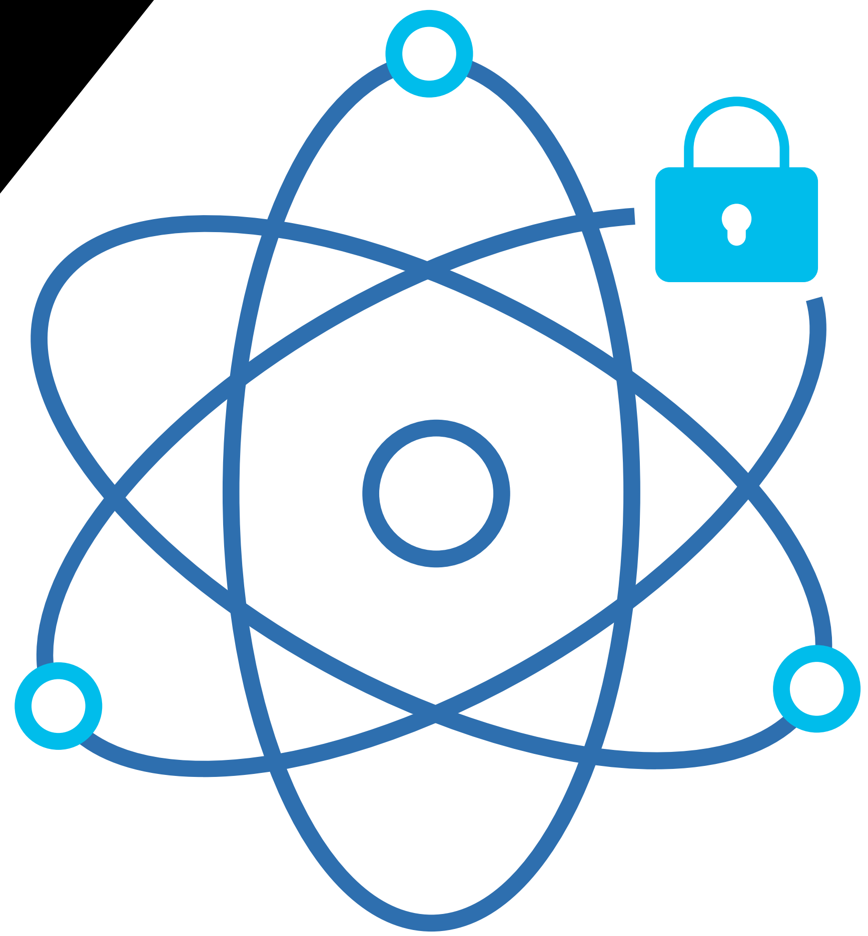


Post Quantum Cryptography in Voice/Video over IP

presented by Johan Pascal

February 4th, 2023

linphone
open source VOIP project

Agenda

I. Context

II. ZRTP overview

III. Post Quantum Key Exchange Mechanism (KEM)

IV. ZRTP adaptation

V. Hybrid KEM

VI. Focus and Conclusion

I. CONTEXT

Quick intro

Linphone

- Is around since 2001
- Is available on GNU/Linux, android, iOS, Windows, Mac
- Uses SIP standards for audio, video and instant messaging
- Secure group messaging using a Signal protocol derivative

Linphone's team also provides

- Flexisip, an open source SIP Proxy
- A free SIP service sip.linphone.org

Media stream encryption

Media Stream encryption: SRTP

- Authenticated Encryption
- AES128,192,256 - Counter Mode or GCM
- RFCs 3711, 6188, 7714
- Requires an external key management

Media stream encryption

Media Stream encryption: SRTP

- Authenticated Encryption
- AES128,192,256 - Counter Mode or GCM
- RFCs 3711, 6188, 7714
- Requires an external key management

SRTP key management

- SDES (RFC4568): key exchanged in SDP
  - SIP proxy can decrypt media streams
- DTLS-SRTP (RFC5764): key exchanged during a DTLS handshake
  - Requires PKI
- ZRTP (RFC6189): key exchange based on (EC)DH
  - No trusted third party required but vocal short authentication string (SAS) comparison

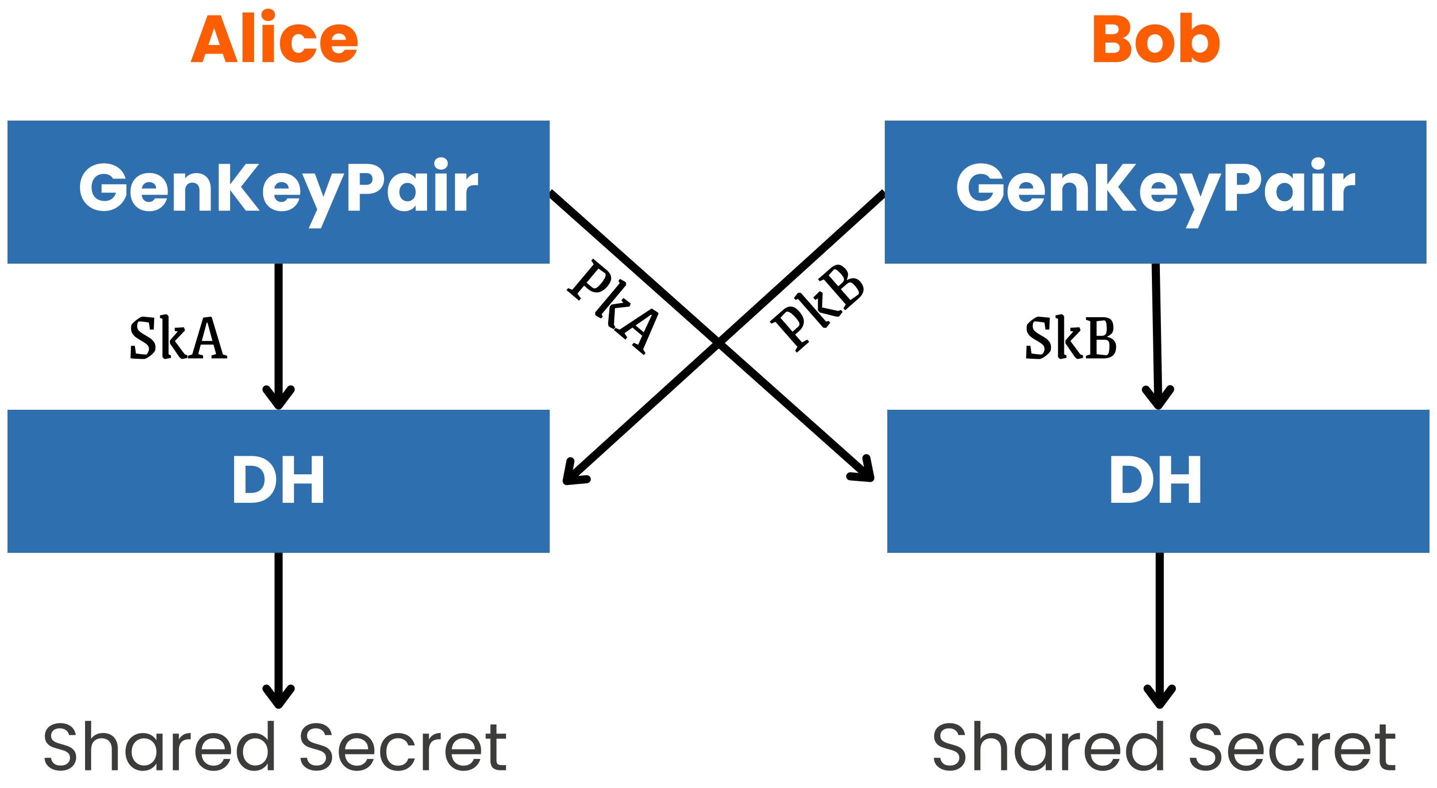
II. ZRTP OVERVIEW

ZRTP overview

- RFC6189, April 2011
- handshake performed on the media stream (over UDP)
- Provides key continuity feature and MitM attack detection
- Based on Diffie-Hellman key exchange

Diffie-Hellman

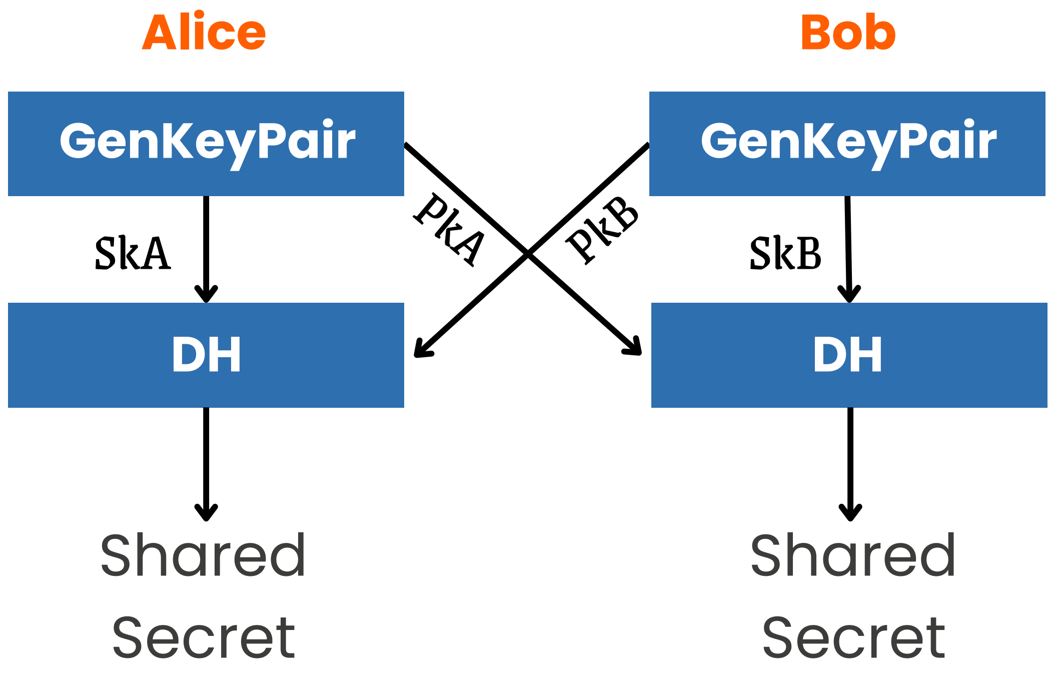
- *SecretKey, PublicKey = GenKeyPair()*
- *SharedSecret = DH(SelfSecretKey, PeerPublicKey)*



ZRTP overview

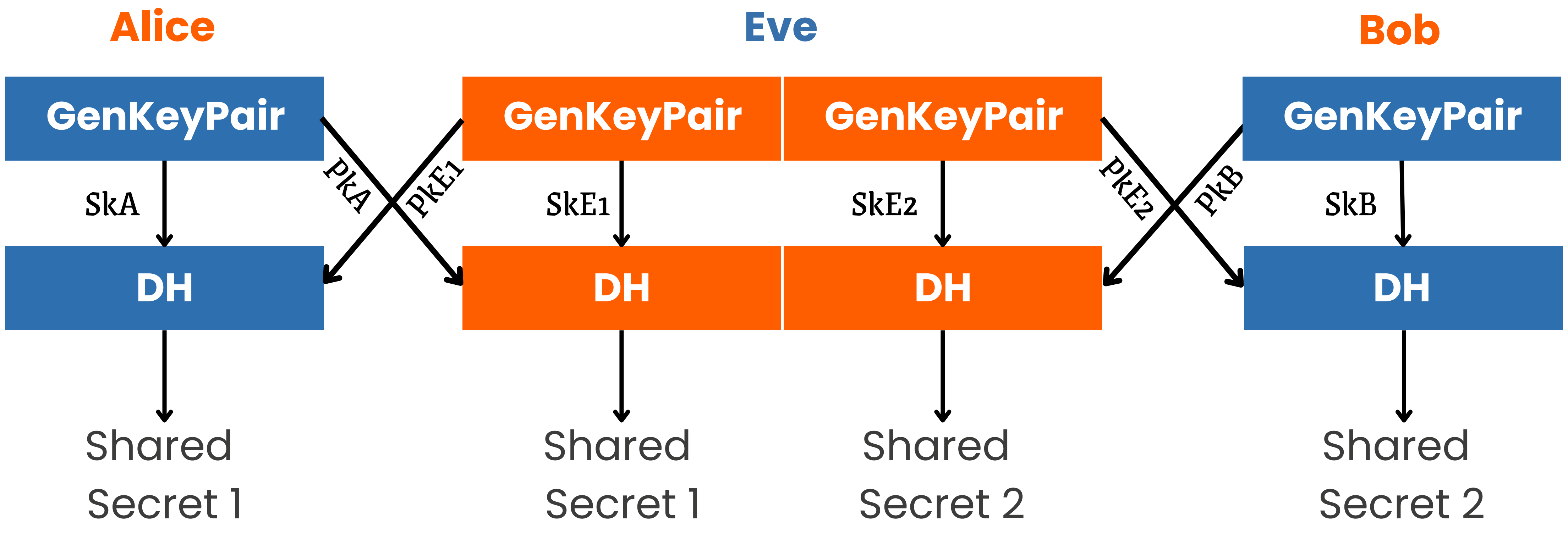
Diffie-Hellman

- *SecretKey, PublicKey = GenKeyPair()*
- *SharedSecret = DH(SelfSecretKey, PeerPublicKey)*



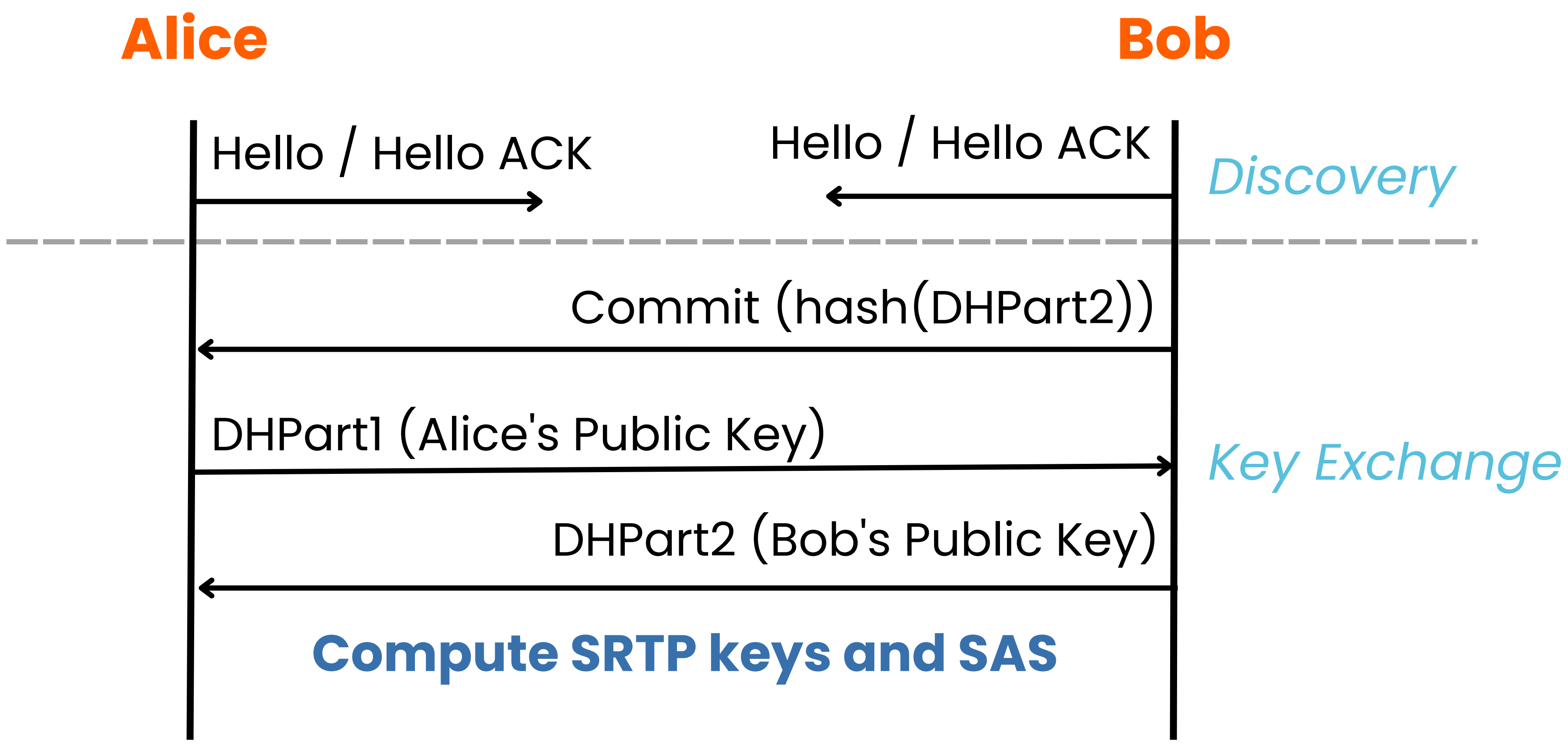
Alice
Bob
GenKeyPair
GenKeyPair
SkA
PkA
PkB
SkB
DH
DH
Shared Secret
Shared Secret

Diffie-Hellman is vulnerable to Man-in-the-Middle (MitM) Attack



Alice
Eve
Bob
GenKeyPair
GenKeyPair
GenKeyPair
GenKeyPair
SkA
PkA
PkE1
SkE1
SkE2
PkE2
PkB
SkB
DH
DH
DH
DH
Shared Secret 1
Shared Secret 1
Shared Secret 2
Shared Secret 2

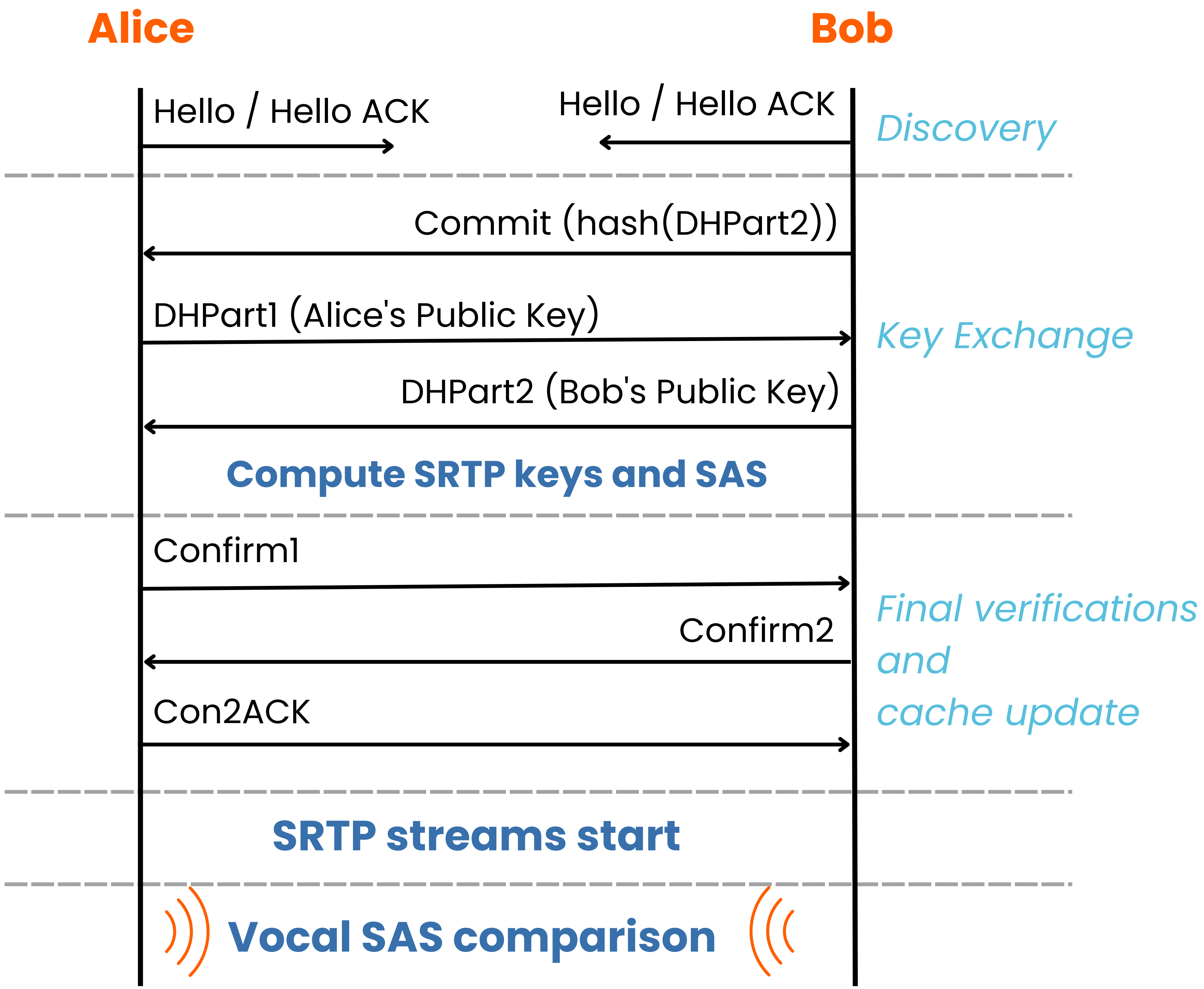
ZRTP handshake



Compute SRTP keys and SAS

- Derive s0 from DH result and transcript:
  - Hello, Commit and DHParts packets
- Derive SRTP keys from s0
- Derive SAS from s0 : 20 bits (4 characters)

ZRTP handshake



Alice

Bob

Hello / Hello ACK

Hello / Hello ACK

Discovery

Commit (hash(DHPart2))

DHPart1 (Alice's Public Key)

DHPart2 (Bob's Public Key)

Key Exchange

Compute SRTP keys and SAS

Confirm1

Confirm2

Con2ACK

Final verifications and cache update

SRTP streams start

Vocal SAS comparison

Compute SRTP keys and SAS

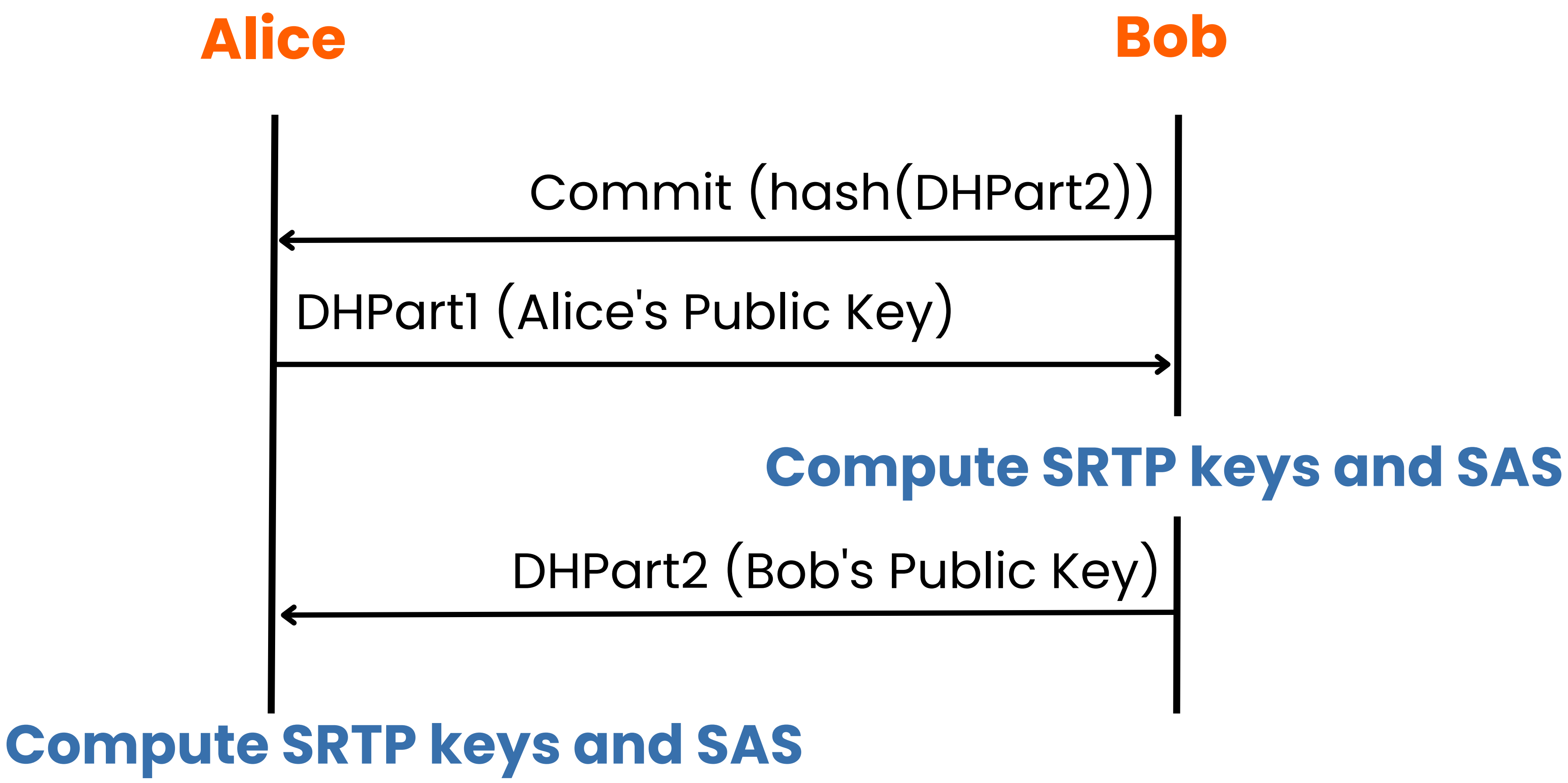
- Derive s0 from DH result and transcript:
  - Hello, Commit and DHParts packets
- Derive SRTP keys from s0
- Derive SAS from s0 : 20 bits (4 characters)

Vocal SAS comparison

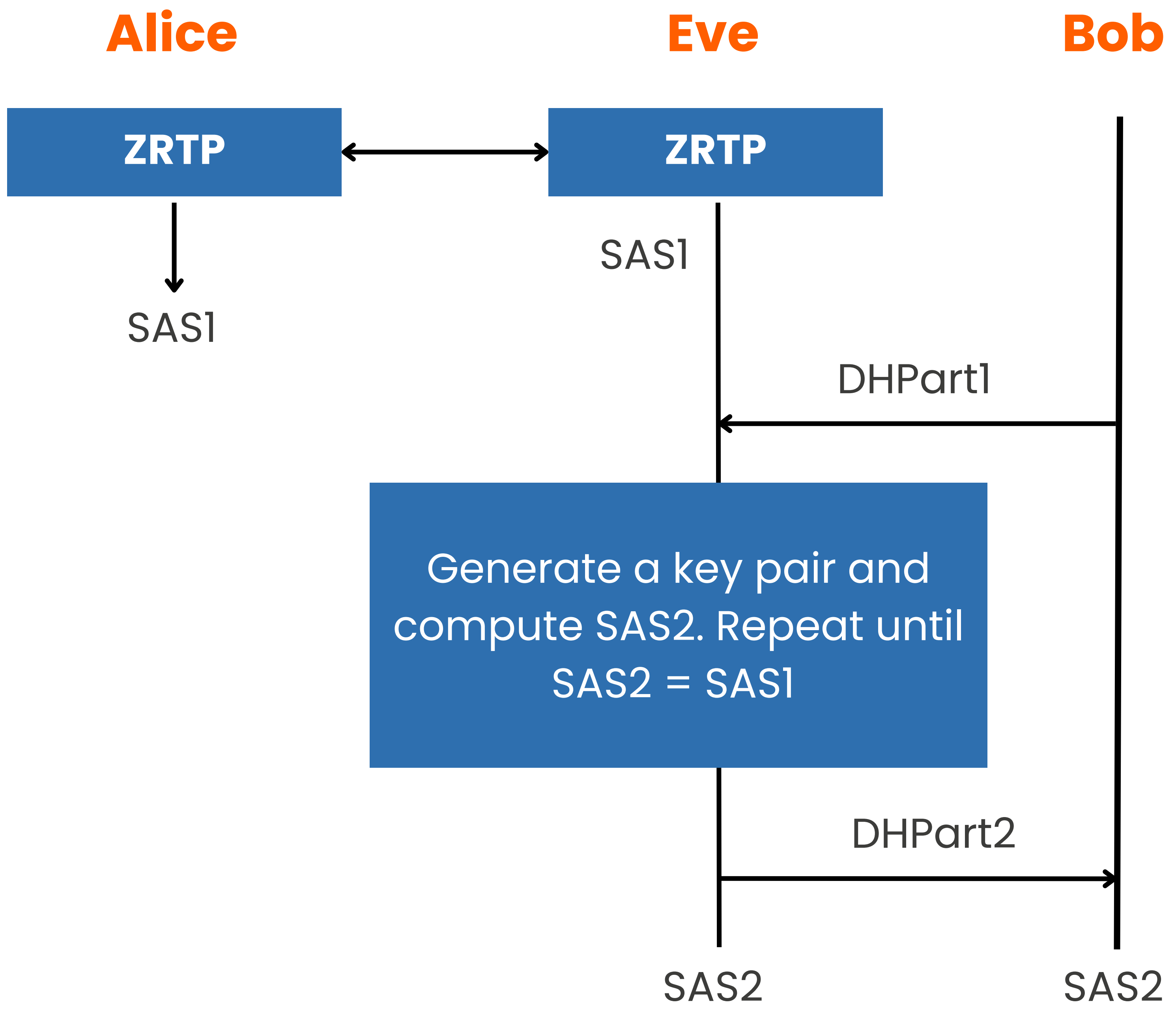
Detect MitM attack:

- Alice and Bob must assert they are using the same public keys
- Effective but not practical: each party reads its own public key to the other
- SAS value is short (4 characters) and based on both parties public keys

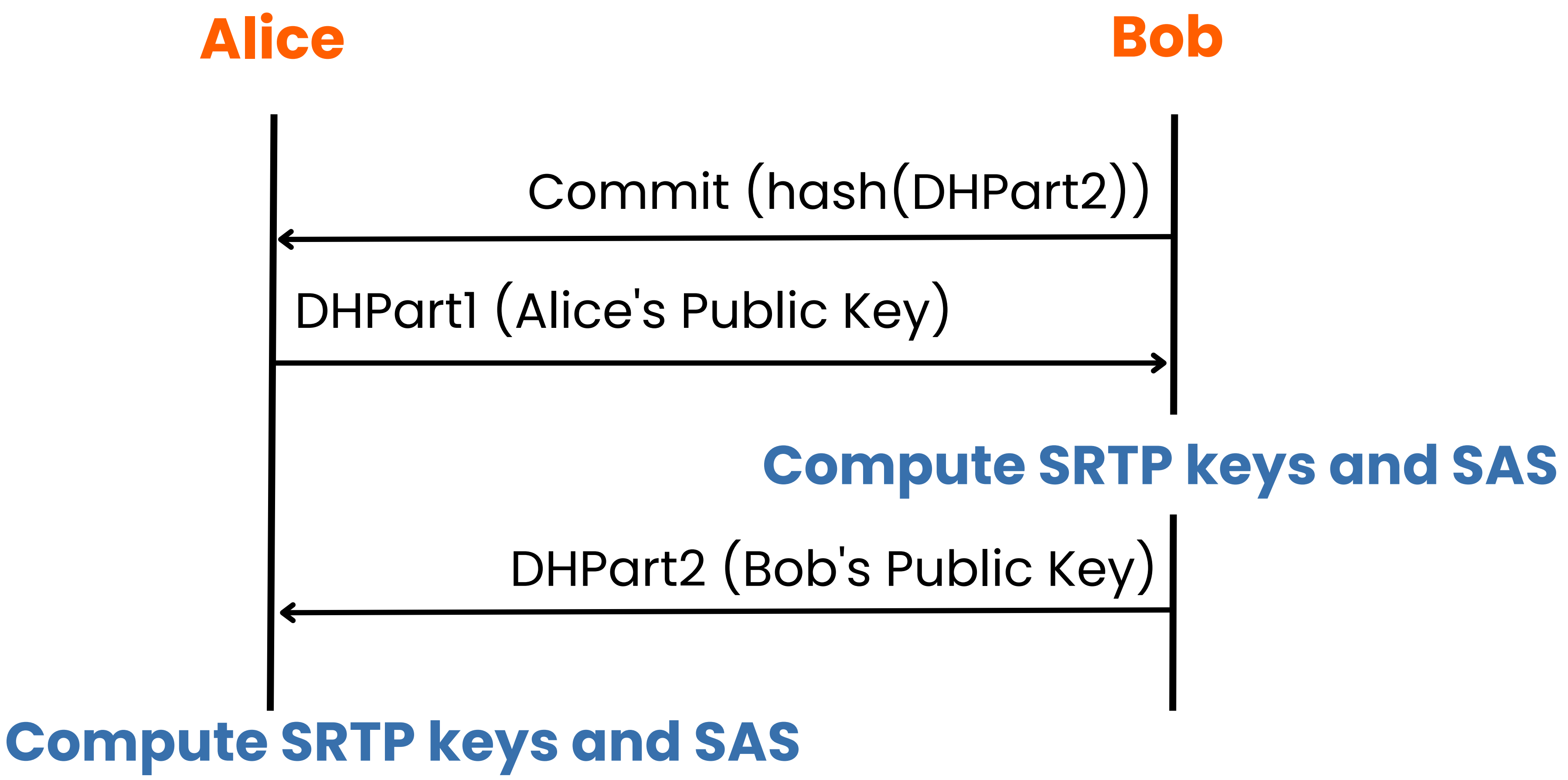
Commit Packet Role


Alice
Bob
Commit (hash(DHPart2))
DHPart1 (Alice's Public Key)
Compute SRTP keys and SAS
DHPart2 (Bob's Public Key)
Compute SRTP keys and SAS

SAS collision attack


Alice
Eve
Bob
ZRTP
ZRTP
SAS1
SAS1
DHPart1
Generate a key pair and compute SAS2. Repeat until SAS2 = SAS1
DHPart2
SAS2
SAS2

Commit Packet Role



Alice

Bob

Commit (hash(DHPart2))

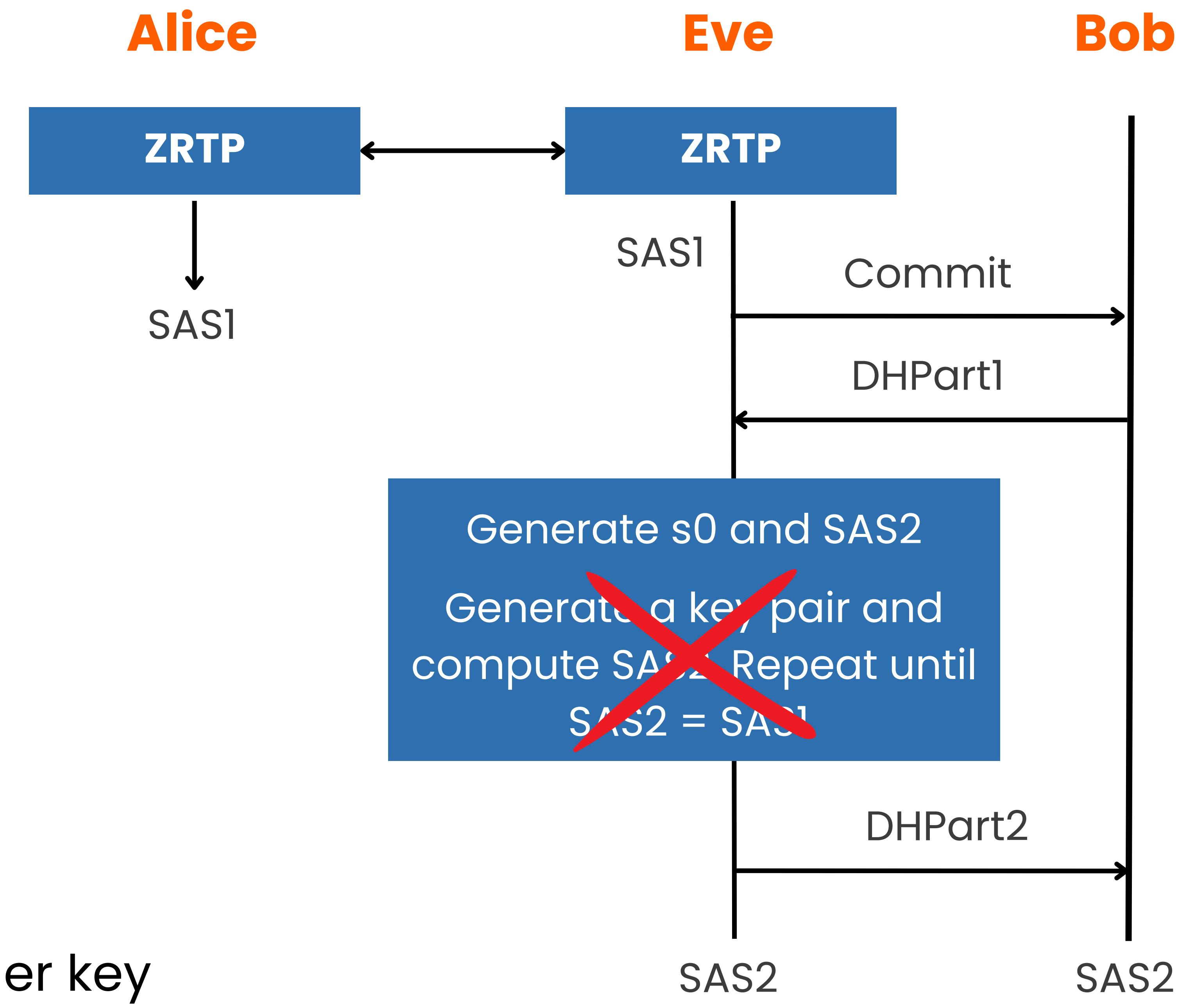
DHPart1 (Alice's Public Key)

Compute SRTP keys and SAS

DHPart2 (Bob's Public Key)

Compute SRTP keys and SAS

SAS collision attack



Alice

Eve

Bob

ZRTP

ZRTP

SAS1

SAS1

Commit

DHPart1

Generate s0 and SAS2

~~Generate a key pair and compute SAS2. Repeat until SAS2 = SAS1~~

DHPart2

SAS2

SAS2

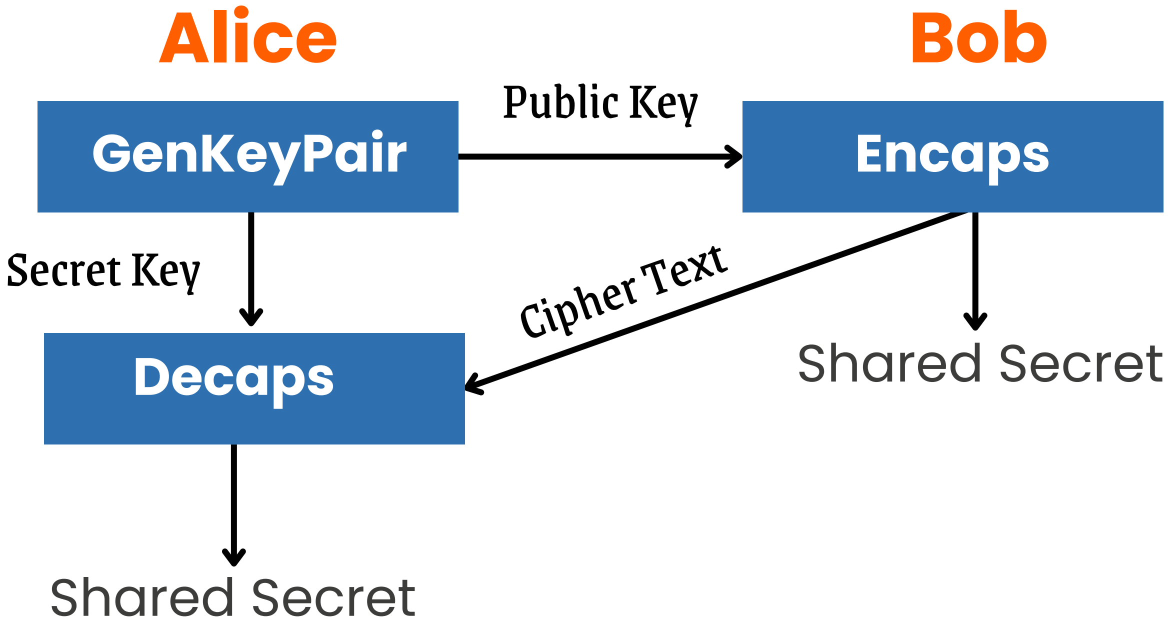
- Bob commits to use a public key without revealing it
- Alice cannot compute the shared secret before sending her key
- Bob cannot change his key to select a s0 and thus a SAS

III. POST QUANTUM KEY EXCHANGE (KEM)

Key Encapsulation Mechanism (KEM) interface

NIST requests Post Quantum key exchange algorithms to use KEM interface :

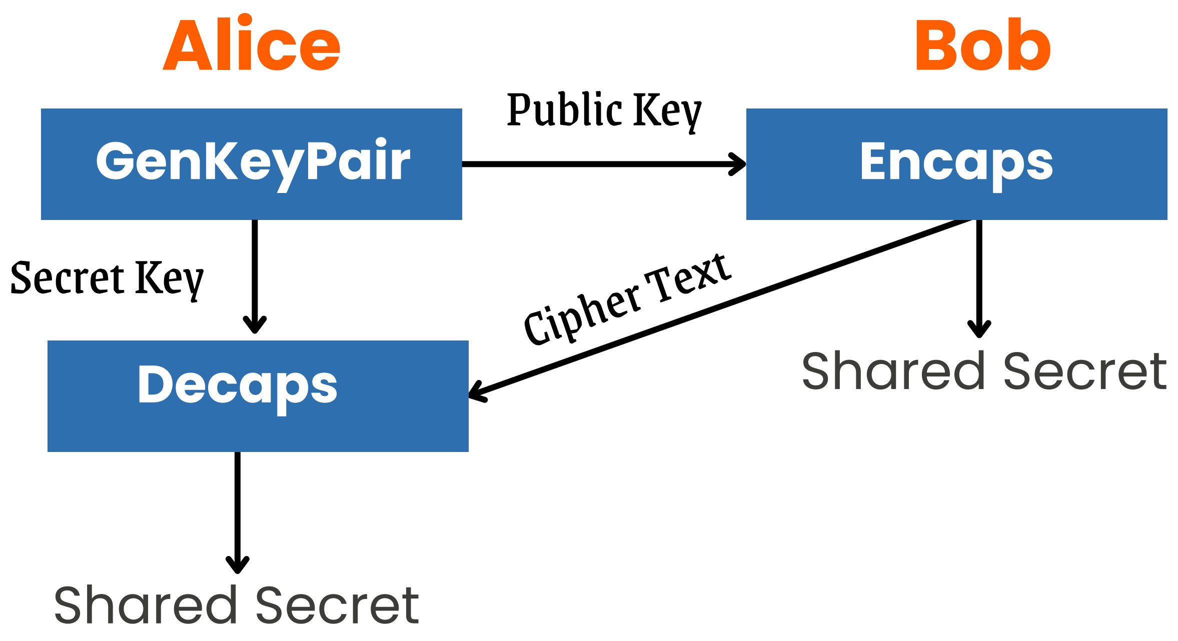
- SecretKey, PublicKey = GenKeyPair()
- SharedSecret, CipherText = encaps(PublicKey)
- SharedSecret = decaps(CipherText, SecretKey)



Key Encapsulation Mechanism (KEM) interface

NIST requests Post Quantum key exchange algorithms to use KEM interface :

- SecretKey, PublicKey = GenKeyPair()
- SharedSecret, CipherText = encaps(PublicKey)
- SharedSecret = decaps(CipherText, SecretKey)



KEM is not Diffie-Hellman :

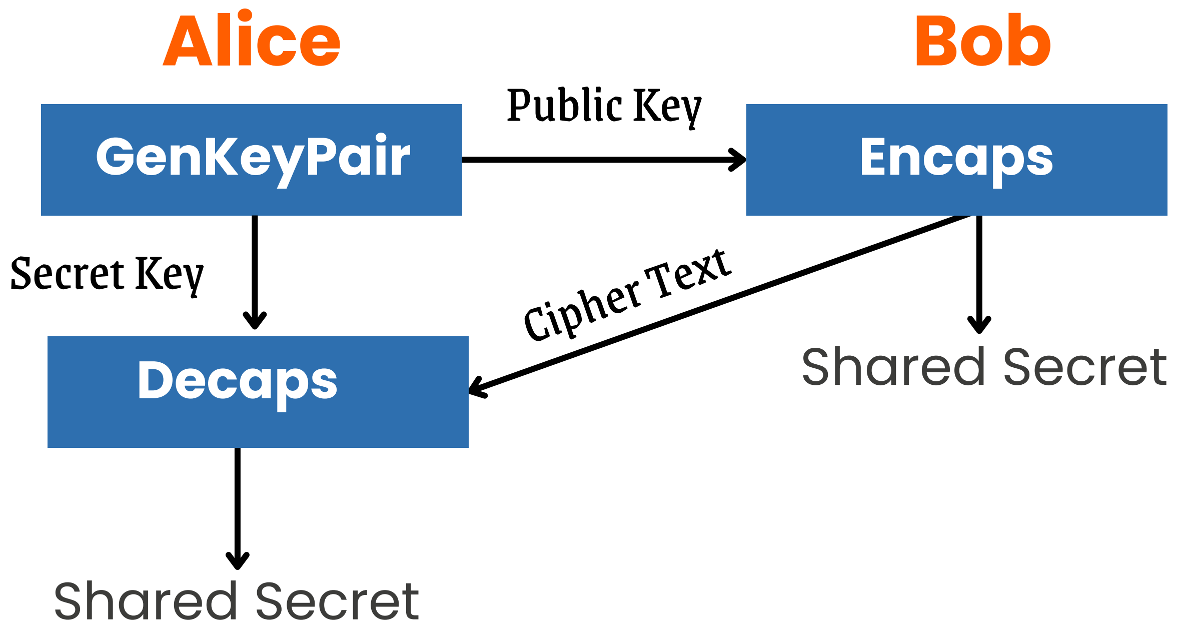
- Not symmetric
- Shared Secret can be selected by one party

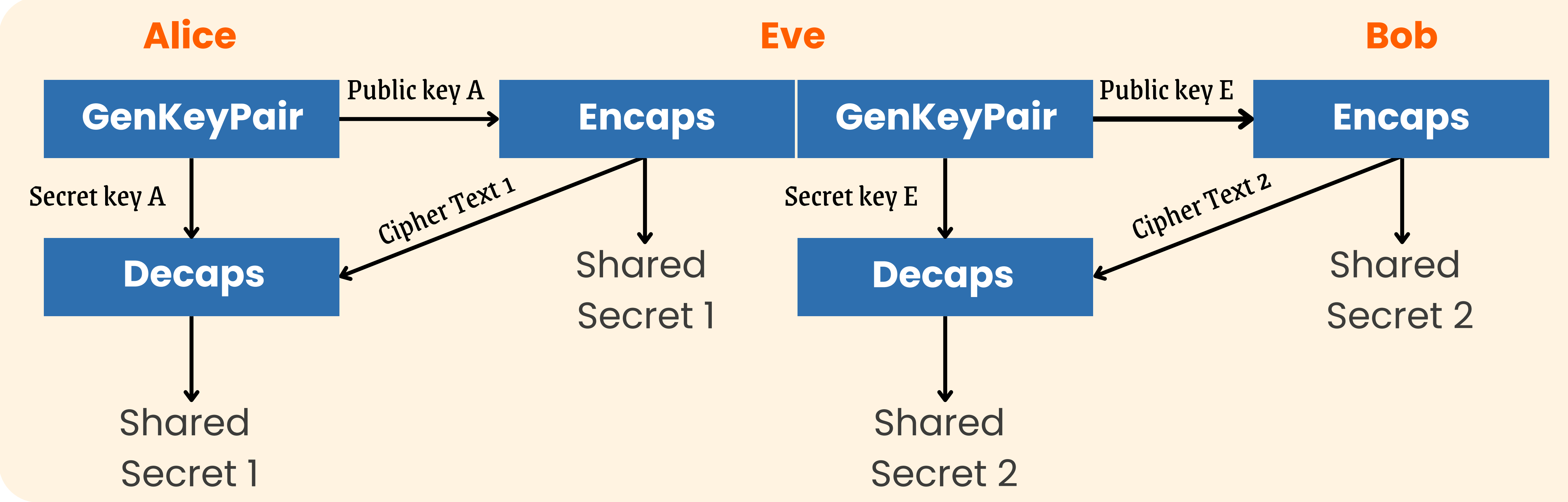
Key Encapsulation Mechanism (KEM) interface

NIST requests Post Quantum key exchange algorithms to use KEM interface :

- SecretKey, PublicKey = GenKeyPair()
- SharedSecret, CipherText = encaps(PublicKey)
- SharedSecret = decaps(CipherText, SecretKey)

KEM too is vulnerable to Man-in-the-Middle (MitM) Attack





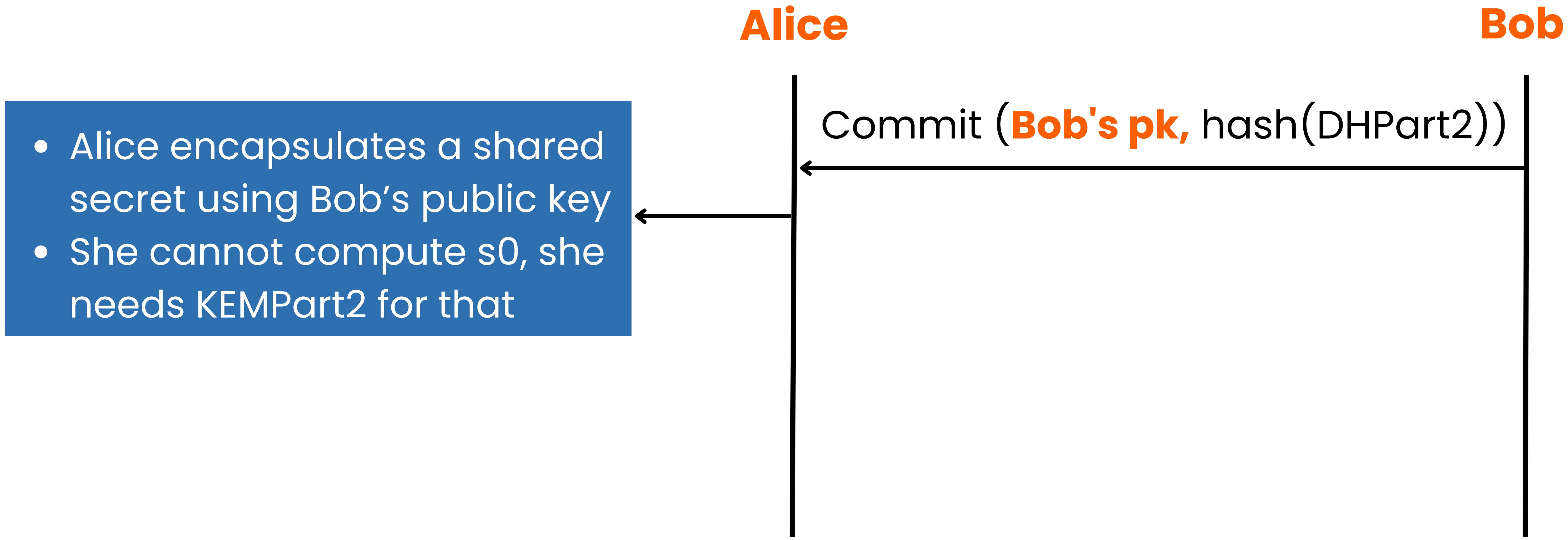
KEM is not Diffie-Hellman :

- Not symmetric
- Shared Secret can be selected by one party

IV. ZRTP ADAPTATION

ZRTP KEM handshake

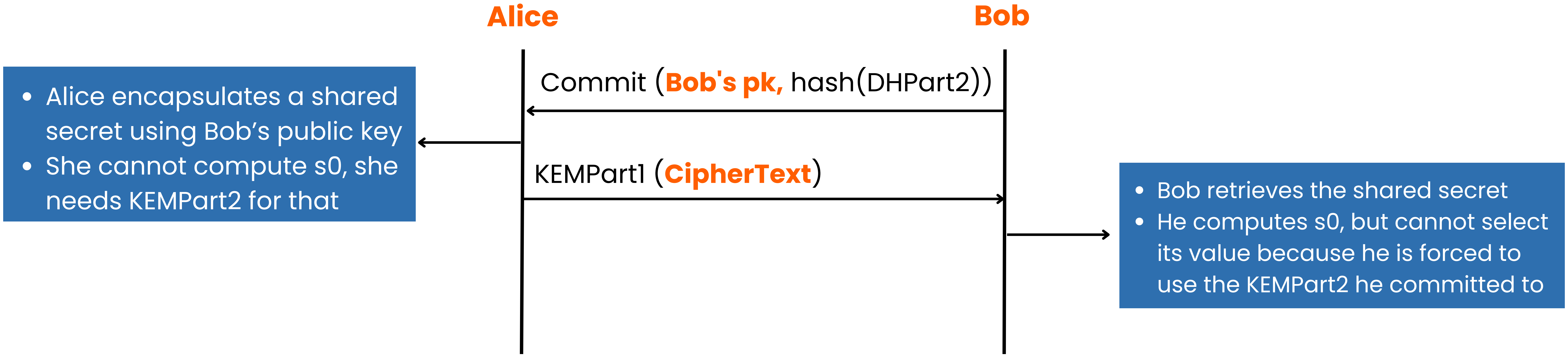
- s0 is derived from KEM shared secret and a transcript including:
  - Commit, KEMPart1 and KEMPart2 packets
- SRTP keys and SAS are derived from s0



- Alice encapsulates a shared secret using Bob’s public key
- She cannot compute s0, she needs KEMPart2 for that

ZRTP KEM handshake

- s0 is derived from KEM shared secret and a transcript including:
    - Commit, KEMPart1 and KEMPart2 packets
- SRTP keys and SAS are derived from s0



Alice

Bob

Commit (**Bob's pk,** hash(DHPart2))

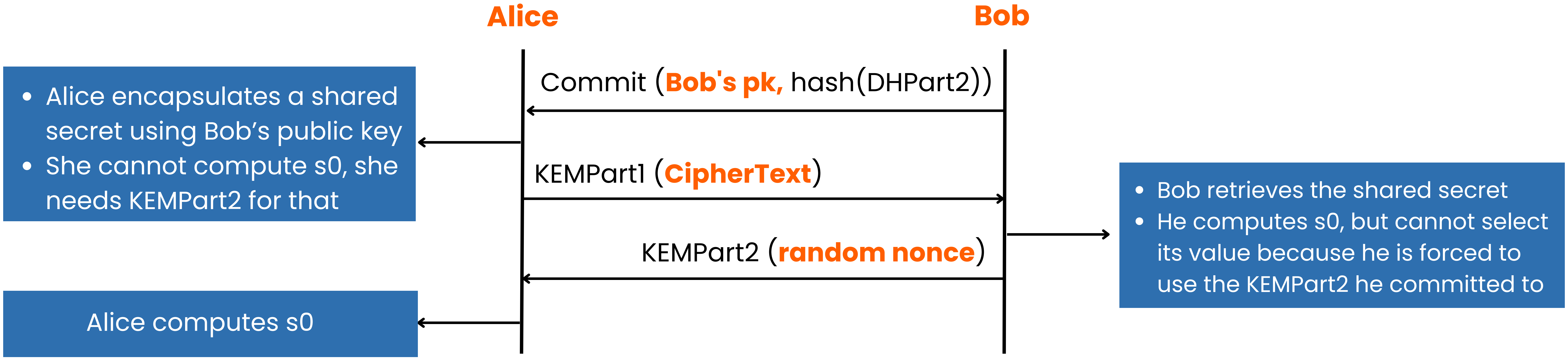
KEMPart1 (**CipherText**)

- Alice encapsulates a shared secret using Bob’s public key
- She cannot compute s0, she needs KEMPart2 for that

- Bob retrieves the shared secret
- He computes s0, but cannot select its value because he is forced to use the KEMPart2 he committed to

ZRTP KEM handshake

- s0 is derived from KEM shared secret and a transcript including:
  - Commit, KEMPart1 and KEMPart2 packets
- SRTP keys and SAS are derived from s0



Alice

Bob

Commit (**Bob's pk,** hash(DHPart2))

KEMPart1 (**CipherText**)

KEMPart2 (**random nonce**)

- Alice encapsulates a shared secret using Bob's public key
- She cannot compute s0, she needs KEMPart2 for that

Alice computes s0

- Bob retrieves the shared secret
- He computes s0, but cannot select its value because he is forced to use the KEMPart2 he committed to

V. HYBRID KEM

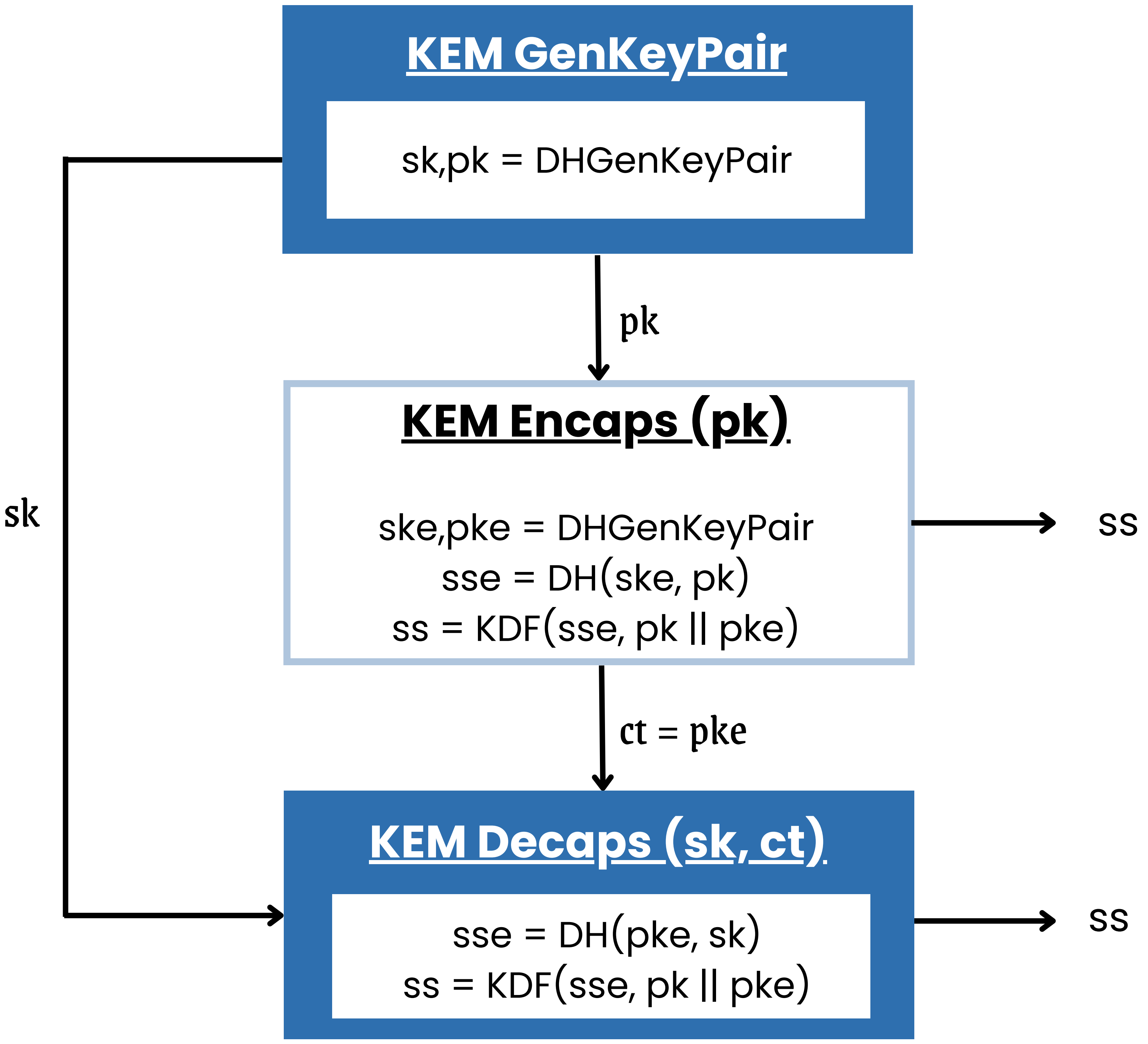
Hybrid key exchange : combine Post Quantum and traditional algorithm

- Post Quantum cryptography is relatively new and weakness may still be found (SIKE is a good example of the unexpected happening…)
- Perform one (or several) Post Quantum key exchange to protect against quantum computer
- and keep using (EC)DH key exchange so current security level is not downgraded

For ZRTP protocol simplicity

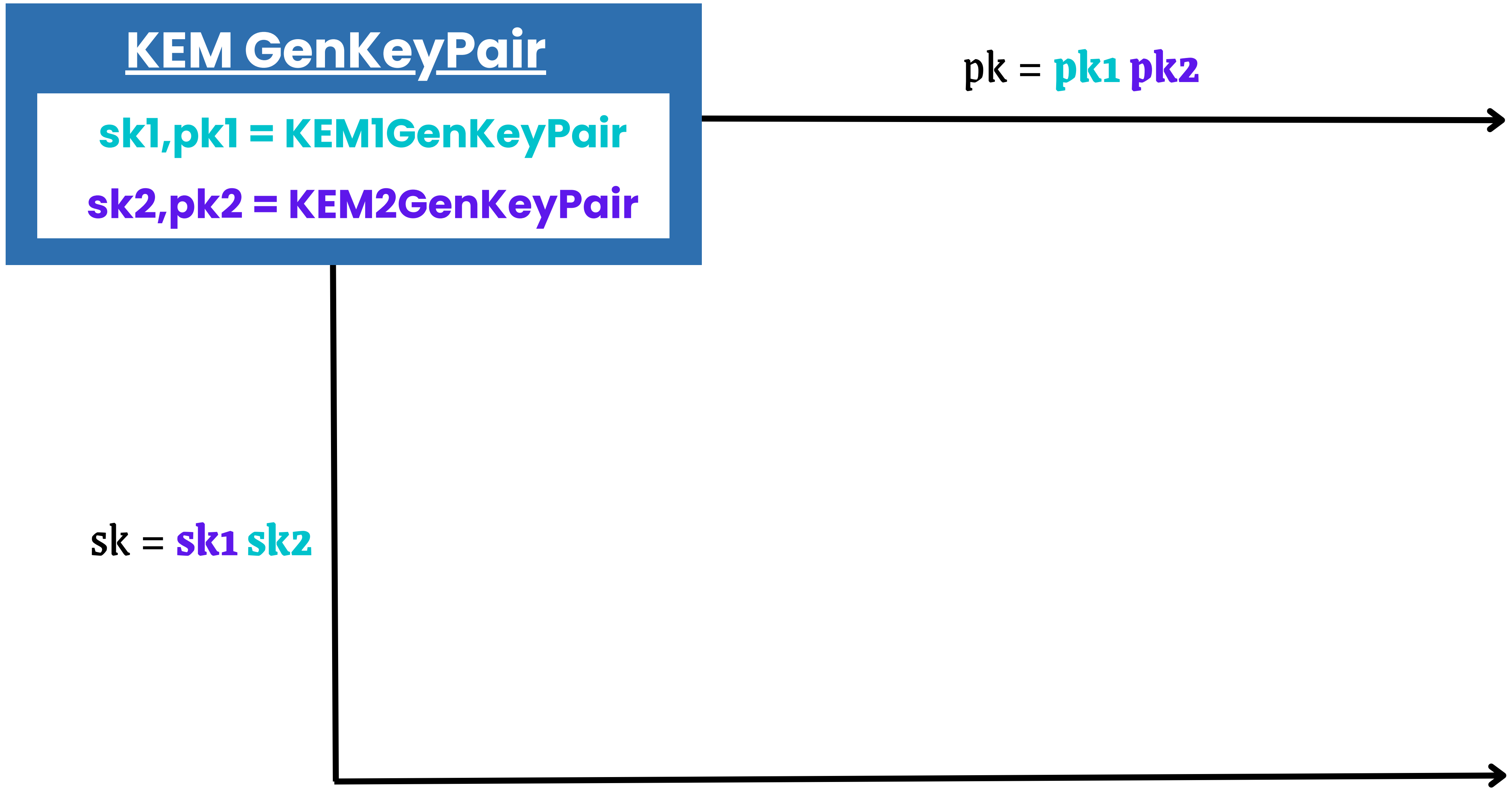
- The (EC)DH key exchange is performed as a KEM
- (EC)DH based and PQC KEM are combined into one KEM

Build a KEM from Diffie-Hellman



Combine two(or more) KEM

N. Bindel and al. - Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange



KEM GenKeyPair

sk1,pk1 = KEM1GenKeyPair

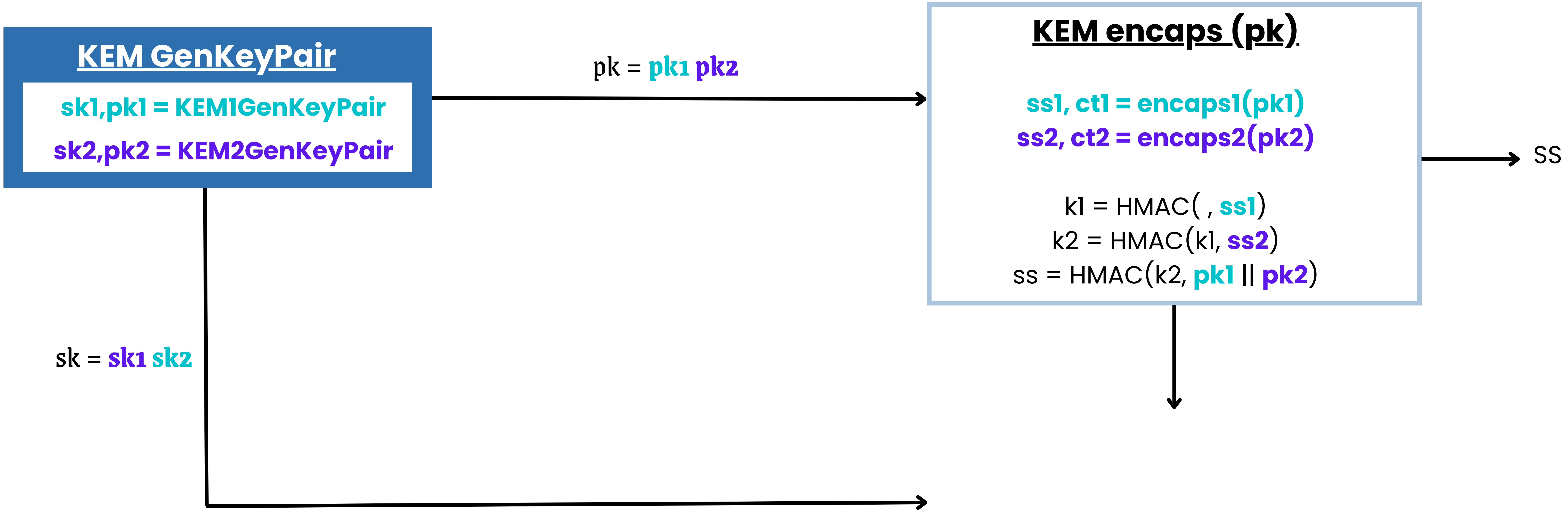
sk2,pk2 = KEM2GenKeyPair

pk = pk1 pk2

sk = sk1 sk2

Combine two(or more) KEM

N. Bindel and al. - Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange



KEM GenKeyPair

sk1,pk1 = KEM1GenKeyPair

sk2,pk2 = KEM2GenKeyPair

pk = pk1 pk2

sk = sk1 sk2

KEM encaps (pk)

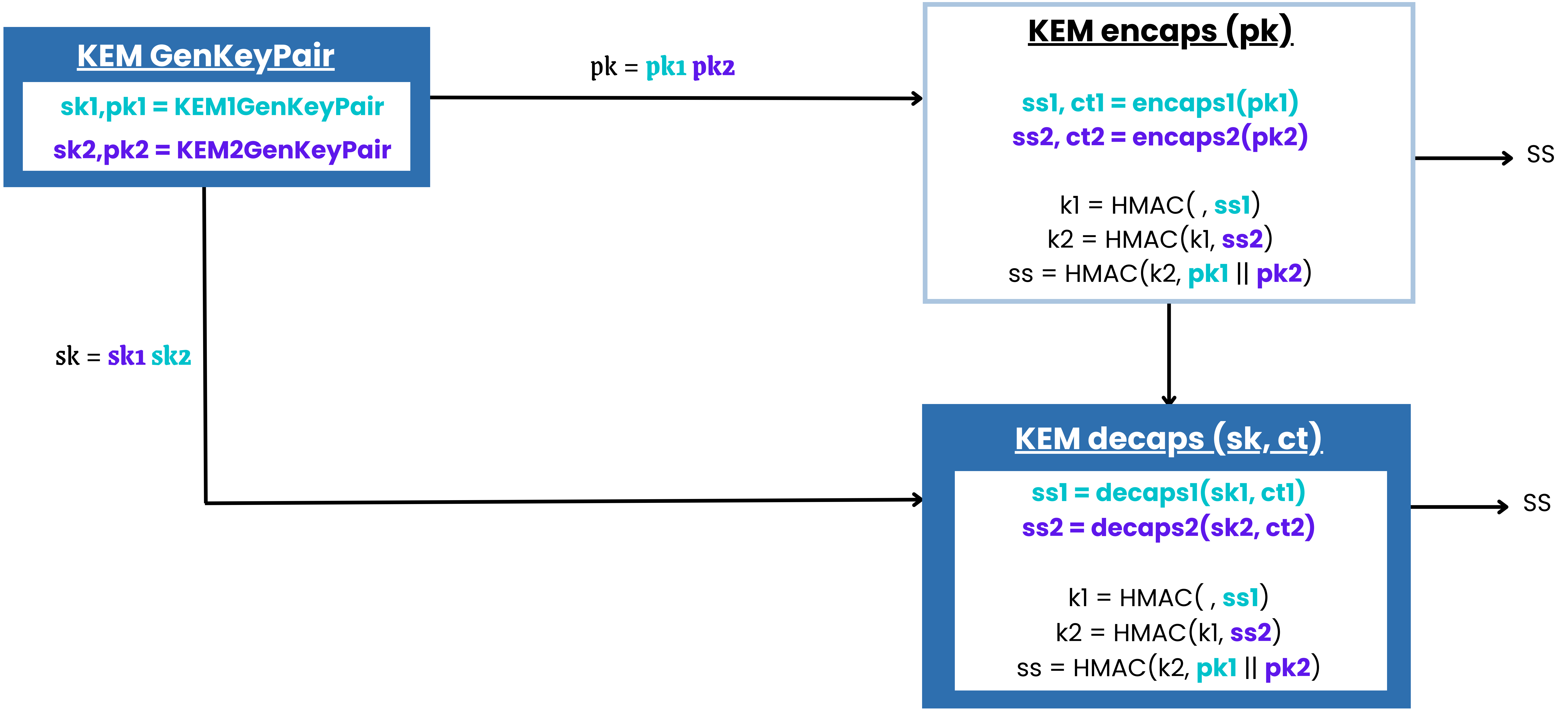
ss1, ct1 = encaps1(pk1)
ss2, ct2 = encaps2(pk2)

k1 = HMAC(, ss1)
k2 = HMAC(k1, ss2)
ss = HMAC(k2, pk1 || pk2)

ss

Combine two(or more) KEM

N. Bindel and al. - Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange



KEM GenKeyPair

sk1,pk1 = KEM1GenKeyPair

sk2,pk2 = KEM2GenKeyPair

pk = pk1 pk2

sk = sk1 sk2

KEM encaps (pk)

ss1, ct1 = encaps1(pk1)
ss2, ct2 = encaps2(pk2)

k1 = HMAC(, ss1)
k2 = HMAC(k1, ss2)
ss = HMAC(k2, pk1 || pk2)

ss

KEM decaps (sk, ct)

ss1 = decaps1(sk1, ct1)
ss2 = decaps2(sk2, ct2)

k1 = HMAC(, ss1)
k2 = HMAC(k1, ss2)
ss = HMAC(k2, pk1 || pk2)

ss

VI. FOCUS AND CONCLUSION

ZRTP handshake is performed in the media stream on UDP

- Typical MTU: 1500 bytes
- Diffie-Hellman public key size: from 32 (X25519) to 384 (DH3072) bytes
- Kyber public key or cipher text : up to 1568 bytes
- HQC cipher text: up to 14 kB

Fragment the ZRTP Commit and KEMPart1 packet

- ZRTP packet header modified to support fragmentation
- Fragmentation is opportunistic to keep interoperability with older version

Crypto libraries

- Liboqs : Kyber, HQC. More PQ KEM available.
- Libdecaf and mbedtls: X25519, X448, HMAC functions

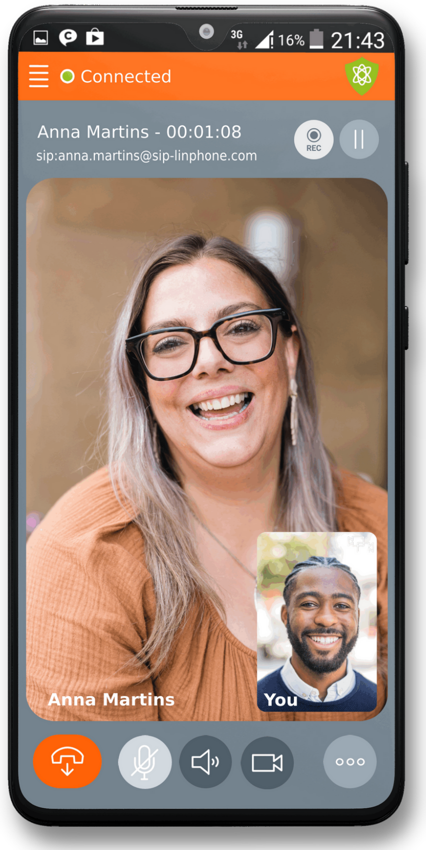
Hybrid KEM

- Provides : ECDH-based KEM for X25519 and X448
- Can combine with Kyber (512, 768 and 1024), HQC(128, 192, 256)
- Can combine more than 2 KEM
- In a dedicated module (under GPLv3) so any project can use it

ZRTP implementation

- Deployed with a preset of hybrid KEM available:
    - X25519/Kyber512, X25519/HQC128, X25519/Kyber512/HQC128
    - X448/Kyber1024, X448/HQC256, X448/Kyber1024/HQC256
- Is fully compatible with older version

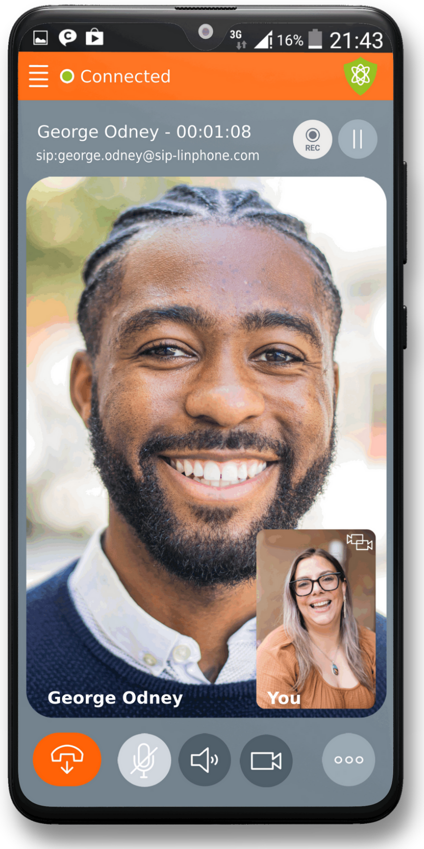
Audio Call

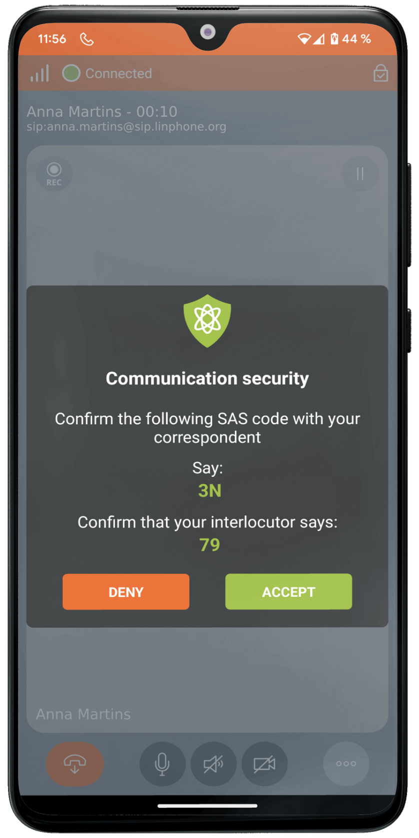


ZRTP handshake

↔

Compute SRTP key and SAS





Communication security

Confirm the following SAS code with your correspondent

Say:
3N

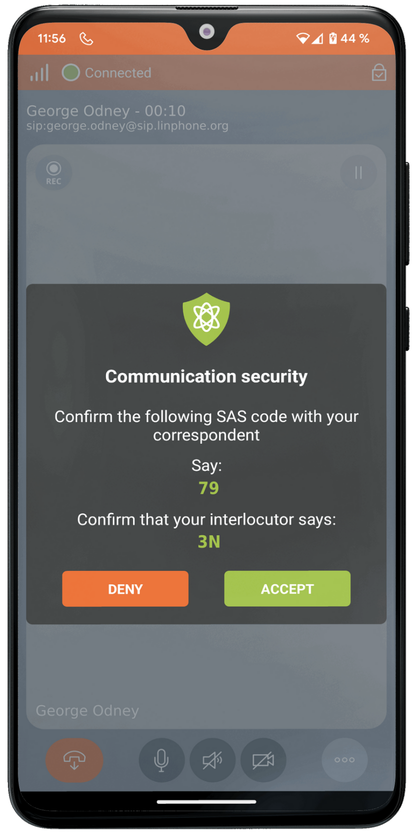
Confirm that your interlocutor says:
79

DENY ACCEPT

Compare SAS

↔

Once in call history



Communication security

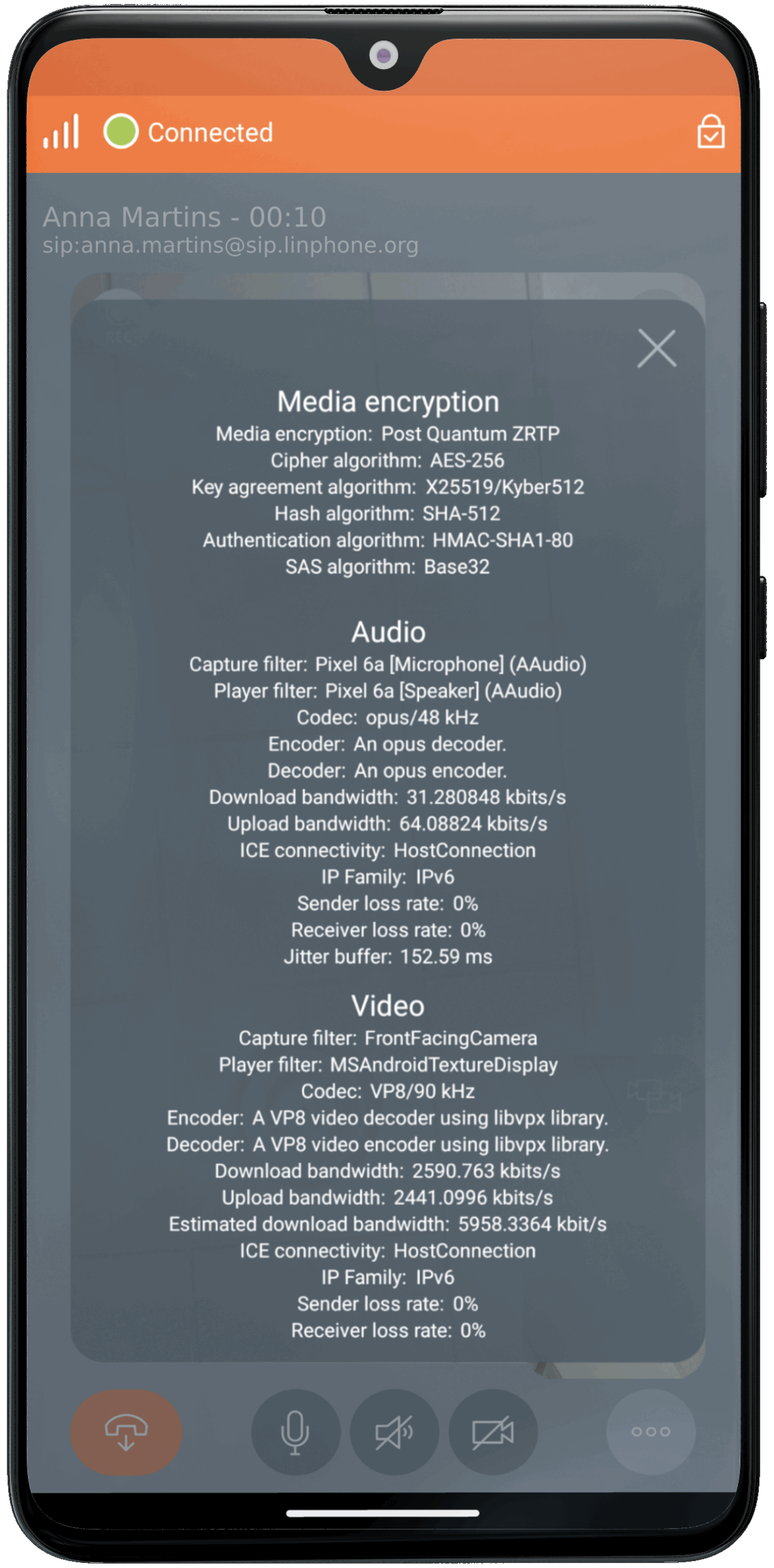
Confirm the following SAS code with your correspondent

Say:
79

Confirm that your interlocutor says:
3N

DENY ACCEPT

Check what type of security :



Media encryption

Media encryption: Post Quantum ZRTP
Cipher algorithm: AES-256
Key agreement algorithm: X25519/Kyber512
Hash algorithm: SHA-512
Authentication algorithm: HMAC-SHA1-80
SAS algorithm: Base32

Audio

Capture filter: Pixel 6a [Microphone] (AAudio)
Player filter: Pixel 6a [Speaker] (AAudio)
Codec: opus/48 kHz
Encoder: An opus decoder.
Decoder: An opus encoder.
Download bandwidth: 31.280848 kbits/s
Upload bandwidth: 64.08824 kbits/s
ICE connectivity: HostConnection
IP Family: IPv6
Sender loss rate: 0%
Receiver loss rate: 0%
Jitter buffer: 152.59 ms

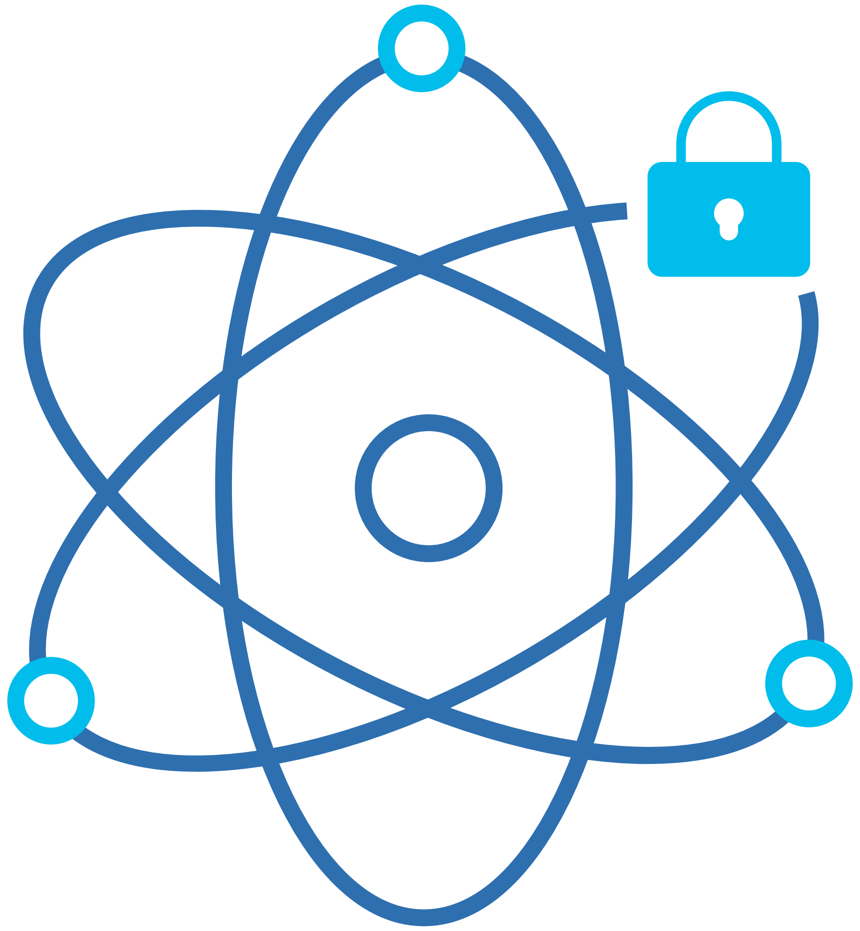
Video

Capture filter: FrontFacingCamera
Player filter: MSAndroidTextureDisplay
Codec: VP8/90 kHz
Encoder: A VP8 video decoder using libvpx library.
Decoder: A VP8 video encoder using libvpx library.
Download bandwidth: 2590.763 kbits/s
Upload bandwidth: 2441.0996 kbits/s
Estimated download bandwidth: 5958.3364 kbit/s
ICE connectivity: HostConnection
IP Family: IPv6
Sender loss rate: 0%
Receiver loss rate: 0%

Useful links

- **Linphone website**: https://www.linphone.org
- **Post Quantum Cryptography in Linphone**: https://www.linphone.org/sites/default/files/pqcrypto_integration-3_0.pdf
- **PostQuantumCryptoEngine module**: https://gitlab.linphone.org/BC/public/postquantumcryptoengine
- **ZRTP implementation**: https://gitlab.linphone.org/BC/public/bzrtp
- **N. Bindel and al. - Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange**:

  https://eprint.iacr.org/2018/903.pdf



Thank you!

linphone

open source VOIP project

Do you have any questions ?

linphone

open source VOIP project