchange certificates

During this step, both nodes establish a shared ephemeral key via a KEM, then use it to encrypt certificates before transmitting them.

The initiator generates an ephemeral key and transmits the encapsulated secret. The responder MUST decapsulate the ciphertext. Both parties MUST derive a shared ephemeral key from the encapsulated secret and the pre-shared secret if it is present. Both parties MUST encrypt and transmit their certificates. Both parties MUST validate the certificate's signature. If the verification of a signature fails, the protocol MUST abort. Implementors need to be careful to avoid aborting based off the other node's identity until the end of the protocol to maintain identity hiding of the peer. Note that key encapsulation mechanism SHOULD be IND-CCA2 and IK-CCA2 secure and SHOULD NOT abort (it SHOULD be an implicit KEM).

3.2.1. Key Derivation of k_hid with preshared secret

```
k_hid <- kdf_1(ss_pre, ss_e || "HID");
```

3.2.2. Key Derivation of k_hid without preshared secret

```
k_hid <- kdf_1(ss_e, "HID");
```

3.2.3. Initiator

```
e_cert_i <- E(k_hid, cert_i);
```

3.2.4. Responder

```
e_cert_r <- E(k_hid, cert_r);
```

3.3. Encapsulate shared secrets

During this step, both nodes generate an encapsulated secret via a KEM. The ciphertexts are exchanged.

3.3.1. Initiator

```
(ct_r, ss_r) <- KEM.encap(pk_i);
```

3.3.2. Responder

```
(ct_i, ss_i) <- KEM.encap(pk_r);
```

3.4. Decapsulate ciphertexts and key derivation

The ciphertexts are decapsulated by both parties. At this point, both sides have all the shared secrets required to derive a set of session keys.

3.4.1. Initiator

```
send_hash <- hash(pk_e, e_cert_i, ct_i);

recv_hash <- hash(ct_e, e_cert_r, ct_r);

transcript <- (send_hash, recv_hash);

k_C_i, k_C_r, k_session = kdf_2(ss_e, ss_i, ss_r, transcript);
```

3.4.2. Responder

```
send_hash <- hash(ct_e, e_cert_r, ct_r);

recv_hash <- hash(pk_e, e_cert_i, ct_i);

transcript <- (recv_hash, send_hash);

k_C_i, k_C_r, k_session = kdf_2(ss_e, ss_i, ss_r, transcript);
```

3.5. Key Confirmation

Key confirmation is done by calculating an HMAC of the chain-hash of all the sent and received messages correspondingly, using the appropriate Key Confirmation key derived in step 3. The initiator MUST use the key K~ir~, and the responder MUST use the key K~ri~.

3.5.1. Initiator

```
C_i <- HMAC(k_C_i, send_hash || recv_hash);
```

3.5.2. Responder

```
C_r <- HMAC(k_C_r, recv_hash || send_hash);
```

3.6. Error Handling

We make no assumptions about the underlying transport that carries PQuAKE messages, because no error - whether maliciously introduced or accidental - in any of its messages can impact correctness of the protocol itself. We consider two kinds of errors:

a. Corruption of a message - will result in protocol failure (abort)

b. Failure to receive a message within expected time interval, aka timeout - will result in protocol failure (abort).

Handling the protocol timeout (how long to wait until declaring failure to receive) is the responsibility of the implementation deployer. The implementer SHOULD make this value configurable.

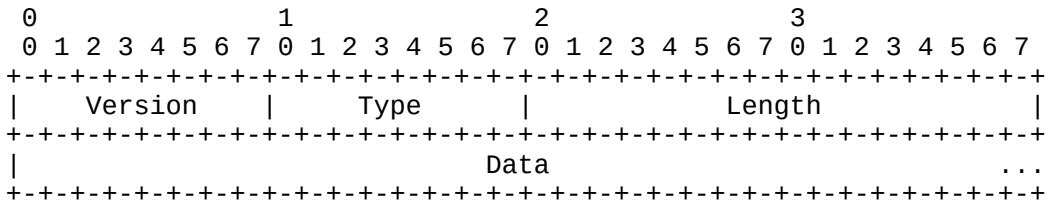
Note: the more the underlying transport does to detect or mitigate line errors (aka, non-malicious errors), the likelier the protocol is to successfully complete. Ideally, the transport would offer at least the capabilities of UDP.

4. Protocol Messages

We do not include explicit checksums because it is practically impossible for the protocol to succeed if any message would arrive corrupted, either maliciously, or by a random error.

4.1. Message Format

A message of the protocol is formatted as follows:



Version: 8 bits

The version field indicates the format of the initiator hello message. This document describes version 1.

Type: 8 bits

This field indicates the current step of the protocol.

Length: 16 bits

Length is the length of the data, measured in octets. This field allows the length of the data to be up to 65535 octets.

Data: variable

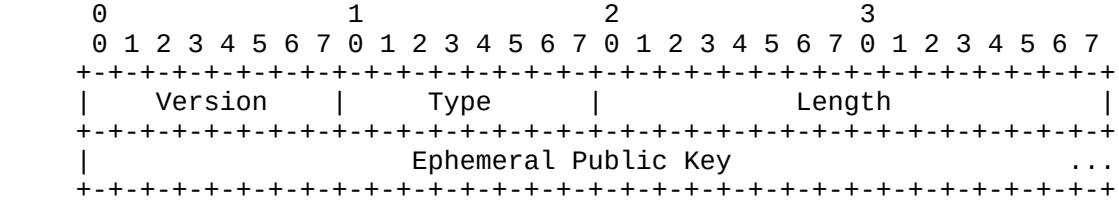
This field changes based on the current step of the protocol.

4.2. Hello Messages

The Initiator generates an ephemeral KEM key-pair (sk_e, pk_e) = KEM.keygen() and sends the public key pk_e to its intended recipient (the Responder). The Responder performs encapsulation (ct_e, ss_e) = KEM.encap(pk_e) and sends the ciphertext ct_e to the Initiator.

4.2.1. Initiator Hello

An Initiator Hello message is formatted as follows:



Version: 8 bits

The version field indicates the format of the initiator hello message. This document describes version 1.

Type: 1

This field indicates the state of the protocol.

Length: 16 bits

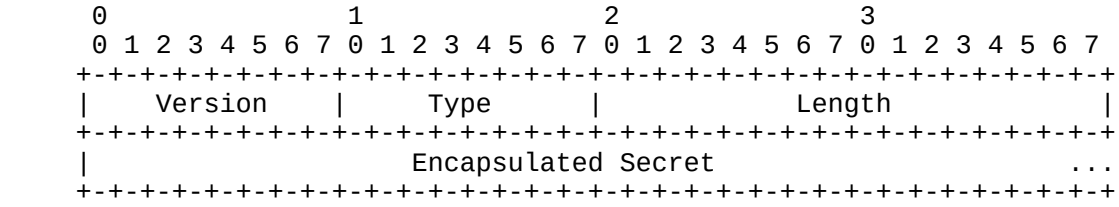
Length is the length of the ephemeral public key, measured in octets. This field allows the length of a ephemeral public key to be up to 65535 octets.

Ephemeral Public Key: variable

This field SHOULD be unique for each protocol execution.

4.2.2. Responder Hello

A Responder Hello message is formatted as follows:



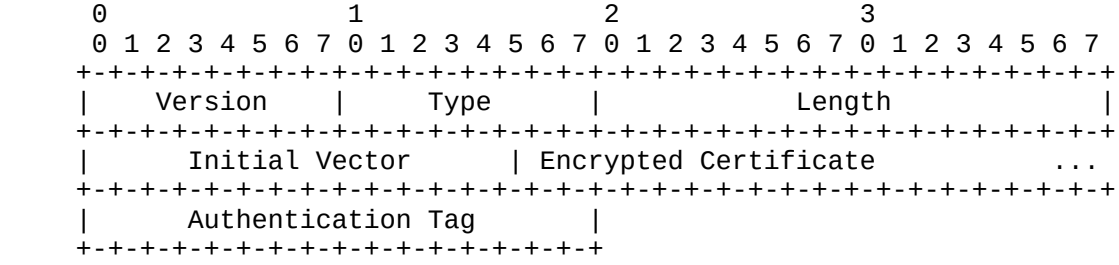
Type: 2

Encapsulated Secret: variable

The size of this field depends on the key encapsulation mechanism used.

4.3. Certificate Exchange

A Certificate Exchange message is formatted as follows:



Type: 3 for initiator, 4 for responder

Initial Vector: 96 bits

Encrypted Certificate: variable

Authentication Tag: 128 bits

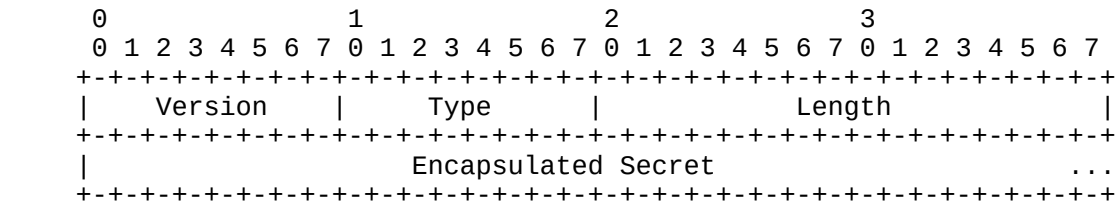
The certificate encrypted with the derived key k_hid.

4.4. Certificate Format

For now, we use standard X.509 certificate [X.509] with OIDs specifying ML-KEM and ML-DSA correspondingly. Future extensions may use different certificate formats.

4.5. Encapsulation

An Encapsulation message is formatted as follows:



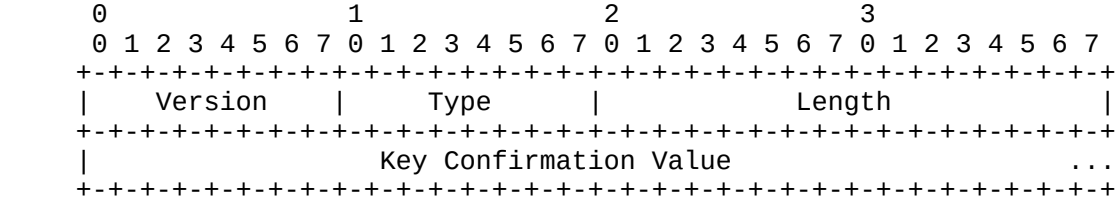
Type: 5 for initiator, 6 for responder

Encapsulated Secret: variable

The size of this field depends on the key encapsulation mechanism used.

4.6. Key Confirmation

A Key Confirmation message is formatted as follows:



Type: 7 for initiator, 8 for responder

Key Confirmation Value: 384 bits (output size of SHA384)

This field is the computed HMAC value of the exchange transcript.

5. Integration into IKEv2

Integration into IKEv2 [IKEV2] results in a hybrid Post-Quantum Key Exchange.

During the IKE_INIT phase, the peers establish ephemeral shared secret key via ECDHE (Ephemeral Diffie-Hellman key exchange), and signal that they will use PQuAKE in the IKE_INTERMEDIATE phase.

Messages exchanged during IKE_INTERMEDIATE phase, perform the actual key exchange.

Instead of explicit signatures during IKE_AUTH phase, both peers exchange Key Confirmation messages here.

5.1. IKE_SA_INIT

The first pair of messages negotiate cryptographic algorithms, exchange nonces, and do an elliptic curve Diffie-Hellman exchange in order to maintain compatibility with standard IKEv2. The initiator indicates its support for Intermediate Exchange by including a notification of type INTERMEDIATE_EXCHANGE_SUPPORTED in the IKE_SA_INIT request message. If the responder also supports this exchange, it includes this notification in the response message.

The IKE_SA_INIT exchange is as follows:

```
Initiator                                                Responder
-----------------------------------------------------------------
HDR, SAi1, KEi, Ni,
N(PQUAKE_SUPPORTED),
N(INTERMEDIATE_EXCHANGE_SUPPORTED) --->
                                            HDR, SAr1, KEr, Nr,
                                           N(PQUAKE_SUPPORTED),
                          <--- N(INTERMEDIATE_EXCHANGE_SUPPORTED)
```

5.2. IKE_INTERMEDIATE

If both peers indicated their support for the Intermediate Exchange, the initiator may proceed with the PQuAKE key exchange.

```
Initiator                                                Responder
-----------------------------------------------------------------
HDR, SK { PQUAKE_INITIATOR_HELLO } -->

                           <-- HDR, SK { PQUAKE_RESPONDER_HELLO }

HDR, SK { PQUAKE_CERT_EXCHANGE_I } -->

                           <-- HDR, SK { PQUAKE_CERT_EXCHANGE_R }

HDR, SK { PQUAKE_KEY_ENCAP_I } -->

                                <-- HDR, SK { PQUAKE_KEY_ENCAP_R }
```

5.3. IKE_AUTH

The last pair of messages (IKE_AUTH) authenticate the previous messages and establish the first Child SA.

```
Initiator                                                Responder
-------------------------------------------------------------------
HDR, SK { PQUAKE_KEY_CONFIRMATION_I } -->

                        <-- HDR, SK { PQUAKE_KEY_CONFIRMATION_R }
```

6. Security Considerations

This is a security protocol, and it holds the properties described in (TODO reference) in the presence of passive or active attacker on the network.

One potential concern is the confidentiality of the peers' identities carried in their certificates. An active attacker can learn their identities during the certificate exchange step. Using a pre-shared secret will prevent disclosure of these certificates, keeping peers identities confidential. Since there are costs associated with out-of-band distribution of that secret, it would be typically shared among the Community of Interest (CoI). In that case, this protocol would protect peers identities against active attackers outside of this Community of Interest, but not against an active attacker that is a member of CoI.

7. IANA Considerations

This document defines a new Notify Message Type in the "IKEv2 Notify Message Types - Status Types" registry:

<TBA> PQUAKE_SUPPORTED

8. References

8.1. Normative References

[IKEV2] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/rfc/rfc7296>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[X.509] Housley, R., Ford, W., Polk, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, DOI 10.17487/RFC2459, January 1999, <https://www.rfc-editor.org/rfc/rfc2459>.

8.2. Informative References

[EAP] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/rfc/rfc5247>.

[MQV] Law, L., Menezes, A., Qu, M., Solinas, J., and S. Vanstone, "An Efficient Protocol for Authenticated Key Agreement", DOI 10.1023/A:1022595222606, 1998, <https://doi.org/10.1023/A:1022595222606>.

Acknowledgments

The authors want to thank Roger Khazan (MIT/LL), Adam Margetts (MIT/LL) for reviewing this work and giving helpful suggestions.

Authors' Addresses

Uri Blumenthal
MIT
United States of America
Email: uri@ll.mit.edu

Brandon Luo
MIT
United States of America
Email: brandon.luo@ll.mit.edu

Sean O'Melia
MIT
United States of America
Email: sean.omelia@ll.mit.edu

Gabriel Torres
MIT
United States of America
Email: gabriel.torres@ll.mit.edu

David A. Wilson
MIT
United States of America
Email: david.wilson@ll.mit.edu