Performing a mutually authenticated key exchange with perfect forward secrecy using a KEM

Ronald Landheer-Cieslak, *Member, IEEE*

***Abstract*—The advent of quantum computing lends urgency to the development of secure protocols that can be built entirely with post-quantum algorithms. This excludes the use of the family of algorithms Diffie-Hellman and Elliptic Curve Diffie Hellman belong to, as there are no algorithms in that family that are likely to be selected in the NIST program for post-quantum cryptography, and previous candidates have been eliminated due to security issues.**

This paper presents a new protocol for a secure mutually authenticated key exchange with perfect forward secrecy, based on the use of KEM algorithms.

***Index Terms*—KEM, Perfect Forward Secrecy, AKE, UAKE, MAKE**

I. Introduction

WITH the advent of quantum computing [1] [2] [3] [4] and the now-pressing need for post-quantum cryptography [5], key encapsulation mechanisms like CRYSTALS – Kyber [6] are set to replace [7] the currently more common key exchange mechanisms like the Diffie-Hellman key exchange (DH) [8] and Elliptic Curve Diffie-Hellman key exchange (ECDH) [9]: NIST has put three digital signature algorithms (DSAs) and one key encapsulation mechanism, (KEM) on a track to standardization, among which Crystals-KYBER, and is in its fourth round of looking for post-quantum cryptography algorithms [7]. They are unlikely to choose additional algorithms for standardization from the Lattice-based Learning-With-Errors family of algorithms that CRYSTALS-Kyber is a member of: the four KEM algorithms that have advanced to the fourth round are [10] BIKE [11], Classic McEliece [12], HQC [13], and SIKE [14]. While SIKE is a DH-type algorithm, it is now known to be broken [15]. None of the remaining algorithms are DH-type algorithms and none of these algorithms are based on the same principles as the chosen KEM. Choosing algorithms based in a diversity of mathematical principles is in line with an aim for cryptographic agility: the idea that if the underlying assumptions of one algorithm's security turn out to be wrong, other algorithms can still be used to make up for that weakness if they rely on different assumptions.

This essentially means there will not be a post-quantum DH-type algorithm in the foreseeable future, which in turn means we need some other means to exchange symmetric keys. These means will need to meet a number of requirements to be generally applicable without too many constraints on their use.

Ronald Landheer-Cieslak is a member of various IEEE working groups and DNP-UG committees pertaining to the securing and automation of power grids, chairs the DNP CSTF, and is founder and sole proprietor for Vlinder Software.

Manuscript started on September 19, 2023, last revised June 8, 2024

A. Requirements for a new key exchange protocol

Obviously, because DH is the only key exchange mechanism that allows for mutually authenticated *off-line* key exchange, the requirements cannot include the "off-line" part. We can, however, require mutual authentication: the ability for Alice to know a message she sends to Bob can only be read by Bob, and for Bob to know a message he receives from Alice was sent by Alice.

We can also require confidentiality. That is: if Alice sets up a shared key with Bob, an attacker, Eve, shall not be able to deduce the key from the exchanged messages, even if she is allowed to interact with either Alice or Bob. There are a few ways to refine this, but Rackoff *et al.* in [16] have a concice description of Eve's challenge, known as a *Chosen Ciphertext Attack (CCA)*: upon observing a one-bit message from Alice to Bob, she is permitted to generate any feasible number of messages, associated with any assigned public key she chooses, and receive their correct decryptions according to the receiver, Bob. If she can, with any significant probability greater than $\frac{1}{2}$, decrypt the intercepted one-bit message after this process, she has succeeded the attack[1].

We can also require perfect forward secrecy. Forward secrecy as defined in [17] means that confidentiality of messages using the key agreed upon in previous key exchanges remains secure, even if one party's private key is compromised. That is: if Alice and Bob perform a key exchange using some protocol P, using their private keys sk_A and sk_B, resp., to arrive at a shared secret K, and Eve obtains either Alice's private key sk_A or Bob's private key sk_B, forward secrecy means she still can't obtain the shared key k.

Perfect forward secrecy adds to this, by also preventing Eve from obtaining future shared secrets exchanged using those same keys. Note, however, that if Eve obtains sk_B, even with perfect forward secrecy, she could impersonate Bob if sk_B is used to authenticate him: compromised keys are still compromised.

Finally, we want crypto-agility: any protocol we design that uses a KEM to exchange keys with confidentiality, mutual authentication, and perfect forward secrecy should not be tied to a single KEM algorithm or an algorithm from a particular family of algorithms. However, we also don't want this new protocol to rely on a plethora of different algorithms: the protocol shall only use a single asymmetric cryptography algorithm and may optionally use a single secure hashing algorithm.

[1]This description of a Chosen Ciphertext Attack, CCA, is reproduced almost verbatim from [16]. The only difference here is that Eve has a name and the fact that the message is one bit is repeated.

□

either accept the input and produce a different $k' \neq k$ or $\perp$ to indicate an error.

As in the general case, a KEM will succeed only if the public key and private key used are from the same pair, KEMs may be used for key exchange protocols. Three such protocols are described in [6]: Kyber.KE, Kyber.UAKE, and Kyber.AKE. None of these schemes actually depend on the Kyber KEM specifically.

II. Building blocks for a new key exchange protocol

A. Key Encapsulation Mechanism

As described in [6], a KEM is part of a family of public key encryption schemes that rely on three functions (there called KeyGen, Encaps, and Decaps, here called G, E and D). The key generation function G generates a pair consisting of a public and private key, $\langle sk\ pk \rangle$. The encapsulation function E takes the public key pk and generates a random secret k from a key space $k \in \kappa$ and the corresponding ciphertext ct. The decapsulation function D takes the ciphertext ct and the private key sk and reproduces the secret k.

The typical use-case for a KEM is for Alice to generate a shared secret she can use to encrypt a message only Bob can read: Alice uses Bob's public key pk_B with the function E to generate the shared secret k and the ciphertext ct. She then sends the ciphertext to Bob (over an open channel: the ciphertext is not secret) so Bob can use his own secret key sk_B to obtain the same shared secret k. This use-case, which is the simplest case for a KEM-based key exchange, allows Alice to be sure that only Bob can read the message, but does not allow Bob to ensure that the message came from Alice. In order to do that, Alice either needs to sign the message she sent using some other algorithm, or Alice and Bob need to use a more intricate key exchange protocol to arrange for *mutual authentication*[2]. This use-case also does not allow for forward secrecy, because Eve can obtain the same shared secret k if she obtains sk_B. Provided she has been able to capture ct (which, as noted, is not a secret and should therefore be assumed to have been captured), once she also has sk_B she can obtain the shared secret k and decrypt anything that may have been encrypted with that secret.

There are a few plausible error-cases for KEM:

1) The public key used by Alice is somehow corrupted or has been tampered with. In this case, if the public key pk' is invalid the D function will reject it with $\perp$. If it is a valid public key, E will generate valid-seeming outputs, but the exchange with Bob will fail later[3].
2) The public key Alice used to generate the ciphertext does not correspond to Bob's secret key. In this case, the D function returns $\perp$ as a rejection. It may, in some cases, produce a false-positive value indistinguishable, from the user's point of view, from success, but will not produce the same value as k. In such cases, subsequent message exchanges will result in errors if k is used as the shared secret by Alice and k' is used as the shared secret by Bob.
3) The ciphertext ct is tampered with. This case is indistinguishable from the previous case: the D function will

[2]Mutual authentication is one of the more desirable properties of key exchange protocols: for a message sent by Alice to Bob, it allows Alice to ensure only Bob can read the message, and it allows Bob to ensure the message was really sent by Alice.

[3]Note we don't go into the possibility of an attack in this context: using a different public key than Bob's is an attack vector when using only KEM in a one-way exchange, because mutual authentication is not available.

B. Secure Hashing Algorithm

As described in [18], a SHA is a function that generates a *message digest* from an input message:

> "When a message of any length $< 2^{64}$ bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then be input to the Digital Signature Algorithm (DSA) which generates or verifies the signature for the message (...). Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature."

Since the publication of [18], uses for SHA have evolved beyond the creation of message digest to their incorporation in key derivation functions [19]. and SHAs themselves have evolved beyond SHA-1 (which has now been compromised [20] and is therefore no longer recommended for use) to SHA-2 and SHA-3 [21].

In this paper, we define a function H which is used to combine shared secrets into a single shared secret. To accomplish this, H is defined as an HMAC-based extract-and-expand key derivation function (HKDF) [19], using an appropriate SHA defined in [21] as the underlying SHA. For the purposes of this paper, the *info* parameter for the HKDF is a public fingerprint determined by Alice, defined below. The *salt* parameter is defined as the concatenated values of Alice's *nonce*, n_A, and Bob's *nonce*, n_B, which are defined below. The *IKM* parameter is further defined as the concatenation of the secrets produced by the KEM function defined above. This means that the notation below $h \leftarrow H\ n_A|n_B\ i\ s_1|s_2|s_3|s_4$, where $|$ is the concatenation operator, means the HKDF function is called with the concatenation of the two values n_A and n_B as the first (*salt*) parameter, the value i as the second (*info*) parameter, and the concatenation of the four values s_1, s_2, s_3, and s_4 as the third (*IKM*) parameter. The function H as defined here encapsulates both the *HKDF-Extract* and *HKDF-Expand* steps defined in [19].

When the underlying SHA function itself is used in the protocol described below, it is indicated as SHA.

C. Random Number Generator

While the KEM function E generates random shared secrets, and the protocol proposed in this paper uses four of them, the HKDF function used to define H takes a *nonce*

parameter, which ought to be random, as well. To generate this parameter, both Alice and Bob generate a random value using a random number generator function R. [22] provides recommendations on how to costruct such a random number generator (or random *bit* generator) for cryptographic calculations. The function R is not further defined in this paper.

D. After the key exchange: authenticated encryption

While not strictly part of the handshake protocol defined in this paper, the handshake allows for a "false start" where the encrypted message is sent by Alice to Bob before the entire handshake is finished and Bob has sufficient information to decrypt it.

In the text below, the encryption function ϵ takes two parameters: a shared secret k and a message m, and generates a ciphertext c_m which is an encrypted version of m. Inversely, the decryption function Δ takes the shared secret k and the ciphertext of the message c_m as parameters and reproduces the original message m.

In order to ensure that the messages are always different from one another, thus avoiding certain classes of attacks, the message should include either a nonce or a sequence number at or near the start.

In order to ensure that both parties are using the same keys and the output message is identical to the input message, the ϵ should be an Authenticated Encryption with Additional Data (AEAD) function [23] such as an AES [24] in GCM mode [25].

E. Summary of the building blocks

The function G takes no parameters and generates a private key sk and a public key pk:

$$G \to \langle sk\ pk \rangle$$

The function E takes a public key as a parameter and generates a shared secret and a corresponding ciphertext:

$$E \to pk \to \langle s\ ct \rangle$$

This "encapsulates" a shared secret and produces both the shared secret s and the corresponding ciphertext ct.

The function D takes a private key and a ciphertext, and outputs a shared secret:

$$D \to sk\ ct \to s$$

This is the *decapsulate* function for KEM.

The function H produces a derived key k from a (concatenated set of) random nonce(s) n, an "info" parameter which may be an empty set $\langle\rangle$, and a (concatenated set of) shared secret(s) s:

$$H \to n\ i\ s \to k$$

The function SHA takes one parameter, m, which may be an arbitrary message, and generates a digest of that message:

$$SHA \to m \to d$$

The function R takes no parameters and generates a random value:

$$R \to r$$

The function ϵ takes a shared key k and a message m and encrypts it:

$$\epsilon \to k\ m \to c_m$$

This produces a ciphertext c_m.

The function Δ takes the same shared key used in ϵ and the ciphertext c_m it produces:

$$\Delta \to k\ c_m \to m$$

This produces the original message m from the ciphertext c_m. □

III. Protocol

Note: When a function is invoked to retrieve a value, this is denoted using a $\leftarrow$, so

$$r \leftarrow F\ a$$

denotes the value r being obtained from the function F using parameter a.

A. Prior to the session

Given:

1) Alice has generated her key pair $\langle sk_A\ pk_A \rangle$
2) Alice has calculated the fingerprint of her public key $f_A \leftarrow SHA\ pk_A$
3) Bob has generated his key pair $\langle sk_B\ pl_B \rangle$
4) Bob has shared his public key pk_B with Alice
5) Alice has shared her public key pk_A with Bob
6) Alice wants to send a message m to Bob

B. Session initiation request (Alice)

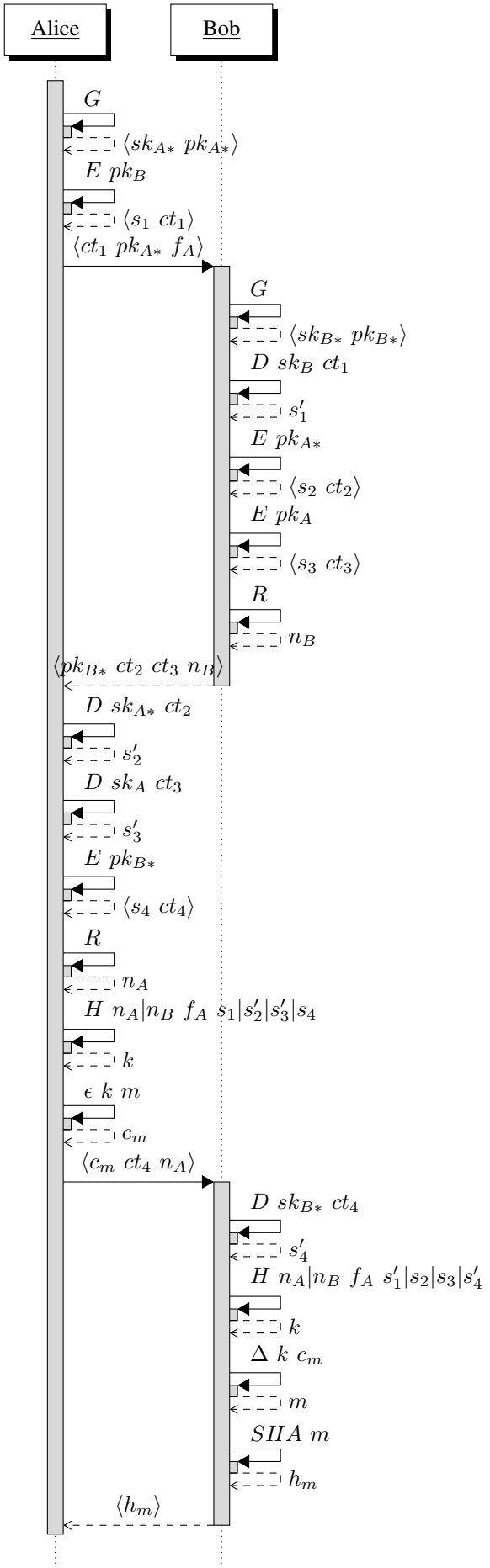
1) Alice generates an ephemeral key pair

$$\langle sk_{A*}\ pk_{A*} \rangle \leftarrow G$$

2) Alice encapsulates a first shared secret

$$\langle s_1\ ct_1 \rangle \leftarrow E\ pk_B$$

3) Alice sends $\langle ct_1\ pk_{A*}\ f_A \rangle$ to Bob, including her public key fingerprint to facilitate his identifying the sender of this first message



C. Session initiation response (Bob)

1) Bob receives $\langle ct_1\ pk_{A*}\ f_A\rangle$
2) Bob generates an ephemeral key pair

$$\langle sk_{B*}\ pk_{B*}\rangle \leftarrow G$$

3) Bob decapsulates the first shared secret

$$s'_1 \leftarrow D\ sk_B\ ct_1$$

Note: $s_1 = s'_1$
4) Bob encapsulates a second shared secret

$$\langle s_2\ ct_2\rangle \leftarrow E\ pk_{A*}$$

5) Bob encapsulates a third shared secret

$$\langle s_3\ ct_3\rangle \leftarrow E\ pk_A$$

6) Bob generates a random nonce

$$n_B \leftarrow R$$

7) Bob sends $\langle pk_{B*}\ ct_2\ ct_3\ n_B\rangle$ to Alice

D. Session completion request (Alice)

1) Alice receives $\langle pk_{B*}\ ct_2\ ct_3\ n_B\rangle$
2) Alice decapsulates the second shared secret

$$s'_2 \leftarrow D\ sk_{A*}\ ct_2$$

Note: $s_2 = s'_2$
Alice may now destroy sk_{A*}.
3) Alice decapsulates the third shared secret

$$s'_3 \leftarrow D\ sk_A\ ct_3$$

Note: $s_3 = s'_3$
4) Alice encapsulates a fourth shared secret

$$\langle s_4\ ct_4\rangle \leftarrow E\ pk_{B*}$$

5) Alice generates a random nonce

$$n_A \leftarrow R$$

6) Alice combines the nonces, the fingerprint she sent in the session initiation request message, and the four shared secrets

$$k \leftarrow H\ n_A|n_B\ f_A\ s_1|s'_2|s'_3|s_4$$

7) Alice encrypts her message using the new key

$$c_m \leftarrow \epsilon\ k\ m$$

8) Alice sends $\langle c_m\ ct_4\ n_A\rangle$ to Bob

E. Session completion response (Bob)

1) Bob receives $\langle c_m\ ct_4\ n_A \rangle$
2) Bob decapsulates the fourth shared secret

$$s'_4 \leftarrow D\ sk_{B*}\ ct_4$$

Note: $s_4 = s'_4$
Bob may now destroy sk_{B*}.
3) Bob combines the nonces, Alice's fingerprint f_A, and the shared secrets

$$k' \leftarrow H\ n_A|n_B\ f_A\ s'_1|s_2|s_3|s'_4$$

Note: $k = k'$
4) Bob decrypts the message

$$m' \leftarrow \Delta\ k'\ c_m$$

Note: $m = m'$
5) Bob calculates a SHA of the message

$$h_m \leftarrow SHA\ m$$

6) Bob sends back the SHA $\langle h_m \rangle$ as an acknowledgement of receipt.

Note that, while $s_1 = s'_1$, $s_2 = s'_2$, $s_3 = s'_3$, $s_4 = s'_4$, and $k = k'$, only $m = m'$ can be proven directly using an AEAD, as described above. As the shared secrets are all input parameters to generate k and k', and the function H used to generate it will almost certainly[4] result in different values for k and k', which in turn will result in the failure of the Δ function to decrypt c_m.

From here on, Alice and Bob share a secret key k (or k' as it's known to Bob) for which Eve would need both private keys from either party to recover it. One of those private keys on either side (sk_{A*} on Alice's side and sk_{B*} on Bob's side) is ephemeral and should be destroyed as soon as it has been used to decapsulate the corresponding shared secret. The fact that these are ephemeral keys, along with the two nonces, provide the perfect forward secrecy.

Note that keying material is sent with the first payload-carrying message. Also note that the protocol described here does nothing to protect against replay attacks or other, similar, attack vectors: the use of a sequence number and/or some key ratcheting scheme would be advisable.

Finally, note that the session completion response message sent by Bob only proves that Bob has been able to set up the keys if Alice sent a message payload in the session completion request message. Including such a message (with a nonce or sequence number of some sort to protect the key and protect against replays) is therefore *not* optional.

□

IV. Analysis

As set out in the introduction above, this protocol needs to meet a certain set of requirements to be useful. Each of these requirements is analysed below, to determine if, and how, the protocol meets them.

[4] H is based on an HKDF, which in turn is based on secure hash functions, so a slightly different set of input parameters should result in wildly different outputs.

A. Mutual authentication

There are two aspects to mutual authentication: a message from Alice to Bob, $m_{A\rightarrow B}$ should only be readable by Bob, and Bob should be able to ensure that Alice sent it.

Within the protocol itself, Alice and Bob exchange several messages, but we also start with a set of assumptions:

1) Alice has a copy of pk_B which she knows to be from Bob
2) Bob has a copy of pk_A which she knows to be from Alice

The first message Alice sends to Bob, $\langle ct_1\ pk_{A*}\ f_A \rangle$, cannot be authenticated by Bob: while it does contain information for Alice to know only Bob can use it (i.e. ct_1 can only be used with Bob's static private key sk_B), it does not contain any information for Bob to verify the message is from Alice. Hence, while Alice can have a certain level of trust in the session she is now building, Bob still has no such trust level, and should not assume, yet, that pk_{A*} belongs to Alice.

The second message, sent by Bob to Alice, contains $\langle pk_{B*}, ct_2, ct_3, n_B \rangle$ and can also not be authenticated by Alice. It does contain two ciphertexts, ct_2 and ct_3, that can only be used by Alice, the former of which also shows that this message is a response to the message sent by Alice, as it uses the ephemeral public key pk_{A*} included by Alice in that message. A state machine implementing Alice's side of this protocol can use that information as confirmation that the message has been received, to move to the next state.

The third message, sent by Alice to Bob and containing $\langle c_m, ct_4, n_A \rangle$, is the first in the sequence that proves to Bob that Alice initiated the session, and that she has received the materials sent by Bob: it contains an encapsulated secret ct_4 that was created using Bob's ephemeral public key pk_{B*}, tying it back to the message sent by Bob in response to the session initiation request, and contains the encrypted message c_m, which can only be decrypted using keying materials that require knowledge of all four key pairs (two private keys and two public keys). Bob's ability to decrypt c_m therefore proves to him that:

1) Alice sent the first message, and owns the private key corresponding to pk_{A*}
2) Alice received the second message
3) Alice encrypted m

At this point, Alice also knows that only Bob (and Alice herself) can decrypt c_m: she has used pk_B as one of the public keys to generate the secret used to encrypt m, and knows that Bob can only decrypt it if he owns both the private keys corresponding to pk_B and pk_{B*}. Based on the assumptions above, that can only be Bob.

For Alice to have a confirmation that Bob has received and decrypted the message $c_m \rightarrow m$, an acknowledgement is sent by Bob containing a (public) value that is some (safe) function of m. To illustrate, a SHA is used here, but if this protocol is used to secure some other protocol, the appropriate response for that other protocol may be used as well, provided it is encrypted with the new shared secret key k.

B. Resistance to eavesdropping

Assuming Eve can eavesdrop on every message exchanged between Alice and Bob but has neither sk_A nor sk_B in her posession:

1) She does not have sufficient information to decapsulate ct_1, lacking sk_B
2) She does not have sufficient information to decapsulate ct_2, lacking sk_{A*}
3) She does not have sufficient information to decapsulate ct_3, lacking sk_A
4) She does not have sufficient information to decapsulate ct_4, lacking sk_{B*}

She therefore cannot reproduce k, and cannot decrypt c_m.

C. Resistance to man-in-the-middle attacks

Assuming Mallory can both eavesdrop and modify arbitrary messages exchanged between Alice and Bob, but has beither sk_A nor sk_B in his posession:

1) Mallory can alter the first message Alice sends to Bob by substituting either ct_1 with ct_1' or pk_{A*} with pk_{M*}, or both. If ct_1' is created using Bob's public key pk_B this altered message will be indistinguishable from a valid message from Alice, from Bob's point of view.
2) Mallory can alter the second message, sent by Bob to Alice, by substituting any of pk_{B*}, ct_2, ct_3, or n_B values for values of his own making, all of which can be seemingly valid. Mallory can do this regardless of whether he has substituted any values in the first message.
3) Mallory can substitute the ct_4 value and n_A value in the third message, but lacking sk_A cannot create a valid value for s_3 and, lacking sk_B, cannot create a valid value for s_1.

Hence, while Mallory can spoof the first two messages of the exchange and part of the third one, doing so will result in different values for k at Alice's end and at Bob's end, resulting in the Δ function failing and returning $\perp$, and Mallory cannot obtain the value of k.

As noted above, this depends on Mallory knowing neither sk_A nor sk_B, thus being unable to obtain s_1 and s_3, and thus being unable to impersonate either Alice or Bob.

D. Perfect forward secrecy

Data-carrying messages exchanged between Alice and Bob use a shared secret k for encryption. To generate this shared secret, four private-public keypairs are used, two of which ($\langle sk_{A*}\ pk_{A*}\rangle$ and $\langle sk_{B*}\ pk_{B*}\rangle$), are ephemeral. Provided sk_{A*} and sk_{B*} are both discarded as soon as possible (see above), if an adversary such as Mallory obtains either sk_A or sk_B, or even both, they will not have sufficient information to reconstitute k as both sk_{A*} and sk_{B*} are still missing, both of which would be needed to obtain k from the public data exchanged between Alice and Bob.

Note that, because k is a symmetric secret, obtaining a copy of *it* is sufficient to decrypt any messages exchanged between Alice and Bob. If this is a real danger for either party, these keys should be short-lived and use of a key ratcheting scheme may be indicated.

E. Algorithm requirements

The functions defined above (G, E, D, H, SHA, R, ϵ, and Δ) place no special requirements on the underlying algorithms not already put on them by the class of algorithm they belong to: they must be secure, have reasonable performance, and fit the general API expected of these types of algorithms.

This means that the protocol described in this paper can be used with a variety of KEM implementations, SHA implementations, random number generators, and AEAD implementations. It also means that the KEM algorithms used with the different keys may also be different, as the secrets generated by the KEM functions are combined in such a way that loss of secrecy of any one of the four generated secrets does not result in the loss of secrecy of the derived shared secret [26].

F. Comparison to KEM.KE, KEM.UAKE, and KEM.AKE protocols

The protocol presented in this paper, which I will call "KEM.MAKE", meaning "mutually authenticated key exchange" for consistency with [6], is built on the same principles as the key exchange (KE), one-sided authenticated key exchange (UAKE), and authenticated key exchange (AKE) protocols presented protocols presented in that paper, but has several advantages over those protocols. By using one extra ephemeral keypair over AKE, we gain perfect forward secrecy at the cost of one key generation, one encapsulation, and one decapsulation. This cost is distributed such that, with the proposed MAKE protocol, both sides perform one G, two E, and two D operations (whereas in AKE, Alice performs on G, one E, and two D operations and Bob performs two E and one D operation).

The ability to mix KEM algorithms is similar between AKE and MAKE, as both use a function H to combine the generated shared secrets.

□

V. Conclusion

I have presented a key exchange protocol permitting mutual authentication and perfect forward secrecy, using only a secure KEM, a secure SHA, and a strong RNG (which is also a prerequisite for a secure KEM). Employing an AEAD, this protocol can be used to create a secure session with an authenticated symmetric key. Loss of one or even both of the static private keys used by the parties does not allow an attacker to recover the secret symmetric key, even if they have recorded all of the messages exchanged between the parties and, provided different KEM algorithms are judiciously combined, loss of security of one of those algorithms also does not compromise the security of the protocol nor allow an attacker to recover the key due to the way the generated shared secrets are combined into a single shared secret key.

■

APPENDIX A
NOTATION

In this paper, functions are denoted as uppercase letters such as G, E, and H. Where necessary, the Greek alphabet may be used to avoid overlap (e.g. E and ϵ are both functions, the former encapsulates and the latter encrypts). Greek letters are also used to indicate sets (e.g. $k \in \kappa$) but it should generally be clear from the context which use-case is being employed.

I use a subscript to indicate who a key belongs to (e.g. sk_A for a key that belongs to Alice) and whether it's ephemeral (e.g. pk_{B*} is an ephemeral public key that belongs to Bob).

A unary function is defined as follows:

$$F \rightarrow r$$

This function takes no parameters and outputs a single value r. A unary function that outputs a tuple of values is noted as:

$$F \rightarrow \langle a, b, c \rangle$$

Functions that take parameters are noted with an extra $\rightarrow$, so a function that takes two parameters and outputs a single value is noted as

$$F \rightarrow p, q \rightarrow r$$

When a function is intended to be composable, additional arrows may be used. For example: a function F that takes a parameter a and a parameter b, and outputs a result r, where the function F having been given the parameter a may be treated as a new function, is denoted as

$$F \rightarrow a \rightarrow b \rightarrow r$$

□

REFERENCES

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] T. Hoefler, T. Häner, and M. Troyer, "Disentangling hype from practicality: on realistically achieving quantum advantage," *Communications of the ACM*, vol. 66, no. 5, pp. 82–87, 2023.

[3] a. E. T. John Preuß Mattsson, Ben Smeets, "Quantum-Resistance Cryptography."

[4] E. Parker and M. J. Vermeer, "Estimating the energy requirements to operate a cryptanalytically relevant quantum computer," *arXiv preprint arXiv:2304.14344*, 2023.

[5] J. R. Biden, "National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems," 2022.

[6] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM," Cryptology ePrint Archive, Paper 2017/634, 2017, https://eprint.iacr.org/2017/634. [Online]. Available: https://eprint.iacr.org/2017/634

[7] "Post-Quantum Cryptography," https://web.archive.org/web/20230630015245/https://csrc.nist.gov/projects/post-quantum-cryptography, accessed: 2023-07-24.

[8] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[9] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[10] "PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates," https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4, accessed: 2023-09-16.

[11] N. Aragon, P. S. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Guneysu, C. A. Melchor *et al.*, "Bike: bit flipping key encapsulation," 2017.

[12] H. Singh, "Code based cryptography: Classic mceliece," *arXiv preprint arXiv:1907.12754*, 2019.

[13] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I. Bourges, "Hamming quasi-cyclic (hqc)," *NIST PQC Round*, vol. 2, no. 4, p. 13, 2018.

[14] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*. Springer, 2011, pp. 19–34.

[15] W. Castryck and T. Decru, "An efficient key recovery attack on sidh," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 423–447.

[16] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Annual international cryptology conference*. Springer, 1991, pp. 433–444.

[17] M. Bellare and B. Yee, "Forward-security in private-key cryptography," in *Topics in Cryptology–CT-RSA 2003: The Cryptographers' Track at the RSA Conference 2003 San Francisco, CA, USA, April 13–17, 2003 Proceedings*. Springer, 2003, pp. 1–18.

[18] J. H. Burrows, "Secure hash standard," *Federal information processing standards publication*, pp. 180–1, 1995.

[19] H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," Tech. Rep., 2010.

[20] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I 37*. Springer, 2017, pp. 570–596.

[21] Q. H. Dang, "Secure hash standard," 2015.

[22] E. B. Barker and J. M. Kelsey, *Recommendation for random bit generator (RBG) constructions*. US Department of Commerce, National Institute of Standards and Technology ..., 2012.

[23] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 98–107.

[24] V. Rijmen and J. Daemen, "Advanced encryption standard," *Proceedings of federal information processing standards publications, national institute of standards and technology*, vol. 19, p. 22, 2001.

[25] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *submission to NIST Modes of Operation Process*, vol. 20, pp. 0278–0070, 2004.

[26] F. Giacon, F. Heuer, and B. Poettering, "KEM Combiners," in *Public-Key Cryptography – PKC 2018*, M. Abdalla and R. Dahab, Eds. Cham: Springer International Publishing, 2018, pp. 190–218.