Post Quantum Cryptography integration in Linphone

Johan Pascal

September 20, 2022
Version 1.0

Contents

1 Changelog

Revision History

Revision	Date	Author(s)	Description
1.0	September 20, 2022	JP	Initial version

2 Introduction

The mathematical problems of integer factorisation and discrete logarithms over finite fields or elliptic curves underpin most of the asymmetric algorithms used for key establishment and digital signatures on the internet. These problems, and hence the algorithms based on them, will be vulnerable to attacks using Shor's Algorithm on a sufficiently large general-purpose quantum computer, known as a Cryptographically Relevant Quantum Computer (CRQC). It is difficult to predict when, or if, such a device will exist. However, it is necessary to defend against this possibility. Data encrypted today with an algorithm vulnerable to a quantum computer could be stored for decryption by a future attacker with a CRQC.(Driscoll, 2022)

This document describes the integration of Post-Quantum Cryptography (PQC) in the Linphone SDK and details the modifications to the ZRTP protocol(Zimmermann et al., 2011) made to allow the use of PQC.

3 Notations

$A \| B$ denotes the concatenation of byte sequences A and B

4 Post-Quantum cryptography

4.1 Open Quantum Safe library

Linphone uses *liboqs*(Stebila and Mosca, 2017) as provider for Post-Quantum cryptography implementation. *liboqs* is a project maintained by Open Quantum Safe. *liboqs* puts together the code source from the official repositories of various PQC algorithms both key exchange and signature algorithms. *liboqs* :

- is frequently updated
- has a large community and is supported by major actors
- provides a consistent API and build scripts for all the NIST PQC standisation candidates
- build scripts' allow algorithm selection

In the current context, the Linphone SDK requires only key exchange PQC. No signature algorithms are needed.

As of August 2022, *liboqs* provides the following Key Encapsulation Mechanism (KEM) algorithms:

- BIKE
- Classic McEliece
- FrodoKEM
- HQC
- Kyber
- NTRU
- NTRU-Prime
- Saber

Kyber being the KEM algorithm first selected to be standardised by the NIST, linphone currently supports Kyber and HQC, specifically the variants Kyber512 and Kyber1024, HQC128 and HQC256. Other suitable, see 5.4, algorithms provided by *liboqs* can be included without major changes to the architecture.

4.2 Hybrid KEM

During the transition from traditional to post-quantum algorithms, it is recommended to combine both types of algorithm. The construction that combines a traditional key exchange with post-quantum key exchanges into a single key exchange is known as hybrid key exchange. We define a hybrid key exchange providing an interface identical to a simple KEM:

Algorithm 1 KEM interface

function KEMGENKEY
  return $publicKey, secretKey$ ▷ Generate and return a public key and a secret key
end function

function KEMENCAPS($publicKey$)
  return $sharedSecret, cipherText$ ▷ Generate a shared secret and encapsulate it in a ciphertext
end function

function KEMDECAPS($secretKey, cipherText$)
  return $sharedSecret$ ▷ Retrieve the shared secret from the ciphertext
end function

4.2.1 ECDH-based KEM

The first step is to build a KEM using a traditional ECDH. This is performed using the method described in the RFC9180 (Barnes et al., 2022) section 4.1.

Algorithm 2 ECDH interface

function ECDHGENKEY
  return $publicKey, secretKey$ ▷ Generate and return a public key and a secret key
end function

function ECDHCOMPUTESHARED($selfSecretKey, peerPublicKey$)
  return $sharedSecret$ ▷ Generate a shared secret from self secret key and peer public key
end function

function ECDHDERIVEPUBLICKEY($secretKey$)
  return $publicKey$ ▷ Derive a public key from the secret one
end function

Algorithm 3 ECDH-based KEM

function KEMGENKEY
  return ECDHgenKey ▷ Returns public and secret key generated by ECDH
end function

function KEMENCAPS($publicKey$)
  $pkE, skE \leftarrow$ ECDHGENKEY ▷ generate an ephemeral key pair
  $ssE \leftarrow$ ECDHCOMPUTESHARED($skE, publicKey$)
  $ss \leftarrow$ HKDF($ssE, pkE \| publicKey$) ▷ HKDF as defined in RFC5869
  return ss, pkE ▷ returns the ephemeral public key as cipher text
end function

function KEMDECAPS($cipherText, secretKey$)
  $ssE \leftarrow$ ECDHCOMPUTESHARED($secretKey, cipherText$) ▷ cipherText is the pkE
  $publicKey \leftarrow$ ECDHDERIVEPUBLICKEY($secretKey$)
  return HKDF($ssE, cipherText \| publicKey$) ▷ HKDF as defined in RFC5869
end function

Linphone SDK produces two variants of ECDH-based KEM. One from X25519, the other from X448 described in (Langley et al., 2016). X22519 and X448 implementation is provided by *libdecaf* (Hamburg, 2014).

4.2.2 KEM combiner

Section 3.3 in (Bindel et al., 2019) describes a way of combining several KEMs into one. We apply this to build an hybrid KEM from two or more KEMs using HMAC-SHA as dual Pseudo Random Function and extractor. PublicKey, secretKey and cipherText sizes are implicitly known for each KEM algorithm, so the function SPLIT can separate the concatenated entities.

The following pseudo code combines n KEMs together, each component is noted KEM_i, with i in range $1, n$.

Algorithm 4 Combined KEMs

function KEMGENKEY
  for $i \leftarrow 1, n$ **do**
    $pk_i, sk_i \leftarrow \text{KEM}_i\text{GENKEY}$
  end for
  return $pk_1 \| .. \| pk_n, sk_1 \| .. \| sk_n$ ▷ returns a concatenation of both keys
end function

function KEMENCAPS($publicKey$)
  $pk_1 \| .. \| pk_n \leftarrow \text{SPLIT}(publicKey)$ ▷ split the public key into its components

  for $i \leftarrow 1, n$ **do** ▷ generate secret and encapsulate it for each component
    $ss_i, ct_i \leftarrow \text{KEM}_i\text{ENCAPS}(pk_i)$
  end for
  $cipherText \leftarrow ct_1 \| .. \| ck_n$

  $k_1 \leftarrow \text{HMAC-SHA}(, ss1)$ ▷ Derive the shared secret from secrets and a transcript
  for $i \leftarrow 2, n$ **do**
    $k_i \leftarrow \text{HMAC-SHA}(k_{i-1}, ss_i)$
  end for
  $sharedSecret \leftarrow \text{HMAC-SHA}(k_n, cipherText)$

  return $sharedSecret, cipherText$
end function

function KEMDECAPS($cipherText, secretKey$)
  $sk_1 \| .. \| sk_n \leftarrow \text{SPLIT}(secretKey)$ ▷ retrieve secret key and cipher text components
  $ct_1 \| .. \| ct_n \leftarrow \text{SPLIT}(cipherText)$

  for $i \leftarrow 1, n$ **do** ▷ retrieve encapsulated secret for each component
    $ss_i \leftarrow \text{KEM}_i\text{DECAPS}(ct_i, sk_i)$
  end for

  $k_1 \leftarrow \text{HMAC-SHA}(, ss1)$ ▷ Derive the shared secret from secrets and a transcript
  for $i \leftarrow 2, n$ **do**
    $k_i \leftarrow \text{HMAC-SHA}(k_{i-1}, ss_i)$
  end for
  $sharedSecret \leftarrow \text{HMAC-SHA}(k_n, cipherText)$

  return $sharedSecret$
end function

Linphone provides several hybrid KEMs:

- two algorithms combined: X25519/Kyber512, X25519/HQC128, X448/Kyber1024, X448/HQC256
- three algorithms combined: X25519/Kyber512/HQC128, X448/Kyber1024/HQC256

Kyber and HQC are based on different mathematical problems, hence the interest of combining them, with the classic ECDH exchange, should one of them be broken in the future.

4.3 Enable PQC in linphone SDK

liboqs is linked to a module called PostQuantumCryptoEngine. To build the requested part of *liboqs* and this module, call the linphone-sdk build command with the option

```
-DENABLE_PQCRYPTO=On
```

5 Audio and video calls

Linphone supports three protocols to initiate a SRTP protected audio/video call: SDES, DTLS-SRTP and ZRTP. We recommend the later for a higher level of confidentiality. The PQC algorithm is available only when using ZRTP(Zimmermann et al., 2011).

5.1 ZRTP

ZRTP is a protocol based on Diffie-Hellman to agree on a session key and parameters for establising a SRTP session. Several features make this protocol secure, among them key continuity and Man-in-the-Middle detection.

In particular the Man-in-the-Middle(MitM) detection feature is based on commitment of an endpoint to provide specific public key material in the next protocol packet. Focusing on the Diffie-Hellman exchange at the core of the protocol(see fig.1 for a complete data flow diagram) we have:

- Bob commits to use a specific DH public key without revealing it (*hash value* in the Commit packet).
- Alice provides her own public key to Bob(*pvr* in the DHPart1 packet). Bob can compute the shared secret using Alice public key and can derive the master key.
- Bob provides his public key(*pvi* in the DHPart2 packet) to Alice in the last packet of the transcripted sequence used to derive the master key.
- Alice checks the public key provided by Bob is the one he commited to use and compute the shared secret.
- on both endpoint the DH shared secret is derived including a transcript of the whole protocole exchange into the master key.
- from the master key is derived a Short Authentication String(SAS) compared vocally between the two endpoints, if the SAS are not matching, it indicates an ongoing MitM. The SAS is short so it is easy to vocally compare but it makes it highly sensitive to collision attack.

This central part of the protocol achieves two functions:

- a secure exchange a shared secret: a passive opponent is not able to retrieve the master key. This is provided by the Diffie-Hellman exchange.
- prevent a collision attack on the SAS by an active opponent. This is provided by the commitment on the public key used by Bob. To perform a SAS collision, a malicious opponent must find a way to have two ZRTP exchanges leading to the same SAS. It boils down to be in position to compute the master key while still being able to modify it. When Bob receives Alice's public key, he has access to all the material to compute the master key. By commiting to use a specific public key before receiving Alice's one, only one value of the master key can be reached making the collision attack on SAS very hard to achieve.

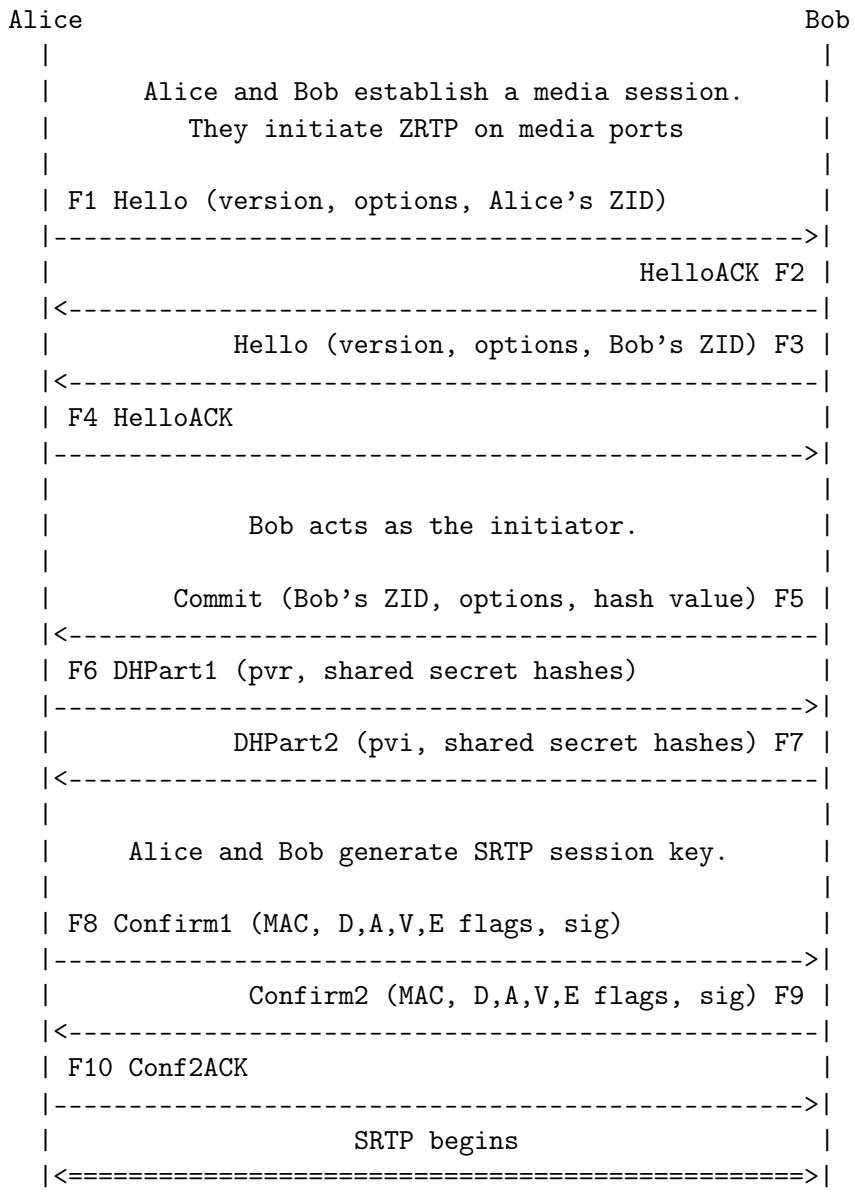

Figure 1: Diffie-Hellman mode ZRTP complete data flow

5.2 KEM version

PQC key exchange algorithm call for candidates from the NIST imposed the usage of a KEM interface. To our purpose, the major difference between DH and KEM interface is that in a DH key exchange, both parties have an exactly symmetric role which is not the case with a KEM scheme.

In the Diffie-Hellman ZRTP exchange, Alice and Bob can compute their DH key pair before hand, so Bob can commit to use a public key without revealing anything of it. The KEM scheme cannot straightforwardly substitute the DH one as one party must have access to the other's public key in order to start the protocol.

To provide the same central properties we had to design a variant of the ZRTP protocol adapted to KEM scheme, the KEM mode. Focusing on the KEM exchange at the core of the protocol(see fig.2 for a complete data flow diagram) we have:

- Bob provides his public key(*pvi* in the Commit packet) and commits to use a specific nonce without revealing it(*hash value* in the Commit packet).
- Alice encapsulates a shared secret in a ciphertext(*pvr* in KEMPart1 packet) using Bob's public key.
  - Alice has access to the shared secret but she cannot compute the master key as she does not have access to the whole exchange transcript.
  - Upon ciphertext reception, Bob can compute the shared secret and derive the master key.
- Bob provides the nonce he commited to(*ni* in the KEMPart2 packet) to Alice in the last packet of the transcripted sequence used to derive the master key.
- Alice checks the nonce provided by Bob is the one he commited to use and compute the master key.
- from the master key is derived a Short Authentication String(SAS) compared vocally between the two endpoints, if the SAS are not matching, it indicates an ongoing MitM. The SAS is short so it is easy to vocally compare but it makes it highly sensitive to collision attack.

This central part of the protocol achieves the two same functions as the DH mode:

- a secure exchange a shared secret: a passive opponent is not able to retrieve the master key. This is provided by the KEM.
- prevent a collision attack on the SAS by an active opponent. This is provided by the commitment on the nonce used by Bob. To perform a SAS collision, a malicious opponent must find a way to have two ZRTP exchanges leading to the same SAS. It boils down to be in position to compute the master key while still being able to modify it.
  - When Alice encapsulates the shared secret using Bob's public key, she cannot derive the master key as she does not have access to the KEMPart2 packet yet.
  - When Bob receives Alice's ciphertext, he has access to all the material to compute the master key. By commiting to use a specific nonce in the KEMPart2 packet before receiving Alice's ciphertext, only one value of the master key can be reached making the collision attack on SAS very hard to achieve.

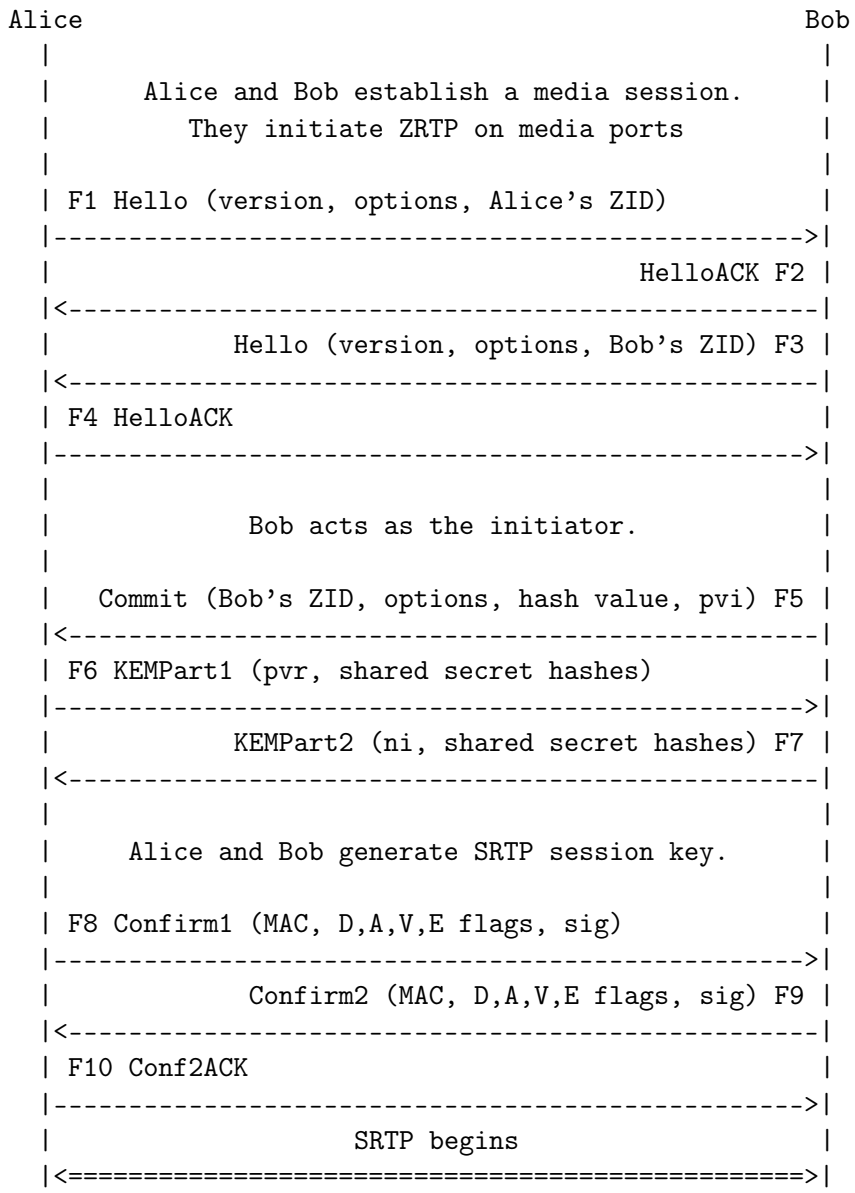

Figure 2: KEM mode ZRTP complete data flow

5.3 ZRTP packet fragmentation

PQC key exchanges produce large public values(public key, cipher text) to be exchanged with the other endpoint. UDP datagrams are often limited to a maximum of 1500 bytes if UDP fragmentation is not desired. In order to compensate for this limitation we introduced a message fragmentation mechanism. ZRTP messages can be fragmented over several zrtp packets.

Multiplexing scheme standard(Petit-Huguenin and Salgueiro, 2016) section 7 reserves for the ZRTP packet format the values 16 to 19 on the first byte to clearly distinguish ZRTP packets from STUN, DTLS, TURN or RTP/RTCP packets.

ZRTP standard as specified in (Zimmermann et al., 2011) uses value 16 as the first byte of ZRTP packet. The ZRTP packet header fields (Sequence Numnber, Magic Cookie, Source Identifier and CRC) description can be found in RFC6189 section 5.

We introduce the use of value 17 as first byte of ZRTP packet to distinguish packets carrying a ZRTP message fragment. Packets holding message fragment get additional fields in the packet header:

- message Id: a unique Id for this message, is attached to the message and is not incremented at each retransmission like the sequence number. It is initialised to a random value and is incremented for each new message generated.
- message total length: size, in 32-bit words of the total message.
- offset: offset of this fragment, in 32-bit words.
- fragment length: size of this fragment, in 32-bit words.

The fragmentation mechanism is opportunistic: A ZRTP exchange using a key agreement algorithm not requiring large public values will use only regular packets and is thus fully compatible with the ZRTP version 1.10 described in RFC6189.

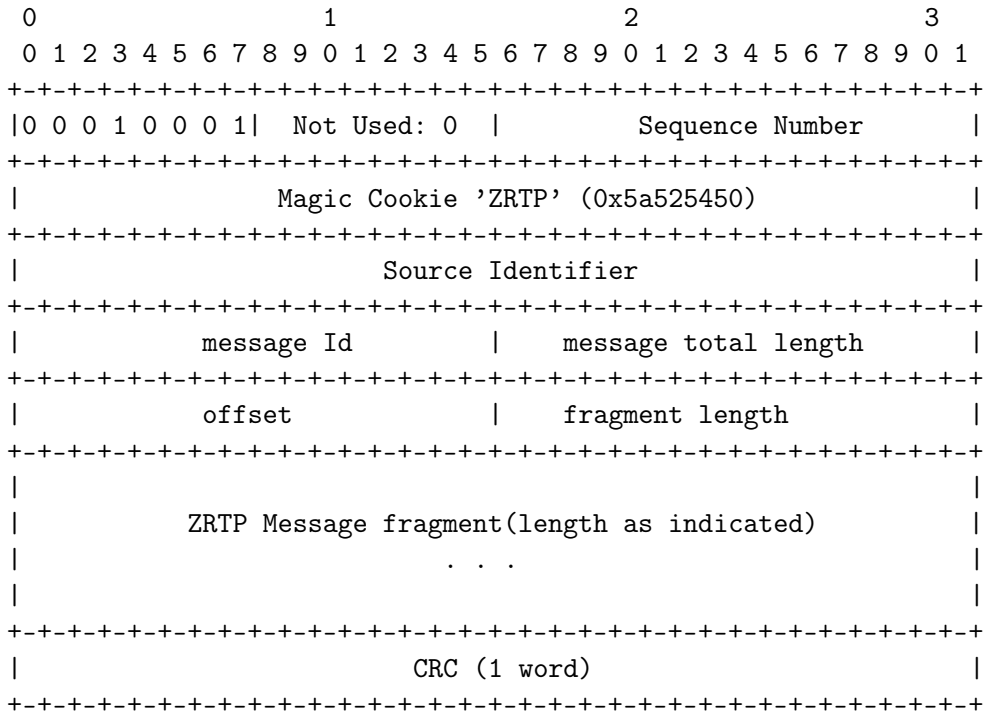

Figure 3: ZRTP packet format for fragmented ZRTP message

5.4 Suitable PQC algorithm properties

ZRTP impose some additional constraints on the choice of PQC KEM algorithm. As public key and cipher text are sent on the network at each call establishment, we must focus on small sized public keys and cipher text. This rules out for example Classic Mc Eliece. Another constraint is to generate a set of key, encapsulate and decapsulate also at each call establishment. These operations must be performed on mobile devices with sometime limited computation power.

6 Instant messaging

Instant messaging is end2end encrypted using Lime(Pascal, 2019), a Signal protocol derivative. This protocol relies on Elliptic-Curve Diffie-Hellman and EdDSA properties and no version using post-quantum cryptography is available yet.

Instant messaging encryption is not encrypted using post-quantum cryptography, this is left for future work.

7 References

Barnes, Richard, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood (2022), "Hybrid Public Key Encryption." RFC 9180, URL https://www.rfc-editor.org/info/rfc9180.

Bindel, Nina, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila (2019), "Hybrid key encapsulation mechanisms and authenticated key exchange." In *Proc. 10th International Conference on Post-Quantum Cryptography (PQCrypto)*, 206–226, URL https://eprint.iacr.org/2018/903.pdf.

Driscoll, Florence (2022), "Terminology for Post-Quantum Traditional Hybrid Schemes." Internet-Draft draft-driscoll-pqt-hybrid-terminology-00, Internet Engineering Task Force, URL https://datatracker.ietf.org/doc/draft-driscoll-pqt-hybrid-terminology/00/. Work in Progress.

Hamburg, Mike (2014), "Ed448-goldilocks." URL https://sourceforge.net/projects/ed448goldilocks/.

Langley, Adam, Mike Hamburg, and Sean Turner (2016), "Elliptic Curves for Security." RFC 7748, URL https://www.rfc-editor.org/info/rfc7748.

Pascal, Johan (2019), "Linphone Instant Message Encryption v2.0." URL https://gitlab.linphone.org/BC/public/lime/blob/master/lime.pdf.

Petit-Huguenin, Marc and Gonzalo Salgueiro (2016), "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)." RFC 7983, URL https://www.rfc-editor.org/info/rfc7983.

Stebila, Douglas and Michele Mosca (2017), "Post-quantum key exchange for the Internet and the Open Quantum Safe project." URL https://github.com/open-quantum-safe/liboqs.

Zimmermann, Philip, Alan Johnston, and Jon Callas (2011), "ZRTP: Media Path Key Agreement for Unicast Secure RTP." RFC 6189, URL https://www.rfc-editor.org/info/rfc6189.