AN ASYMMETRIC-BASED POST-QUANTUM CRYPTOGRAPHIC PROTOCOL FOR SPACE MISSIONS

Requirements and possible protocol designs

Andreas Hülsing Tanja Lange Fiona Johanna Weber

May 1, 2024

Contents

Author list in alphabetical order, see: https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf

1. Introduction

Communication between satellites and mission control centers is currently protected via the Space Data Link Security Protocol (SDLS). At its core this protocol uses symmetric algorithms (most importantly symmetric encryption) to secure communication.

Key-updates are possible in SDLS, but they too are only using symmetric primitives. This has the advantage that they are not immediately vulnerable to quantum-computers, but the major downsides that a key-update cannot recover security after a key-compromise if the adversary receives a network-transcript of the key-update, that the approach does not scale well to decentralized networks, and that it limits operational efficiency, especially in federated operations.

The goal of this study is therefore to design a key-update/establishment protocol that relies on asymmetric algorithms for authenticity and confidentiality. A slightly unusual aspect of this goal is that the necessary protocol for this is not just a handshake-phase of a more complex protocol that is only ever run in conjunction with a data-transmission protocol, but actually independent.

Because of the looming threat that quantum-computers may in the not so far future be able to break all asymmetric primitives that are currently in widespread use, the protocol should include other, so called “post-quantum” algorithms that are designed to withstand that threat. The primary source for post-quantum algorithms is the post-quantum competition run by NIST, that recently selected four winners, that are bound to be standardized. Beyond these winners there also a handful of more specialized algorithms that were standardized by the IETF as well as algorithms that have not (yet) been chosen by NIST, but were declared to be acceptable by various European government bodies or are in widespread use. (See Sections 5.1 and 5.2 for more details.)

2. Background

In the following we provide some background knowledge and use the opportunity to establish notation. While we assume familiarity with general cryptographic concepts like public key encryption and signature schemes, cryptographic hash functions and message authentication codes (MAC), we give some background on authenticated key exchange (AKE), Diffie-Hellman Key Exchange (DHKX), and Key Encapsulation Mechanisms (KEMs). Afterwards we give a brief tour of the Noise protocol framework which provides a compact notation to describe AKE protocols.

2.1. Authenticated Key-Exchange

An Authenticated Key-Exchange (AKE) is a protocol in which two parties negotiate a shared secret that is only known to them in a way that one (one-sided) or both (mutual) parties have to prove their identity. This is commonly done by means of demonstrating knowledge of a secret private key for which the associated public key is known to the other party. The established shared secret can then be used to secure communication using symmetric cryptography. Well known examples are the hand-shake phases of TLS, IKE, WireGuard, or Signal.

Early AKE-schemes, such as the ones used in SSL assumed that no private keys would ever be compromised. These schemes lost confidentiality of all past (and present) messages whenever a private key was compromised, e.g., when a server was taken over by a malicious party. Because of this, modern versions added the requirement of forward-secrecy: past communication should remain confidential even if the long-term private key is later compromised. Signal, WireGuard, and modern versions of TLS all have this property.

A common way to achieve forward-secrecy is the use of an ephemeral key-exchange. A key-exchange in the context of this paragraph is any protocol that creates a shared-secret between two honest parties, that a passive network-attacker cannot learn. In this study we call a key-exchange "ephemeral" if no component, except for the resulting exchanged key is used outside the specific interaction[1]. Ephemeral asymmetric key-pairs in particular have to be generated at the start of the interaction and must be fully discarded at the end of it. This is in contrast to the traditional use of asymmetric encryption where public keys remain valid for a long time and can be used by many parties to encrypt messages.

Terminology-wise we note that forward-secrecy is sometimes also called "perfect forward-secrecy", to distinguish it from "weak forward-secrecy". We don't use these terms because weak forward-secrecy is too weak to be of much interest and because there is nothing perfect about (regular) forward-secrecy: Conventionally "perfect" is used to state that a property holds unconditionally without exception, even with unbounded adversaries such as the confidentiality of a correctly used one-time-pad or the completeness of many traditional cryptographic schemes. Pretty much all protocols that claim forward-secrecy today, can however be broken by attacks that run in exponential-time, contradicting the idea that the property holds perfectly.

More recently the complementary property has also come into the focus of AKE-designers: Post-Compromise Secrecy. It means that if one party in an interaction gets only temporarily compromised for some period of time, i.e., the honest user regains exclusive control of all their secrets (for example through a device-update that removes malware), confidentiality of the communication is reestablished for all future messages. AKEs that use ephemeral key-exchanges to achieve forward-secrecy usually get post-compromise secrecy for free, though it is possible to create counter-examples to that rule.

Further properties of AKEs that are nowadays discussed include deniability, anonymity

[1]Less restrictive definitions of "ephemeral" use the term to refer to short-lived keys but permit usage in different interactions

and group key agreement (the establishment of secrets that are shared between more than two parties), none of which are considered relevant for the given use-case.

2.2. Diffie-Hellman Key Exchange (DHKX) and Key Encapsulation Mechanisms (KEM)

Modern AKE protocols are built from more basic primitives. The most commonly seen primitive is Diffie-Hellman Key Exchange (DHKX), a two party, non-interactive key exchange protocol. In DHKX, both parties are in possession of a key pair consisting of a private key a (resp. b) and a public key A (resp.B). When party $\mathbf{A}$ receives the public key B they can derive a shared secret $k = \mathsf{DH.derive}(a, B)$ applying the derive function to their private key a and the public key B. The same shared secret $k = \mathsf{DH.derive}(b, A)$ can be computed by $\mathbf{B}$ using their private key and $\mathbf{A}$'s public key. Note that this protocol does not require any interaction between the two parties if the public keys are known (e.g., were retrieved from a public server). So far, there is no post-quantum secure Non-Interactive Key Exchange (NIKE) that is commonly considered secure. (First proposals exists though [16, 21, 41, 18]). The NIST post-quantum cryptography standardization process did not consider DHKX-like schemes.

In place of NIKEs, NIST selected Key Encapsulation Mechanisms (KEMs) which are closely related to (and usually built from) public key encryption (PKE) schemes. A PKE takes as input the receiver's public key and a message and produces the ciphertext encrypting the message as output. The receiver decrypts the ciphertext with their private key to obtain the message. In practice, a PKE is most often used to actually encrypt a random message which is afterwards hashed to obtain a shared key to be used with symmetric cryptography. A KEM is in some sense a limitation of a PKE to this very use-case. Hence, a KEM is merely used for key transport. A KEM takes as input only the receiver's public key and produces two outputs: the ciphertext and a shared key. Only the ciphertext is sent to the receiver who then uses their private key to decapsulate the ciphertext and obtain the same shared key.

Ideally this always works, but for many KEMs in the NIST PQC competition there is a small probability δ that the decapsulation will not return the same shared secret that the encapsulating party received. Most KEMs in the competition that suffer from decryption failures use the Fujisaki-Okamoto (FO) transform to achieve IND-CCA-2 security (sometimes referred to as active security). More precisely, they use FO with implicit rejection, i.e., they output a pseudo-random value if a decryption failure happened. These failures are identified by some internal check failing or decapsulation outputing a failure. If this was caused by an honest failure rather than by an attack, the sender and receiver end up with mismatching secrets. This property is generally known as *δ-correctness* in contrast to the traditional *perfect correctness* where $\delta = 0$.

RSA key transport used in earlier versions of TLS implemented a KEM and one can also view DHKX as a KEM: the sender's public key forms the ciphertext and the shared key

k above is the shared key in the KEM [2].

One can also generically turn a PKE scheme into a KEM. Key generation creates a public-private key pair as for PKE. Encapsulation samples a random message m from the message space and uses the encryption algorithm to produce the ciphertext c. The shared key is then $k = h(m)$ for some key-derivation function h. The receiver decrypts c to m and obtains the same $k = h(m)$.

2.3. Noise and Noise-patterns

Noise is a generic framework for key-exchange protocols (authenticated and unauthenticated). It is built around DHKX which is used to establish both confidentiality and authenticity. Given that DHKX does not withstand attacks using quantum computers, Post-Quantum Noise was developed. In brief, Post-Quantum Noise makes use of KEMs in place of DHKX. If the used KEMs withstand quantum attacks then so does the bigger protocol.

Noise introduced a very compact way to describe the core idea of key-exchanges between two parties called the initiator and the responder. The initiator initiates the communication and is placed on the left. Consider the following example:

```
<- s
...
-> psk, skem, e, s, sig
<- ekem, s'[opt1]
```

Each line consists of a packet being sent from one party to the other. The ASCII-arrows indicate the direction in which that packet is sent, `->` indicates that the sender is the initiator of an exchange, `<-` that the sender is the responder. The `...` separates any information-exchange that occurred before the protocol is run from the actual protocol. In this example the initiator received the receiver's long-term public key `s` beforehand out-of-band (one may think of this as a setup).

A noise pattern only defines which cryptographic objects are exchanged between the two parties. A modern key exchange protocol usually consists of several actual key exchanges. In Noise as well as Post-Quantum Noise, parties start deriving session keys (and encrypting all traffic with these using AEAD) as soon as possible from the currently established session key (if existing) and any newly derived shared secret. I.e., as soon as the first DHKX public key is received, the shared secret k is derived and used to derive or update the session key. Similarly, as soon as a KEM ciphertext is received, decapsulation is used to derive the shared secret k and derive or update the session key.

The various tokens then have the following meanings (and implications):

[2]While this is functionally a KEM, one has to compute the shared key of the KEM as the hash of the shared key generated by DH and the sender's public key to achieve an actively secure KEM.

- `psk` refers to a pre-shared secret that is used to establish an initial shared secret to encrypt the further communication.
- `s` refers to a long-term public-key. If a shared secret has already been established, this key will be (symmetrically) encrypted in transit.
- `e` refers to an ephemeral public key. In Noise and PQC Noise it is not encrypted in transit, however we follow the maxime that everything is encrypted that can be encrypted and thus deviate from this convention.
- `skem` refers to a ciphertext for the peer's long-term KEM public-key. This will create/update the shared-secret in a way, such that it remains secure, if the receiver's long-term key and the sender's randomness are not compromised. If there is a pre-shared secret this ciphertext will be encrypted in transit.
- `ekem` refers to a ciphertext for the peer's ephemeral public-key. This will create/update the shared-secret in a way, such that it remains secure, if the sender's randomness and receiver's ephemeral secrets are both not compromised. This ciphertext is never encrypted in Noise and PQNoise but we encrypt it where possible.
- `sig` refers to a signature under the sender's long-term key whose message is a hash of the full transcript of the handshake up to that point.
- `s'` is an extension that we propose in this work. It refers to an updated long-term public-key that is supposed to replace the old long-term key of the sending party after the successful completion of the handshake.
- `X[optN]`, where `X` is one of the tokens above and `N` is an integer, is a second extension that we propose in this work. We use it to indicate that the protocol can run as if the token `X` was part of the pattern or as if it was not there, the decision occurring at runtime. The integer `N` is simply used to separate multiple independent optional components.
- `confirm` is an extension that indicates a key confirmation message which sends an AEAD ciphertext for an empty message to prove knowledge of the current session key. (In Noise this is given by an arrow without tokens, we only make this explicit.)
- (There are further tokens for various combinations of Diffie-Hellman secrets which are out-of-scope for this work.)

The above example therefore reads as follows:

- The initiator knows the responder's long-term public-key and they furthermore have a pre-shared secret.
- Both parties derive a key k_0 from the pre-shared secret.
- The initiator uses the responder's long-term public-key to encapsulate a secret for him and derives the shared key k_1 from it and k_0.
- The initiator uses k_1 to encrypt a freshly generated ephemeral public-key and her long-term public-key.
- The initiator uses her long-term private signing key to sign the full transcript up to that point and sends it to the responder.
- The responder decrypts/decapsulates everything he received, verifies the signature, and generates a ciphertext and shared secret for the initiator's ephemeral KEM

key and derives k_2 from the shared secret and k_1.
- The responder may optionally encrypt a new long-term public key with k_2 and sends his packet to the initiator.

We note that all secrets k_i are derived from all shared-secrets that have been computed up to that point. We refer to the last k_i that both parties compute as the pre key; the following discusses how the final key is derived from this pre key and the transcript.

3. Requirements & Constraints

In this section we discuss the requirements and constraints agreed upon with ESA following a series of meetings with ESA experts. We begin with the requirements. Afterwards we list the constraints. For each of them we do a brief pre-assessment.

3.1. Requirements

The following requirements detail the main objective of this study.

3.1.1. Authenticity

Authenticity refers to the property that parties can be certain that they are interacting with the party that they think they are interacting with.

In some missions, where security is of high priority, authenticity has to be bidirectional.

In some other missions authenticating the satellite to mission control might not be necessary, because the control center knows where the satellite is supposed to be and would use communication channels that are physically very narrow. In any case this is a per-mission property and thus does not have to be negotiated at runtime.

3.1.2. Confidentiality

Confidentiality refers to the property that the content of an interaction remains secret except to the authorized parties. In the context of a key-exchange this content refers to the resulting key being only known to the parties involved in the exchange, not anyone else, including active or passive network-attackers.

Confidentiality of the payload-data is usually a security-goal but sometimes less important than authenticity and availability.

For the purposes of a key-update where a key derived from the shared secret is then used in SDLS this however means that since the knowledge of the shared secret is also used to authenticate a party, full confidentiality of that secret is required regardless.

3.1.3. Availability

In the given setting, availability refers to the ability to communicate with the satellite.

Availability is generally very important. Very short-term loss might however be tolerable. With regards to DoS-protection it is not necessary to consider simultaneous connection-attempts, continuous attempts to authenticate/connect are however in scope.

3.1.4. Hybrid Security

Hybrid systems combine multiple algorithms. Traditionally this expression was used to describe a combination of public-key and symmetric-key cryptography. In the context of post-quantum cryptography the term is now more narrowly used to describe combinations of pre- and post-quantum schemes. Hybrid Security is then the property that a protocol uses multiple algorithms to instantiate a primitive in a way such that the protocol remains secure even if a subset of the algorithms turn out to be insecure.

For the key-update protocol ESA requests that post-quantum primitives are used in a hybrid manner with well-established (pre-quantum) primitives.

The easiest way to achieve this is through the use of hybrid primitives, for example KEMs that are a combination of a lattice-based KEM and a KEM based on an elliptic-curve Diffie-Hellman key-exchange. This approach has several advantages over using the same primitive with different instantiations multiple times: It simplifies the protocol and its analysis, because the added complexity is contained in a sub-component that is not further decomposed in any case. It allows for greater flexibility when choosing or replacing concrete primitives, because the analysis depends less on the concrete instantiations. The main disadvantage of this approach is that it requires the definition of the hybrid primitive, though for the primitives that we consider relevant here (KEMs and Signatures), this is not a real problem, as there are established combiners that we will discuss in Section 5.4.

3.1.5. Adversaries

The attacker-model includes the full range of possibilities, from nation-states to individuals. With regards to security parameters, NIST proposed levels I, III, and V, which correspond to the security of AES with 128, 192, and 256 bit keys against brute-force key search, respectively, are to be considered. While denial-of-service attacks via connection attempts of multiple actors do not have to be considered, multiple connection attempts of a single actor are to be considered.

All considered cryptographic algorithms provide parameters for these security levels. Therefore, this is not problematic. When it comes to availability the best that can be achieved on the protocol level is to avoid the introduction of vulnerabilities that allow an adversary to exhaust system resources with little adversarial effort, e.g., by filling up memory with initial connection attempts.

3.1.6. Ciphersuite negotiation vs. fixed choices

We say that a protocol is cryptographically opinionated if it prescribes certain algorithms without any option to negotiate others at runtime. The opposite of this is where some portfolio of permitted algorithms is defined and the ciphersuite is negotiated as part of the protocol. The latter has the downside that attackers might try to interfere with the negotiation process in a so-called downgrade attack; the former has the downside that weak defaults affect all users.

For the key-update mechanism there is no desire for a ciphersuite-negotiation at runtime, so cryptographically opinionated protocols are fine. A change to a new ciphersuite can be done via a protocol upgrade to a new version that is not expected to interoperate and that may do different things.

3.1.7. Updateable Long-term keys

One requirement that was not explicitly listed in the initial statement of work, but found to be potentially desirable in follow-up discussions is the ability to upgrade the long-term-keys of the parties.

There are three ways to perform a long-term key-update:

- As part of a separate protocol,
- as a part of the key-update mechanism that is always run, and
- as a part of the key-update mechanism that is not always run.

The first option has the disadvantage that it would require the implementation and analysis of another protocol, which might increase the attack-surface.

The second option is the most obvious one, but it essentially turns the long-term keys into short-term keys as well. This has several advantages, such as a significantly reduced attack-surface for replay-attacks, but may come at the cost of larger packets, due to the need to always include the next public-keys.

The third option is essentially a trade-off between typical packet-size and the advantages of single-use identity-keys.

3.2. Constraints

The following constraints clarify the setting in which the protocol will be used. Especially, they point out limitations and constraints imposed on the solution.

3.2.1. Mission Classes

The primarily targeted missions use small to mid-sized commercial or institutional satellites in possibly larger constellations (more than 100 satellites), usually in low earth orbit.

The secondary targets are larger institutional or commercial satellites that use unclassified, standardized security functions.

CubeSat (cubes with 10cm side-length and no more than 2 kg weight) are out of scope for now, but may be interesting later.

Governmental satellites that are used for classified data are out of scope.

With regards to the number of endpoints, the number of mission control systems on the ground is generally very limited (1, or 1 + redundancy), whereas the number of satellites may be comparatively high (in the hundreds or thousands).

Preliminary assessment: We do not foresee these properties to cause any meaningful issues. Even the high end of the number of satellites is rather small compared to what many cryptographic protocols that run over the internet have to deal with.

3.2.2. Execution Environment

The IT-infrastructure on the ground is generic and for the purposes of this project not meaningfully constrained.

The On-Board Computers (OBCs) that are still in use include some older chips such as GR740 and NG-Ultra; on new system the use of 8 MIPS (Millions of Instructions Per Second, not to be confused with the MIPS architecture)) is likely acceptable. On old systems there is essentially no idle-time, whereas newer systems may have a little bit of it (2%, which equates the aforementioned 8 MIPS). Beyond that, there are no real additional constraints on energy-use.

The storage-capacity on the satellite may lie between 256 GB and 1 TB.

Preliminary assessment: Given the performance of most post-quantum primitives we do not expect any of these environments to be meaningful constraints of the protocol.

3.2.3. Connection

The latency of a connection starts at 40-50 ms for low earth orbit and grows with the distance from there. For communication with Mars for example it lies in the 5-20 min range.

Packet sizes are around 1 KB. Packet-Fragmentation should preferably be avoided, but this is no hard requirement.

The Throughput depends on the concrete link:

- Sband: 1 Kbit/s - 100 Kbit/s
- Xband: up to 100s of Mbit/s, usually D/L only
- KA Band: up to several Gbit/s, mainly D/L

- Typical: 256kbps up, 2Mbps down. some uplinks 64kbps or 128kbps

The total communication-time over these links should be limited to 1 minute, because the contact time in low earth orbit lies in the 7-10 minute range.

Preliminary assessment: In general this will not cause any issue for the protocols we design. Sizes depend on the cryptographic schemes used to instantiate them though. For most schemes these bandwidths are unlikely to cause significant issues, as the resulting packet sizes fall into the 3-5 KB range. However, there are exceptions when using the most conservative cryptographic schemes. In these cases, care has to be taken. Fragmentation is unavoidable for sending ephemeral keys for all systems, it can be avoided for KEM ciphertexts for some systems.

3.2.4. Public Key Infrastructure

A Public Key Infrastructure (PKI) can be used to easily update long-term keys of various parties.

As of now there are some standardization efforts, but nothing is implemented yet.

Preliminary assessment: Because there is currently no PKI in place, we will assume that long-term keys of the possible peers are pre-installed.

4. Design Considerations

In a deviation from the more traditional design, where keys are always established as part of the handshake of a data transmission protocol, the key-update/establishment protocol for SDLS will be a stand-alone protocol that can be run independently from the data-transmission protocol. Not only does this reduce difficulty of an analysis, it also decreases the complexity (and thereby the attack-surface) and allows to update keys less often which reduces the communication and computational load. Additionally it minimizes the changes required on the SDLS protocol itself.

We deviate from Noise in the way we derive the final key because we are designing a key exchange rather than a full secure channel protocol which also transmits data. Firstly Noise uses its final keys already during the handshake-phase to encrypt messages before receiving a key-confirmation from the peer. In order to ensure that the resulting keys of the key-update mechanism are truly fresh and can be used without any worries, we instead derive the final keys only once everything in the handshake has been completed by the party in question.

Furthermore we combine the resulting pre-keys with the handshake-hash to improve the robustness against replay-attacks. Noise does not do this as it also covers the data communication layer. There, it includes a transcript-hash as associated data in every ciphertext sent, so replays could not send any new data. In our case we treat SDLS as a standalone protocol to which we need to deliver a fresh key.

Lastly we always insist on key-confirmations. This is an aspect of signature-based protocols that could potentially save half a round-trip at the expense of a much more fragile protocol that has to use “external” replay-protection and may in the most extreme cases even result in an unrecoverable loss of a shared communication-key with the satellite. We provide a much more detailed discussion of this property in Section 6.4 after we have establish more technical context.

Considering that ESA stated that it is often infeasible to impersonate a satellite due to the physical properties of the data-link between orbit and ground, we also provide a variant that does not authenticate the responding party (here: the satellite) in exchange for a slightly more lightweight protocol. While the use of such a protocol requires careful consideration, a scenario in which a channel can be assumed to be partially authenticated can be an instance where the bandwidth-savings may be enough to justify the use of the weaker protocol. We thus provide such a version as an additional variant.

We remark here that such a version assigns trust to whoever provides the out-of-band authentication, which in the motivating example would be the ground-stations, not mission-control. Whether this trust is reasonable is not something that we can comment on, beyond *strongly* advising to thoroughly evaluate that question before any use of protocols that are only partially authenticated.

5. Available PQ KEMs and Signatures

In this section we provide an overview of the available post-quantum KEMs and signature schemes. We focus on NIST finalists as well as alternates and discuss their advantages and disadvantages. For a more detailed overview covering more background see [12], for NIST’s motivation see [2].

5.1. KEMs

The following KEMs managed to become finalists or alternate candidates in the third round of the NIST-competition; the first three are what we consider to be the most relevant ones for this work.

Kyber: By now the NIST process finished and selected Kyber [43] as the new standard for key encapsulation. This makes Kyber an obvious choice to consider. Kyber is a lattice-based scheme that (on a high level) follows the LPR blueprint [32] to design a lattice-based PKE which is then turned into a KEM using the FO transform [24]. Kyber has very good overall performance and reasonably small key- and ciphertext-sizes. The security of Kyber is based on the hardness of standard lattice problems over module-lattices. The wider community considers the security of Kyber well understood.

Classic McEliece: The McEliece cryptosystem dates back to the beginnings of public key cryptography. It is widely considered one of the most conservative KEM constructions. For that reason, the NIST candidate Classic McEliece [3] has been officially recommended by the German BSI, the French ANSSI and the Dutch National Cyber Security Center. The scheme has a very interesting performance characteristic: While public keys are massive, ciphertext size is far smaller than for any other PQ-KEM. Moreover, implementations are fast. The small ciphertexts can make Classic McEliece an interesting choice in settings where public keys can be transmitted out-of-band or ahead of time.

Frodo: FrodoKEM [35] is a lattice-based KEM that, like Kyber, follows the LPR + FO blueprint. The big difference to Kyber is that Frodo uses unstructured lattices instead of module lattices which is generally considered the more conservative choice. However, this comes at a cost. Frodo has significantly worse sizes compared to Kyber. Still, due to the conservative security, Frodo is approved by BSI/ANSSI/NL-NCSC.

Saber: The Saber [20] construction is extremely close to that of Kyber with the one major difference that it uses rounding instead of adding an error. This bases security on the Learning-With-Rounding (LWR) problem instead of the Learning-With-Errors (LWE) problem. Consequently, it is in all performance aspects quite similar to Kyber, while possibly somewhat more efficient on constrained devices. NIST's report stated as reason to select Kyber over Saber was that for LWR fewer cryptanalysis results were available than for LWE.

NTRU: The NTRU [17] cryptosystem is the oldest lattice-based encryption scheme. It is based on the NTRU problem, a problem over certain structured lattices. While performance is overall good, it is slightly worse than that of Kyber, especially key generation is somewhat slower which affects usage for ephemeral keys. The scheme has seen a long history of cryptanalysis without any major attacks (this is in contrast to NTRUSign which was broken). Based on this, also the security of NTRU is widely considered well understood by now. Google announced [28] that they are using NTRU to secure internal communications.

NTRU Prime: The general idea of NTRU Prime [8] is to move lattice-based schemes to a different kind of lattice for which the team claims that it provides less exploitable structure than other lattices. The NTRU Prime family contains two KEMs: Streamlined NTRU Prime (sntrup) and NTRU LPRime (ntrulpr). The former is a variant of NTRU over the NTRU Prime lattice, the latter is a variant of the LPR approach, similar to Kyber & Co over the NTRU Prime lattice. Performance-wise the two schemes are similar to the originals with the originals being slightly larger (as NTRU Prime uses rounding), though ciphertexts are marginally larger because the system uses an extra 32 bytes for a key confirmation hash. NIST stated that they did not select NTRU Prime because there was insufficient evidence for the security advantages of the scheme. Streamlined NTRU Prime is used as default in OpenSSH [36].

SIKE: The SIKE scheme was a KEM-candidate in Round 4 of the NIST competition based on the isogeny problem with additional information between supersingular curves. However, SIKE has been broken [15, 33, 40] in 2022 and the underlying problem has been shown to be solvable in polynomial time.

BIKE and HQC: NIST moved two more code-based KEMs into Round 4, namely BIKE [6] and HQC [1]. These use structured codes similar to the cyclotomic rings in structured lattices. The use of structured codes allows them to achieve significantly smaller public keys than McEliece at the expense of larger ciphertexts. The ratio of ciphertext and public key size matches that of systems based on lattices, both structured and unstructured, and the sizes are between those of structured lattices and those of FrodoKEM. Also security of these schemes is less understood compared to lattice-based schemes: Code-based schemes using structured codes have a somewhat troubled history, e.g., systems using cyclic codes (rather than quasi-cyclic codes) were broken. The codes used in these schemes have not been subject to an attack but are also more recent and fewer cryptanalysis results are known than for the lattice-based schemes.

In our theoretical analysis we focus on Kyber, McEliece, and Frodo. The reason is that the performance of Saber, NTRU, and NTRU Prime is close enough to that of Kyber to not make a huge difference in a theoretical analysis (note that when evaluating practical performance, the slower key generation of NTRU and sntrup may make a difference) [2]. Similarly, BIKE and HQC are similar in performance to structured lattice-based schemes (although noticeably larger) [2]. Therefore results would be similar to those for Kyber and will not open up different trade-offs. Frodo demonstrates what it costs to go to unstructured lattices. SIKE is broken [15] and there is no similarly-small scheme in the NIST process.

An ever reoccurring topic when it comes to lattice-based KEMs is the topic of patents. We are no patent lawyers and cannot comment on this issue. NIST announced that it has come to agreements with holders of patents that potentially concern Kyber. According to NIST, the agreement allows the royalty free use of Kyber as standardized by NIST. We leave it to patent lawyers to discuss any further issues related to patents.

We depict the sizes of private keys (sk), public keys (pk) and ciphertexts (ct) for the three systems in the reference implementation in Table 1 and Figure 1, for comparison the former also includes the pre-quantum elliptic-curve scheme X25519. Note that private keys often can be compressed down to a seed at the expense of somewhat slower decapsulation times, e.g., the key format of Classic McEliece supports compressed private keys of size 32 bytes.

Table 1: Key- and Ciphertext-sizes, Estimated NIST security-level ("Sec") and probability of decapsulation-failure (δ) of various KEMs. X25519 is an elliptic curve based pre-quantum key-exchange whose characteristics are representative of those of elliptic-curve based KEMs, which are our recommendation for the fallback-scheme in hybrid KEMs. Were it not for quantum-attacks X25519 would likely be classified as a level-1 scheme.

Scheme	SK	PK	CT	Sec.	δ
X25519	32	32	32	-	0
Kyber-512	1632	800	768	1	2^{-139}
Kyber-768	2400	1184	1088	3	2^{-164}
Kyber-1024	3168	1568	1568	5	2^{-174}
mceliece348864	6492	261120	96	1	0
mceliece460896	13608	524160	156	3	0
mceliece6688128	13932	1044992	208	5	0
mceliece6960119	13948	1047319	194	5	0
mceliece8192128	14120	1357824	208	5	0
FrodoKEM-640	19888	9616	9720	1	$2^{-138.7}$
FrodoKEM-976	31296	15632	15744	3	$2^{-199.6}$
FrodoKEM-1344	43088	21520	21632	5	$2^{-252.5}$

5.2. Signatures

NIST PQC chose three signature-schemes and additionally standardized a stateful signature-scheme before that:

Dilithium: Dilithium [31] is NIST's primary choice for signatures. Is is based on structured lattices, using module lattices as Kyber. Public keys and signatures have about the same size and are both somewhat larger than their KEM counterparts. It provides decent performance and size-characteristics, without major downsides.

Falcon: NIST also recommends FALCON [39], another lattice-based signature scheme using structured lattices, for settings in which small signatures are important. While having the same overall characteristics as Dilithium, signatures in Falcon are significantly smaller, so much smaller that they are smaller than Kyber ciphertexts. The downside of the scheme is that its definition uses floating-point arithmetic which is not available on smaller platforms and which is notoriously difficult to implement securely.

SPHINCS+: SPHINCS+ [25] is the third recommendation of NIST. The system is based on hash functions. While it is extremely conservative in its assumptions, even more so than established pre-quantum schemes, and its public keys are small, its signatures are most likely too large to be of any use in environments with

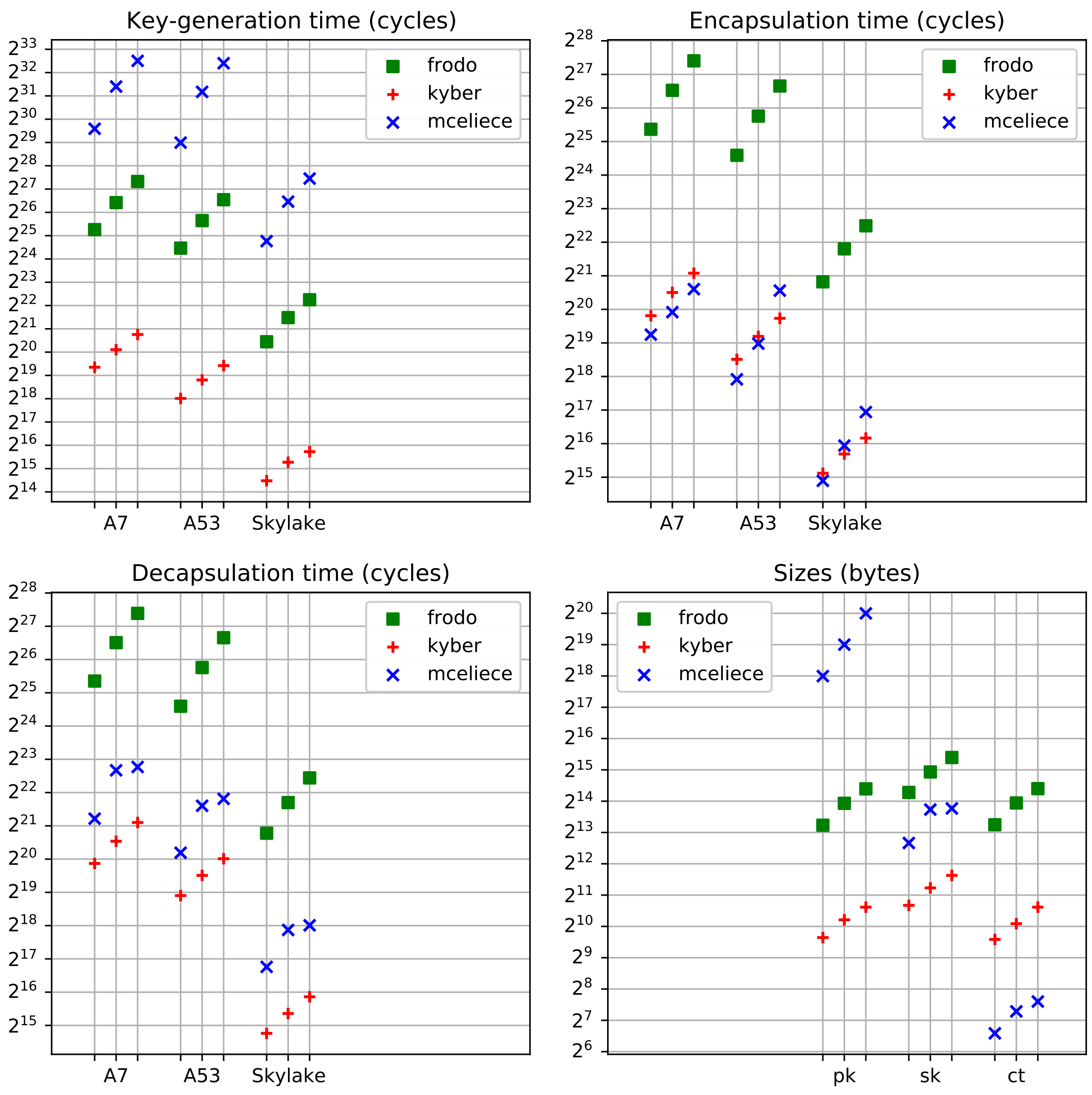

Figure 1: Size- and Performance-characteristics of Frodo, Kyber and McEliece at NIST Level 1, Level 3 and Level 5 on ARM Cortex-A7, ARM Cortex-A53 and Intel Skylake CPUs using benchmarking data from eBACS.

limited bandwidth, such as communication with satellites. Despite this limitation, SPHINCS+ is built from components that may be useful for this project.

Beyond these signature-schemes that NIST choose during the post-quantum competition and which follow the regular definition of signature schemes, there are also stateful hash-based signature-schemes that the IETF and NIST have standardized and that are in the process of being standardized by ISO. They provide possibly interesting trade-offs and are already under consideration with ESA for the purpose of signing software:

XMSS: RFC 8391 [26] standardizes XMSS: Like SPHINCS+ it is a hash-based signature-scheme with extremely conservative assumptions (SPHINCS+ can in fact be seen as an extended version of XMSS), but unlike SPHINCS+ it has to maintain a state when signing: Each signing-operation updates that state and reusing an old state destroys the security of the entire scheme. Furthermore the system fixes the number of signatures that can be made with one key, i.e., the state-update can only be performed a limited number of times (a typical limit would be 2^{20}). Signatures in XMSS are much smaller than in SPHINCS+, the exact length depends on the number of signatures the key can sign, and the performance is good.

LMS: RFC 8554 [34] standardizes Leighton-Micali Hash-Based Signatures (LMS), another stateful hash-based signature scheme. The scheme is very similar to XMSS, with slightly larger signatures, a reliance on random oracles in its security-proof and about four times better overall performance.

WOTS: At their core SPHINCS+, XMSS and LMS are all schemes that are designed around the idea of using Winternitz One-Time-Signatures (WOTS) in a way that allows to sign more than one message. WOTS are hash-based signatures that can only be used to sign one message and become insecure if they are used twice. They have very small public keys and depending on the exact parameterization also quite small signatures and fast verification. As such we consider them a viable signature-scheme too, as long as their main-downside of only being able to sign once does not cause issues.

Beyond these options, the following algorithms also made it into the third round of the NIST-competition:

Rainbow: Rainbow [22] is a multivariate scheme that became a NIST-finalist. It shares the typical performance characteristic of multivariate schemes in that it offers very short signatures and very large public keys. However, Beullens [11] showed that the design does not offer any advantages over the plain unbalanced oil-and-vinegar construction and broke the level-1 parameters.

GeMSS: GeMSS [14] is a multivariate scheme based on HFEv-. It has large public keys (between 0.3 and 3 megabytes) and small signatures. It was an alternate candidate in the NIST-competition and its security was degraded by [45].

Picnic: Picnic [46] is a scheme that like SPHINCS+ only relies on symmetric primitives. The schemes have similar properties and in the end Picnic simply failed to demonstrate a clear advantage over its slightly more conservative competitor, causing it to loose the NIST-competition in which it was an alternate candidate.

NIST has started a new competition for post-quantum signatures and submissions were due in June 2023. We do not consider these submissions in scope as they have not received sufficient security analysis beyond some that are already completely broken.

In summary Dilithium and Falcon are the two primary options for a signature-scheme if bandwidth is a concern. The choice between them is largely a trade-off between higher bandwidth-use and higher implementation-difficulty. In case state-based schemes are acceptable, XMSS and LMS are possible options too that pay for their larger signatures with massively smaller public keys and radically fewer assumptions. The most extreme option is then to use WOTS directly which only allows to sign a single message.

Beyond these options, SPHINCS+ and Picnic have to be considered non-viable due to their large signatures, and Rainbow and GeMSS do not reach the claimed security levels. For these reasons we do not consider these options further for the key-update mechanism.

We depict the sizes of the NIST-winners and the elsewhere standardized stateful signature schemes in Table 2.

Table 2: Estimated security-level, Key- and signature-sizes of various signature schemes. The security-level (”Sec“) is given as NIST-level, where levels 1/3/5 are roughly equivalent to the security of AES 128/192/256 against brute-force key search under quantum and classical attacks. Levels 2 and 4 are matched with collision-resistance of a hash function with 256 and 512 bits, respectively. The sizes of the WOTS+-keys are given assuming the use of key-compression. ECDSA is a pre-quantum scheme for cases where hybrid signatures are desirable; were it not for quantum-attacks it would likely be classified as a level-1 scheme.
(*) The stateful signature-schemes XMSS and LMS do not explicitly standardize the format of their secret-keys. The given sizes are assuming the use of pseudorandom key generation. Any efficient implementation will require an additional state with a size depending on the implementation and used parameters, typically in the order of kilobytes. Depending on the implementation, the whole state or most of it is non-secret information.

Scheme	SK	PK	Sig	Sec
Dilithium2	2544	1312	2420	1
Dilithium3	4016	1952	3293	3
Dilithium5	4880	2592	4595	5
Falcon-512	1281	897	666	1
Falcon-1024	2305	1793	1280	5
SPHINCS+-128s	64	32	7856	1
SPHINCS+-128f	64	32	17088	1
SPHINCS+-192s	96	48	16224	3
SPHINCS+-192f	96	48	35664	3

Scheme	SK	PK	Sig	Sec
SPHINCS+-256s	128	64	29792	5
SPHINCS+-256f	128	64	49856	5
XMSS-SHA2_10_256	*64	64	2500	5
XMSS-SHA2_16_256	*64	64	2692	5
XMSS-SHA2_20_256	*64	64	2820	5
LMS_SHA256_M32_H15 with LMOTS_SHA256_N32_W4	*52	56	2664	5
LMOTS_SHA256_N32_W1	*52	56	8516	5
LMOTS_SHA256_N32_W2	*52	56	4292	5
LMOTS_SHA256_N32_W4	*52	56	2180	5
LMOTS_SHA256_N32_W8	*52	56	1124	5
WOTS+(32,16)	32	32	2144	5
WOTS+(32,4)	32	32	4256	5
ECDSA-P256	32	32	64	-

5.3. Combining PQC with ECC

Requirement 3.1.4 asks to deploy the post-quantum algorithms in combination with “pre-quantum” public-key cryptography as also required by several European security agencies, including BSI and ANSSI. For the pre-quantum algorithms, elliptic curve cryptography (ECC) is preferable over RSA-based solutions due to their size. The common way to combine PQC and ECC algorithms is by the means of so called combiners. In the following we discuss combiners for KEM and signature. Another detailed discussion of the topic can be found in ENISA’s integration study on post-quantum cryptography [10].

5.4. KEM-Combiners

The goal of a KEM-combiner as considered in this work is to construct a KEM $\mathcal{S}$ from two KEMs $\mathcal{P}_1, \mathcal{P}_2$ such that $\mathcal{S}$ is secure as long as at least one out of $\mathcal{P}_1, \mathcal{P}_2$ is secure. For security we require security against active attacks, which is called ciphertext-INDistinguishability under Chosen Ciphertext Attacks (IND-CCA). Consequently, we describe our protocol with respect to a single KEM that can then be instantiated with any secure KEM, including a KEM that is obtained via any secure KEM combiner.

It turns out that the design of a robust KEM combiner is not as straightforward as one may think. There are several pitfalls possible as observed by Giacon, Heuer, and Poettering [23]. However, there also exist several efficient and secure solutions. The combiners commonly run the two schemes $\mathcal{P}_1, \mathcal{P}_2$ independently and only combine their data objects afterwards for derivation of the shared secret. The most simple and still secure construction takes the shared secrets k_1, k_2 and the ciphertexts c_1, c_2 of $\mathcal{P}_1$, and

$\mathcal{P}_2$, respectively, and computes the final shared key as

$$k = \mathcal{H}((k_1 \| k_2), (c_1 \| c_2)),$$

where $\mathcal{H}$ is a key-derivation function.

This construction can be proven secure modeling $\mathcal{H}$ as a random oracle, a common heuristic used when proving practical cryptographic schemes secure. Several of the CCSDS-recommended cryptographic algorithms are proven secure in the random-oracle model. Hence, this does not add any new assumption to the full protocol. In practice, $\mathcal{H}$ can be implemented preferably with HMAC-SHA2, or KMAC-SHA3.

A variant of this combiner is currently discussed in IETF's CFRG for standardization [37] and is also used in a proposed Internet-Draft for Post-Quantum OpenPGP [29]. Moreover, variants of this design are discussed in CFRG that fix the combined KEMs (c.f., X-Wing [7]). This has the advantage that the combiner can be optimized for the choices made, but of course comes at the cost of generality.

To highlight this once more: The exact choice of combiner does not matter for the protocol proposed here, as long as the resulting KEM achieves IND-CCA security given that at least one the combined KEMs does the same.

For the purposes of this report we will assume the use of the generic combiner [37] with hashed Diffie-Hellman based on X25519 as pre-quantum KEM as it works with all considered schemes. Other curves could of course be used instead, which may or may not change size and/or performance slightly, depending on the replacement.

5.5. Signature-Combiners

While designing a secure KEM-combiner is somewhat non-trivial, the opposite is the case for signatures. As long as the standard security notion of Existential Unforgeability under Chosen Message Attacks (EUF-CMA) is the goal, the trivial combiner – double signature – works. For that the message is simply signed with both signature schemes that are used and then the resulting signatures are concatenated for the combined signature. Verification works by splitting the signatures again into two and verifying both of them. If any verification fails the combined signature must be rejected, otherwise the signature is valid.

We remark that this way of combining signatures does not achieve Strong Existential Unforgeability (SEUF-CMA), which states that an adversary should also be unable to create a different signature for a message for which they already received a valid signature. This is because if one of the signature schemes is broken an attacker may be able to provide a different signature on the message under that scheme while keeping the signature part under the secure scheme, thus providing a new combined signature on the same message. That said, unlike EUF-CMA, SEUF-CMA is not a commonly required property and in particular not needed for any of our proposed protocols.

There do exist more involved security notions for signature combiners (c.f., [13]). These are motivated by the use in systems that have to provide legacy options, to achieve backwards compatibility (e.g., can one signature be verified without verifying the other). In the given setting, there is no system for which we have to provide such options and therefore, these notions are not of relevance here.

In places where a pre-quantum signature is necessary we will assume the use of ECDSA [5], in particular with the NIST P-256 curve. ECDSA is listed as acceptable in the CCSDS-document on Cryptographic Algorithms and outperforms the alternatively listed RSA-signatures on almost every metric. As with the KEM-fallback this is largely a representative choice and could be replaced with any other signature-scheme (for example EdDSA [9]).

6. Proposals

In this section we provide outlines for possible protocols with different trade-offs. In general, our proposals can be seen as the main blueprints that can be used (and have been used) to build authenticated key-exchange using KEMs. Thereby they are summarizing the core of all previous proposals for post-quantum secure communication using KEMs and signatures. The difference between the existing previous proposals is mostly motivated by considerations regarding backwards-compatibility or non-cryptographic, protocol-specific considerations.

As a baseline, we consider the traditional approach of using signature key pairs as long-term keys and KEM (traditionally public key encryption) key pairs as ephemeral keys. This approach was for example used in TLS up to version 1.2. Most early proposals for post-quantum secure communication focused on TLS and followed this general approach (see for example [42, 19, 38]). Next, we discuss different variations of a more recent approach proposed for PQC which only uses KEM key pairs, motivated by the difference in artifact sizes between PQ-KEM and -signatures. This approach was used by the PQ protocols PQWireGuard [27], KEM-TLS [44] and PQNoise [4]. Finally, we discuss some more involved variations of the KEM+Signature approach.

6.1. KEM+Signature Exchanges

The traditional approach revolves around using an ephemeral KEM key pair to encapsulate the session key and sign the ephemeral public key with a long-term signing key.

This is for example the approach used by TLS 1.3 when Diffie-Hellman key exchange is used. This was already used for RSA key transport methods in previous versions of TLS. Given that RSA encryption in this case is used like a KEM, a post-quantum KEM can be used as drop-in replacement.

This kind of exchange can be described by the following noise-pattern:

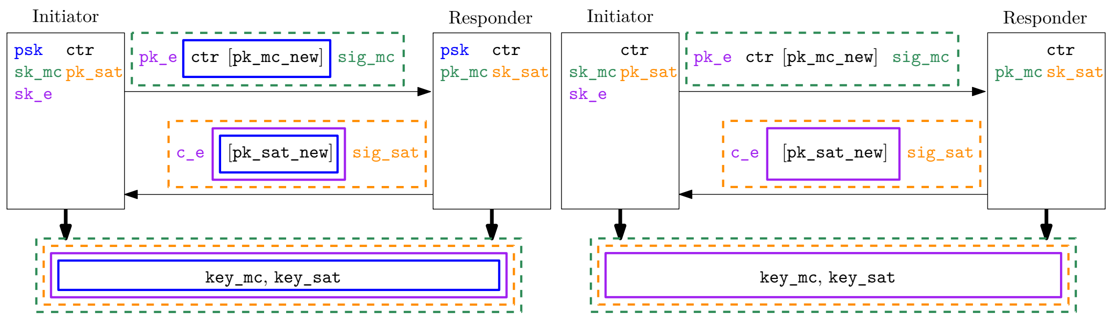

Figure 2: Overview of a KEM+Signature exchange. Solid boxes indicate that confidentiality and in case of the blue boxes authenticity of their content are protected by the same-color secret. Dashed boxes indicate that only the authenticity, but not the confidentiality is protected. Values in square-brackets are optionally updated longterm keys and may be skipped. The left diagram depicts the situation in which there is an uncompromised pre-shared key (psk) and the right one the situation where there is not.

```
-> s
<- s
...
-> psk, e, s'[opt1], sig
<- ekem, s'[opt2], sig
-> confirm
```

We provide detailed code of a variant of this proposal in Appendix A. Detailed code for this proposal would differ from the code shown there in that the AEAD decryption calls require error handling to deal with invalid messages. KEMs are expected to use implicit rejection and thus always output a key, correctness is then established by using the key in AEAD encryption or decryption.

We note that there may not always be a psk that can be used to derive security from it. In those cases that key should be set to an all-zero bitstring of appropriate length. It will no longer add any security in those cases, but since its inclusion in the hand-shake is a pure defense-in-depth-measure, this does not significantly endanger the security of the overall protocol. We include the psk because it is computationally cheap to do so and it provides a fairly robust security-fallback, not because our analysis relies on it. See Figure 2 for a diagram that depicts the difference in protection of the transmitted data depending on whether the used psk is secure.

This proposal is the baseline. It follows a well vetted design for which numerous implementations already exist (without the public key update). Any other proposal would have to beat this one to be considered.

The modification for a partially authenticated version, which derives the authenticity of the responder from outside facts, would differ from this version in that the responder's packet would only include the ciphertext for the ephemeral KEM (`ekem`) and drop the signature and the updated long-term-key.

6.2. Dual/Triple-KEM Exchanges

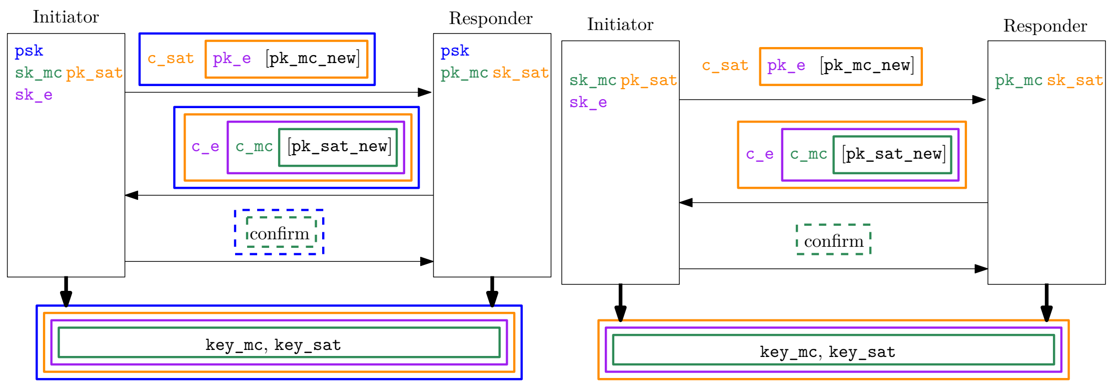

Figure 3: Overview of the Triple-KEM exchange. Solid boxes indicate that the confidentiality of their content is protected by the same-color secret. Solid blue boxes and boxes in the color of the senders's longterm secret furthermore guarantee authenticity at the time at which they are received. Dashed boxes indicate that only the authenticity (when received), but not the confidentiality is protected. By the time the protocol completes (that is when it outputs `key_mc` and `key_sat`) the entire transcript is authenticated. Values in square-brackets are optionally updated longterm keys and may be skipped. The left diagram depicts the situation if there is an uncompromised pre-shared key (psk) and the right one the situation where there is not.

The more recent approach to authentication in a post-quantum setting is the use of a long-term KEM-key to effectively run a challenge-response protocol. If Alice wants to authenticate herself to Bob, she has Bob encapsulate a shared secret for her long-term key and send her the ciphertext. If she is then able to create an AEAD-ciphertext using the shared secret as key, Bob can conclude upon successful decryption that he is talking to Alice. The encapsulated shared secret was only accessible to himself and the owner of the private key (Alice). Since it is (by assumption) not practically possible to create an AEAD-ciphertext for a given key without knowing said key, only Alice or Bob could have created that ciphertext.

Existing examples for PQ protocols that follow this approach are PQWireGuard [27], KEM-TLS [44] and PQNoise [4]. The following proposal is close to the pqKK-pattern of PQNoise up to the optional inclusion of new long-term public keys.

```
-> s
```

```
<- s
...
-> psk, skem, e, s' [opt1]
<- ekem, skem, s'[opt2]
-> confirm
```

This approach has one main advantage over the more traditional signature-based authentication: Post-quantum KEMs tend to be more efficient than signature-schemes, especially in terms of artifact size. Hence, trading signatures for KEMs leads to protocols with lower communication cost. It comes as a nice addition that this approach is secure against replay-attacks by construction.

Just as with the Sign+KEM-approach there may not always be a previous shared key (psk) that can be used to derive security from it. Our response to that problem is the same as with Sign+KEM as well: The psk should in those cases be initialized with an all-zero bitstring, as it is merely used as a defense-in-depth-measure. See Figure 3 for a diagram that depicts the difference in protection of the transmitted data depending on whether the used psk is secure.

If only the authenticity of the initiator has to be ensured because the authenticity of the responder can be guaranteed out-of-band, Triple-KEM can be simplified into **Dual-KEM**: Dual-KEM differs from Triple-KEM only in that the initiator would not send a ciphertext for the responder's long-term-key and that the responder's answer would drop the updated long-term-key. The benefit of this approach would lie in a more compact first packet and less computation because a third of the asymmetric operations could be skipped. The disadvantage is that Dual-KEM does not itself guarantee any authenticity of the responder. Sections 9 and 10 provide detailed descriptions of Triple-KEM and Dual-KEM.

6.3. Variants

Below we discuss two variations of the Sign+KEM-approach using stateful (hash-based) signatures. First, we discuss what happens if one simply replaces the signature scheme with a stateful one. Then we discuss how this approach can be optimized using one-time signature chains.

6.3.1. Stateful Signatures

The above two proposals work for any standard KEM and signature scheme. We remark that if the number of uses of a long-term key is limited by policy, stateful signatures like XMSS or LMS could potentially be used. The big downside of stateful signatures is that their secret key changes over time. If an old key state is reused they become insecure. This can happen for example as a result of using the secret key of an outdated backup. While this forms an unacceptable hazard for systems that face end-users, keeping a state may be a manageable task in an expert-controlled environment like a mission-control center or a satellite similar to how stateful hash-based signatures are used in code signing.

Furthermore using LMS or XMSS as (stateful) signature-schemes has also advantages:

- Both have great performance- and size-characteristics.
- Unlike the NIST-winners they are already standardized.
- The schemes do not introduce any new security assumptions but rely on the security of hash functions – a necessity for any signature scheme that can handle arbitrary length messages.
- For that reason, ANSSI states that there is no need for a hybrid solution when using hash-based signatures.

The resulting protocol is identical to the KEM+Signature-proposal, except that the secret keys for the signature-scheme change after each signature and that occasional updates of the long-term keys are no longer optional.

6.3.2. Signature Chains

If a satellite only communicates with one mission-control center (or multiple, but in such a way that the interactions are completely independent of each other), it is possible to go even further and move to signature-chains. This idea was proposed in the context of sensor networks where computational resources are sparse. In the vanilla version of this idea, each key pair is used to authenticate one message, together with the public key of the next key pair. In this case, a so called one-time signature scheme like WOTS can be used, which forms the main ingredient of stateful hash-based signature schemes, including XMSS and LMS.

As a noise-pattern this would look as follows:

```
-> s
<- s
...
-> psk, e, s', sig
<- ekem, s', sig
```

This solution can be an attractive solution due to the reduced computational cost and the comparatively small signature size. On the down-side, this involves more complicated key management as the “long-term” key pair of a party has to be updated with every key exchange.

Care has to be taken to handle exceptional cases like if a message got lost, or if either side got interrupted during the exchange. On both sides the countermeasure is the same. All randomness required by a party to run the protocol is sampled as first step of the protocol and stored in non-volatile memory. Only then do the parties run the protocol, using the stored randomness. This way the protocol becomes deterministic and an aborted protocol run can be reproduced with exactly the same messages. As long as parties only update the locally stored key pair in storage when the protocol successfully terminated, de-synchronization can be avoided.

6.4. Generic Comparison

As outlined in Section 7.1, the Triple-KEM approach has the potential advantage, that it allows to mix and match different KEMs to protect different properties using different trade-offs. If long-term-keys are for example never updated, it would be possible to instantiate them with Classic McEliece to benefit from the small ciphertexts, but use Kyber for ephemeral keys where the public key has to be transmitted on every exchange. This would improve the performance, the bandwidth-requirements and potentially even the security.

An advantage of the signature-based approach on the other hand is that it technically allows to design a protocol that saves half a round-trip because it does not strictly require a key-confirmation message. Assuming that a sufficiently secure replay-protection is in place, the first message in the protocol can be assumed to be authentic by the satellite, because it is by assumption not possible for an attacker to forge the included signature and the replay-protection prevents replays. Similarly the response could be assumed to be authentic by the receiver for the same reasons.

The problem with this approach is that it is very fragile and minor issues that would be fine with a key-confirmation message may cause persistent denial-of-service attacks:

Any weakness in the replay-protection-system would not just allow for the initiation of a handshake via replaying the previous message, but would in fact complete the protocol, possibly replacing a critical communication-key on the satellite with one that is unknowable to mission-control if mission-control already discarded its ephemeral keys. Replay-protection systems are however not trivial to get right, typical approaches include counters which requires maintaining state or time-based approaches which require synchronized clocks *and* a very tight control of the acceptance-window of a message: If the window is open for even slightly too long, replays become possible for attackers who have very low-latency access to previously sent messages. A window that is slightly too short will prevent that attack, but may also prevent the honest user from sending a message within that window. In the context of satellites that may change their distance from at least the ground-stations very quickly and may in the extreme orbit bodies other than earth, an exact calculation of that window, while certainly possible, is sufficiently non-trivial to make its inclusion in a cryptographic module questionable.

Even if there are no problems with the replay-protection-system, a further potential issue could occur if the satellite's response is lost for whatever reason. In this case the satellite would perform the requested key-update, but the ciphertext needed by mission-control to compute the shared secret would never arrive, causing functionally the same problem described in the previous case. We remark that while transport-layer protocols could mitigate this issue to an extend, they would functionally turn the approach into a more-than-one round-trip protocol.

For these reasons we can already state here, that we do not recommend versions of the signature-based approaches that skip key-confirmation.

7. Alternative approaches not explored

Here we discuss concepts we briefly considered and decided against. The first one is a different approach towards combining PQC and ECC. The second proposal is an alternative way to achieve forward-secrecy.

7.1. Hybrid scheme

The common way to combine PQC and ECC is via the use of primitive combiners, This approach has the advantage that it massively simplifies the analysis of the greater protocol, because the protocol-analysis will generally not take the concrete instantiation of a primitive into consideration, but merely that the primitive is secure. The disadvantage is that sizes and computation of the combined schemes add up and that some extra calls to a key-derivation function are needed.

Alternatively it is also possible to use different instantiations of the same primitive in the actual protocol. For example, we can instantiate the long-term KEM key pair in the triple-KEM proposal with Classic McEliece, and the ephemeral key pair with Kyber. This approach was introduced by PQWireGuard as a means to optimize package size (though there a variant of Saber is used for the ephemeral key instead of Kyber and the scheme recommends to use an ECC-fallback regardless). This exploits that McEliece has extremely small ciphertexts and if the public key never has to be sent in the protocol, its size does not matter too much.

This approach achieves full hybrid security in the absence of key compromise. However, as soon as a key (long-term or short-term) is compromised, security is fully based on the security of the scheme with the non-compromised key. Moreover, authentication becomes non-hybrid in this case. Because of these issues we did not further consider this option as a full replacement of the regular hybrid approach. We do note however that it can still act as a defense-in-depth-measure and may be useful for different reasons (such as performance- or bandwidth-advantages).

Table 3: Sizes for a Triple-KEM Kyber-McEliece Hybrid in bytes, without and with long-term key update. Here McEliece is used for long-term keys, Kyber for ephemeral keys

Scheme	Packet 1	Packet 2	Packet 3
TK(Kyber512,McEliece348864)	928	896	16
TKU(Kyber512,McEliece348864)	262048	262032	16

7.2. Forward Secrecy through Symmetric Ratchet

Forward Secrecy states that communication has to remain confidential even if an involved party becomes compromised at a later point in time. The traditional way to achieve this is through the use of ephemeral key-exchanges as part of a handshake – a later compromise of the long-term secrets does not allow to break confidentiality of previous communication once the ephemeral secrets of that communication are deleted.

We note that this is not the only way to achieve this property though: It is also possible to use a traditional "symmetric ratchet". The core-idea of that approach is that instead of updating the shared secret via a new key-exchange, new keys can essentially be derived from old keys via a one-way function. This way all involved parties can perform key-updates on their own without having to interact with their peers, as long as everyone agrees on the update-policy. This approach is remarkably efficient and is often worth doing, but while it protects against future compromise it cannot recover from past compromise.

The approach made it into our final version in so far that we mix the previous key into the hash-object as a defense-in-depth-measure, but the inability to create new and independent key-material meant that we did not rely on it as a primary defense.

8. Evaluation

The packet-sizes for the various possible protocols are depicted in Table 4.

Table 4: Sizes of different instantiations in bytes. SK = Sign + KEM, SKU = Sign+KEM with update of longterm-key, SC = Signature-Chain, TK = Triple-KEM, TKU = Triple-KEM with update of longterm-key.

Scheme	Packet 1	Packet 2	Packet 3	Sec
SK(Kyber512+X25519+Dilithium+ECDSA)	3348	3300	16	1
SKU(Kyber512+X25519+Dilithium+ECDSA)	4692	4644	16	1
SK(Kyber512+X25519+Falcon+ECDSA)	1594	1546	16	1
SKU(Kyber512+X25519+Falcon+ECDSA)	2523	2475	16	1

Scheme	Packet 1	Packet 2	Packet 3	Sec
SK(Kyber512+X25519+XMSS-SHA2_10_256)	3364	3316	16	1
SKU(Kyber512+X25519+XMSS-SHA2_10_256)	3428	3380	16	1
SC(Kyber512+X25519,WOTS+(32,16))	3024	2992	16	1
SC(Kyber768+X25519,WOTS+(32,16))	2408	3312	16	3
SC(Kyber1024+X25519,WOTS+(32,16))	3792	3792	16	5
SC(Kyber1024+X25519,WOTS+(64,16))	10032	10032	16	5
TK(Kyber512+X25519)	1664	1632	16	1
TKU(Kyber512+X25519)	2496	2480	16	1
TK(Kyber768+X25519)	2368	2272	16	3
TKU(Kyber768+X25519)	3584	3504	16	3
TK(Kyber1024+X25519)	3232	3232	16	5
TKU(Kyber1024+X25519)	4832	4848	16	5
TK(Kyber512,McEliece348864, both + X25519)	992	960	16	1
TKU(Kyber512,McEliece348864, both +X25519)	262144	262128	16	1

The hybrid-versions assume that a KEM-combiner is used with a KEM-version of X25519, but without turning WOTS+ into a hybrid. These numbers would change slightly depending on whether only EKEM should be hybrid or whether the signatures should also be made hybrid. That said, the comparison clearly demonstrates that the use of a hybrid elliptic curve scheme comes at little cost with regards to packet-size compared to using only the post-quantum scheme. We do not consider instantiating ephemeral KEMs with Classic McEliece due to the large size of the public key.

All of these sizes assume the use of an AEAD-scheme that adds a constant 16 bytes to the length of the plaintext; this assumption holds for many widely used schemes, in particular AES-GCM with a full-sized authentication-tag. The sizes are however independent of the choices made for the involved hash-functions (except for the case of hash-based signatures), as the protocol does not transmit any hashes.

We note that all of these proposals offer forward-secrecy, post-compromise security, updateable long-term keys and some form of hybrid security. This also holds if the network fragments packets, but in the event that fragmentation causes data to arrive out-of order or not at all, the protocol as presented would not be able to reorder or re-request them, and instead assume an attack and drop the connection. The layer handling fragmentation would be required to ensure the correct ordering and re-requesting of lost packets if dropped connections are an issue.

8.1. Conclusion

Our primary recommendation is Triple-KEM(Kyber+X25519):

- Instead of two PQC and ECC schemes in case of Sign+KEM, only one scheme each has to be implemented, reducing the chance of mistakes and the attack surface.

- Packets are generally smaller than for Sign+KEM, except for Kyber+Falcon without key update. However, Falcon amplifies the above issue as it is known to be notoriously hard to implement securely due to the use of floating point arithmetic.
- In terms of speed, we expect Triple-KEM(Kyber+X25519) to clearly outperform Sign+KEM(Kyber512+X25519+Falcon+ECDSA).

This comes with the variant Dual-KEM(Kyber+X25519):

- Besides giving up responder-authenticity on the protocol-level, the variants are very similar.
- The bandwidth-savings equate to the size one Kyber-ciphertext in all interactions; additionally not updating the long-term-key saves the size of Kyber-public-key.
- Dual-KEM can clearly be expected to have better performance than Triple-KEM, both for computation time and (more important) bandwidth.

Our second choice is still Sign+KEM(Kyber512+X25519+Falcon+ECDSA).

- The sizes are best in class as long as no key update is done and even then they are at a close second place.
- The scheme still outperforms the hash-based proposals even though it adds ECC.
- It avoids the necessity to maintain a state.
- However great care needs to be taken to implement Falcon securely.

The proposals using XMSS / LMS, or one-time signature chains may be of interest in case that a mission already has an implementation of the primitive available on the device. In this case, savings in code-size could be achieved and state-handling is likely already supported.

9. Triple-KEM in Detail

The following listing provides a detailed description of Triple-KEM. It uses the following primitives as building-block:

- `NoiseHashObject`, a construction that Noise-style protocols use and that was first formalized as its own primitive in PQNoise [4]. For a complete description see Section 11.2; intuitively this primitive consists of a state (here: `prho_state`) that can be used with the two functions `input` and `finalize`: Input takes a state and a bitstring as arguments and returns an updated state and a freely choosable number of outputs. If the bitstring that has to be provided as argument is pseudo-random, than all subsequent outputs are pseudo-random as well. `finalize` works in the same way, except that it does not output an updated state.

  For simplicity we use class-style notation with the hash-object, that is we write `out = state.input(in)` instead of `state, out = input(state, in)`.

- `H`, a collision-resistant hash-function, for example SHA3. Functionally we use this to implement a hash-object that uniquely hashes the network-transcript up to a certain point. Multiple values are always added seperately (H(H(state, x), y) instead of H(state, x, y)) to ensure domain-separation.

- The value PROTOCOL_TOKEN is a token unique to the protocol; it can prevent protocol confusion and provides a degree of domain-separation. This parameter is public and its concrete value freely choosable. Possible values would for example be the ASCII-encoding of "SDLSP-KeyReplacment-3KEM-3Kyber-Version1", UUIDs or values unique to the network that the satellite is supposed to participate in.
- IKEM, RKEM The long-term KEMs used by the initiating party (usually: mission-control) and the responding party (usually: the satellite). In our recommendation both of these are set to Kyber+X25519.
- EKEM The ephemeral KEM, in our recommendation Kyber+X25519
- AEAD An **A**uthenticated (symmetric) **E**ncryption-scheme with support for **A**ssociated **D**ata, for example AES-GCM. The inputs to AEAD.enc are the key, the plaintext, a nonce, and the associated data; the output is the ciphertext. The inputs to AEAD.dec are the key, the ciphertext, the nonce, and the associated data; the output is the plaintext or failure (if authentication fails).
- KDF a Key-Derivation Function, for example HMAC-SHA2

For KEM KEM, KEM.enc produces the ciphertext and the shared key using the receiver's public key; KEM.dec produces the shared key from the ciphertext using the receiver's secret key.

The following protocol does not include statements for error handling. If the AEAD returns failure this error must be caught and handled appropriately. Benign reasons may be decryption failures in the KEMs leading to mismatching secret keys or communication failures introducing errors. The latter should be caught by error-correcting codes.

The final output of the protocol consists of the two keys key_mc and key_sat that can then be used to encrypt data symmetrically (primarily via SDLSP). We output two keys in order to enable assigning one key as exclusive sending-key to each involved party as a robustness measure (in particular there is less danger of nonce-reuse if every party has its own sending-key), but note that both keys are equally secure and that only using one of them is possible.

Additionally h_6 forms a computationally unique (based on the collision-resistance of H) handshake-hash that can safely be used as a session-identifier to uniquely identify a session. The final algorithms on both sides send_3 and receive_3 can be altered to additionally output the value of h_6 without any security impact if such a session identifier is required on a higher protocol level.

(The following notation largely aligns with Python; "+" when used with respect to byte-sequences means concatenation.)

```python
def send_1(update_longterm: bool):
    prho_state_mc = NoiseHashObject.create()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN)
    k_0 = prho_state_mc.input(psk)
    c_sat, k_sat = RKEM.enc(pk_sat)
    c_0 = AEAD.enc(k_0, c_sat, 0, h_0)
    h_1 = H(h_0, c_0)
    k_1 = prho_state_mc.input(k_sat)
    pk_e, sk_e = EKEM.gen()
    payload = pk_e
    if update_longterm:
        pk_mc_new, sk_mc_new = IKEM.gen()
        payload += pk_mc_new
    c_1 = AEAD.enc(k_1, payload, 0, h_1)
    h_2 = H(h_1, c_1)
    send(c_0, c_1)

def receive_1((c_0, c_1)):
    prho_state_sat = NoiseHashObject.create()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN)
    k_0 = prho_state_sat.input(psk)
    c_sat = AEAD.dec(k_0, c_0, 0, h_0)
    k_sat  = RKEM.dec(sk_sat, c_sat)
    h_1 = H(h_0, c_0)
    k_1 = prho_state_sat.input(k_sat)
    pk_e, pk_mc_new = AEAD.dec(k_1, c_1, 0, h_1)
    h_2 = H(h_1, c_1)

def send_2(update_longterm: bool):
    c_e, k_e = EKEM.enc(pk_e)
    c_2 = AEAD.enc(k_1, c_e, 1, h_2)
    h_3 = H(h_2, c_2)
    k_2 = prho_state_sat.input(k_e)
    c_mc, k_mc = IKEM.enc(pk_mc)
    c_3 = AEAD.enc(k_2, c_mc, 0, h_3)
    h_4 = H(h_3, c_3)
    k_3 = prho_state_sat.input(k_mc)
    if update_longterm:
        # This does not enforce knowledge of the new key,
        # but that should not be a problem:
        pk_sat_new, sk_sat_new = RKEM.gen()
        c_4 = AEAD.enc(k_3, pk_sat_new, 0, h_4)
        h_5 = H(h_4, c_4)
        send(c_2, c_3, c_4)
    else:
        h_5 = h_4
        send(c_2, c_3)

def receive_2((c_2, c_3, c_4)):
    c_e = AEAD.dec(k_1, c_2, 1, h_2)
    k_e = EKEM.dec(sk_e,c_e)
    h_3 = H(h_2, c_2)
    k_2 = prho_state_mc.input(k_e)
    c_mc = AEAD.dec(k_2, c_3, 0, h_3)
    k_mc = IKEM.dec(sk_mc, c_mc)
    h_4 = H(h_3, c_3)
    k_3 = prho_state_mc.input(k_mc)
    if c_4 != "":
        pk_sat_new = AEAD.dec(k_3, c_4, 0, h_4)
        h_5 = H(h_4, c_4)
    else:
        h_5 = h_4

def send_3():
    c_5 = AEAD.enc(k_3, "", 1, h_5)
    send(c_5)
    if pk_sat_new:
        pk_sat = pk_sat_new
        pk_sat_new = None
    if sk_mc_new:
        sk_mc = sk_mc_new
        sk_mc_new = None
    h_6 = H(h_5, c_5)
    pre_key_mc,pre_key_sat = prho_state_mc.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, key_sat

def receive_3(c_5):
    m = AEAD.dec(k_3, c_5, 1, h_5)
    if m != "":
        error()
    if sk_sat_new:
        sk_sat = sk_sat_new
        sk_sat_new = None
    if pk_mc_new:
        pk_mc = pk_mc_new
        pk_mc_new = None
    h_6 = H(h_5, c_5)
    pre_key_mc,pre_key_sat = prho_state_sat.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, key_sat
```

10. Dual-KEM in Detail

The easiest way to turn the above triple-KEM-protocol into a dual-KEM-protocol would be to instantiate the satellite-KEM with a Null-KEM, that uses the empty string as secret-key, public-key, ciphertext and shared secret and just returns those values from gen, enc and dec. As an added bonus this even preserves the validity of all proofs that do not rely on any properties of the satellite-KEM, such as the below proofs for initiator-authenticity and confidentiality.

The downside of this approach is however that it results in a protocol that does slightly more work and sends slightly more data than necessary. For this reason we provide a dedicated Dual-KEM, that has everything related to the satellite-KEM fully stripped out.

The following listing provides a detailed description of Dual-KEM. It uses the same building-blocks as before and closely follows the design of Triple-KEM. It skips the use of some indices in the variables (there is for example no `h_1`) to stay consistent with the naming in Triple-KEM.

```python
def send_1(update_longterm: bool):
    prho_state_mc = NoiseHashObject.create()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN)
    k_0 = prho_state_mc.input(psk)
    pk_e, sk_e = EKEM.gen()
    payload = pk_e
    if update_longterm:
        pk_mc_new, sk_mc_new = IKEM.gen()
        payload += pk_mc_new
    c_1 = AEAD.enc(k_0, payload, 0, h_0)
    h_2 = H(h_0, c_1)
    send(c_1)

def receive_1(c_1):
    prho_state_sat = NoiseHashObject.create()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN)
    k_0 = prho_state_sat.input(psk)
    pk_e, pk_mc_new = AEAD.dec(k_0, c_1, 0, h_0)
    h_2 = H(h_0, c_1)

def send_2():
    c_e, k_e = EKEM.enc(pk_e)
    c_2 = AEAD.enc(k_0, c_e, 1, h_2)
    h_3 = H(h_2, c_2)
    k_2 = prho_state_sat.input(k_e)
    c_mc, k_mc = IKEM.enc(pk_mc)
    c_3 = AEAD.enc(k_2, c_mc, 0, h_3)
    h_5 = H(h_3, c_3)
    k_3 = prho_state_sat.input(k_mc)
    send(c_2, c_3)

def receive_2((c_2, c_3)):
    c_e = AEAD.dec(k_0, c_2, 1, h_2)
    k_e = EKEM.dec(sk_e,c_e)
    h_3 = H(h_2, c_2)
    k_2 = prho_state_mc.input(k_e)
    c_mc = AEAD.dec(k_2, c_3, 0, h_3)
    k_mc = IKEM.dec(sk_mc, c_mc)
    h_5 = H(h_3, c_3)
    k_3 = prho_state_mc.input(k_mc)

def send_3():
    c_5 = AEAD.enc(k_3, "", 1, h_5)
    send(c_5)
    if sk_mc_new:
        sk_mc = sk_mc_new
        sk_mc_new = None
    h_6 = H(h_5, c_5)
    pre_key_mc,pre_key_sat = prho_state_mc.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, key_sat

def receive_3(c_5):
    m = AEAD.dec(k_3, c_5, 1, h_5)
    if m != "":
        error()
    if pk_mc_new:
        pk_mc = pk_mc_new
        pk_mc_new = None
    h_6 = H(h_5, c_5)
    pre_key_mc,pre_key_sat = prho_state_sat.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, key_sat
```

11. Analysis

The Triple-KEM-protocol provides the following properties:

1. **Confidentiality**: Honestly generated keys remain indistinguishable from randomness (=confidential) if the ephemeral randomness used during their creation remains confidential.
2. **Authenticity**: A party cannot be impersonated, as long as its long-term public key and the peer's ephemeral randomness remain uncompromised.
3. Honestly generated keys remain confidential if the pre-shared key remains uncompromised.
4. Honestly generated keys remain confidential as long as one party's long-term key and the peer's ephemeral randomness remain uncompromised.
5. As long as a connection remains confidential (see above), no passive attacker can learn more about a new long-term public-key than can be extracted from ciphertexts for that public key.

We will prove properties 1 and 2, as they cover the requirements. Properties 3 and 4 fall into the category of defense-in-depth where they are useful to mitigate some attacks that may result from cryptanalytic progress or issues with the implementation, but should not be relied upon on their own and are not required to meet the requirements. The same holds for Property 5, which given the lack of a need for privacy is not itself relevant here, but may mitigate some attacks by denying the adversary knowledge about targeted public-keys.

The Dual-KEM-protocol provides most of the same properties, except for the authenticity of the responder:

1. **Confidentiality**: Honestly generated keys remain indistinguishable from randomness (=confidential) if the ephemeral randomness used during their creation remains confidential.
2. **Authenticity**: The initiator cannot be impersonated, as long as its long-term public key and the responder's ephemeral randomness remain uncompromised.
3. Honestly generated keys remain confidential if the pre-shared key remains uncompromised.
4. Honestly generated keys remain confidential as long as one party's long-term key and the peer's ephemeral randomness remain uncompromised.
5. As long as a connection remains confidential (see above), no passive attacker can learn more about a new long-term public-key than can be extracted from ciphertexts for that public key.

Considering the similarity between the protocols, the proofs for the Triple-KEM are, with the exception of the responder-authenticity, fully applicable. Given the lack of built-in responder-authenticity, Dual-KEM only achieves our targeted security-notion if that authenticity is provided from other sources. Considering that the motivation behind not requiring that built-in authenticity is that impersonating an orbiting satellite on a physically narrow channel is assumed to be infeasible, we will make that assumption for the proof.

We note that additionally (although out of scope for this study), the use of a pre-shared secret allows to prevent a denial of service attack based on the cryptographic protocol. More specifically, the use of the pre-shared secret allows the responder, i.e., the satellite, to reject connection requests based on the first message and thereby prevents an attacker from filling the memory with invalid partial sessions.

11.1. Model

We will use a modified version of the eCK-model [30]. At its core the eCK-model (both the original and our version) works in a setting where n_p parties can run up to n_s key-exchanges (or „sessions“) each between them to compute shared secrets. The adversary is in full control of the network and can tell parties when to send or receive network-packets. Eventually the adversary gets to choose a target-session for which it receives a candidate for the shared secret, that depending on the challenge-bit is either the actual shared secret or a random and independent value of the same distribution.

If the adversary manages to guess the challenge-bit correctly while observing certain freshness-conditions, it wins. A protocol is considered secure if there is no Quantum Polynomial Time (QPT)- attacker whose win-probability is non-negligibly higher than 0.5.

Our biggest modification to this model is that we do not provide an oracle to reveal the ephemeral key to outlaw some classes of attacks that are out-of-scope. While this results in a strictly weaker security-notion, we still benefit from the familiarity of eCK and note that ephemeral key corruptions in the real world are almost exclusively caused by bad random-number-generators and that the best way to prevent them is to harden the random-number-generator.

This simplifies the freshness-predicates, because several failure-conditions depend on the adversary querying ephemeral keys. Specifically a session sid is now considered fresh, unless:

- The adversary queried to reveal the session key of sid or of its matching session (if it exists).
- No matching session to sid exists and the adversary queried the long-term key of the peer before the completion of the session.

The second, much more minor modification is that our version generates an initiator- and a responder-key per session pair, rather than a single secret. As acquiring either of them is considered a break, this is largely a technicality.

We will henceforth refer to this notion as “eCK-NEC”, with NEC being short for No Ephemeral Corruption.

Furthermore we introduce the variables n_i and n_r that refer to the maximum number of initiators and responders respectively: While both of these values are upper-bounded by n_p, especially n_i will most of the time be significantly smaller in our use-case, allowing us to get a better security-statement at no cost. For the same reason we separate n_s into n_{s_i} and n_{s_r} as the maximum number of sessions that an initiator or responder may respectively run: For the same reason for which there are often few initiators, these few initiators may run a significantly higher number of sessions than most responders. Lastly we remark that post-quantum KEMs often do not offer perfect correctness, that is even an honestly generated ciphertext might not successfully decapsulate to the same shared secret that the sender obtained when encapsulating. We will denote the probability of this happening to a KEM KEM as $\mathsf{KEM}.\delta$.

11.2. Pseudo-Random Hash-Object (PRHO)

A major component of our protocol is the precise mechanism used for hashing. PQNoise [4] introduced an abstracted interpretation of these hash-chains in the form of a (Noise-) Pseudo-Random Hash-Object (PRHO), that produces an unlimited number of values that are indistinguishable from randomness, once it has been fed randomness.

It furthermore proved that the way the hashing is done securely implements such a PRHO. Because this instantiation is precisely what we need as well, we point to the PQNoise-paper for a more detailed discussion and analysis and will henceforth treat the protocol as using a Noise-PRHO directly.

Formally a Noise-PRHO is a tuple of three deterministic algorithms: create, input, finalize, and an integer-constant n.

create$(1^\lambda) \to s$ (conceptually) takes a security-parameter λ and returns a state s.

input$(s, m) \to s', h$ takes a state s and message $m \in \{0,1\}^*$ and returns a new state s' and a list $h \in (\{0,1\}^\lambda)^n$ of hashes of length n .

finalize$(s, m) \to h$ works like input, except that it does not return a state.

For $n = 2$, which is the relevant case for us, the instantiation of this primitive that Noise and its derivatives use and that we recommend as well then looks as follows:

```
def create():
    return ""

def input(state, m):
    tmp = hmac_hash(state, m)
    new_state = hmac_hash(tmp, b"\x00")
    h1 = hmac_hash(tmp, new_state + b"\x01")
    h2 = hmac_hash(tmp, h1 + b"\x02")
    return new_state, [h1, h2]

def finalize(state, m):
    h_0, [h_1, h_2] = input(state, m)
    return [h_0, h_1]
```

We remark that the only situation where we require both outputs are the ones where finalize is called, making it a desirable optimization to skip the computation of `h_2` in all cases; we only include it here for the sake of staying in line with pre-existing literature.

11.3. Outline

While eCK-NEC covers both authenticity and confidentiality at the same time, we treat them largely separately as we consider that approach more intuitive.

11.3.1. Authenticity

The way the protocol provides authenticity is identical for both parties: It uses a long-term KEM to derive a shared secret that should only be known to the encapsulating party and the owner of the secret key and only accepts once the encapsulating party receives a response that is encrypted with an AEAD-scheme that uses a key derived from that shared secret. On a high-level they both work as follows:

- In the first game-hop we abort if there is ever a hash-collision. This is sound, because this should never happen if the hash-function is secure.
- In the second game-hop we guess the impersonated party and its peer and the attacked session and abort upon a wrong guess.
- In the third game-hop we replace the shared-secret with a random string. We derive the security of this step from the security of the long-term KEM in question.
- In the fourth game-hop we replace the subsequent outputs of the PRHO with randomness. We derive the security of this step from the security of the hash-object.
- In the fifth game-hop we abort the interaction upon receiving a reply in the challenge-session. This is secure, because a valid reply could be used to break the AEAD-security.
- At this point it is trivially impossible for the adversary to impersonate a party in the challenge-session.

11.3.2. Confidentiality

Once authenticity is fully established, confidentiality only has to deal with passive attacks. As we do not use keys that result from sessions that did not reach the stage where the final key was accepted, and because any session that reached that stage without being fully honest would be a break of authenticity, the only sessions that remain to be considered are sessions in which all messages are delivered honestly between the parties.

- In the first game-hop we abort if there is ever a hash-collision. This is sound, because this should never happen if the hash-function is secure.
- In the second game-hop we guess the attacked initiator and initiator-session and abort upon a wrong guess.
- In the third game we abort if there is ever a second honest initiator-session that uses the same ephemeral public key as the target-session.
- In the fourth game we guess the peered responder and responder-session that is targeted and abort upon a wrong guess.
- In the fifth game we abort if there is ever an honest responder session that recreates the ciphertext for the ephemeral KEM in a non-targeted session.
- In the sixth game we abort if a non-partnered session shares the handshake-hash with the target-session. (This is purely conceptual at this point.)
- In the seventh game-hop we replace the shared-secret with a random string. We derive the security of this step from the security of the ephemeral KEM in question.

- In the eighth game-hop we replace the subsequent outputs of the PRHO with randomness. We derive the security of this step from the security of the hash-object.
- In the ninth game-hop we replace the returned keys with randomness. We derive the security of this step from the PRF-security of the used key-derivation-function.
- At this point the final keys are random, which restricts the adversary to guessing a random bit with advantage 0.

11.4. Formal Security-Statements

Theorem 1. *There is no polynomial-time quantum-adversary that can win the* eCK-NEC*-game against Triple-KEM, with more than negligible probability. Specifically we find for all such adversaries $\mathcal{A}$:*

$$\mathsf{Adv}^{\textit{eCK-NEC}}_{\mathcal{A},\,\mathsf{3KEM}}(1^\lambda) \leq \left(\begin{array}{rrcl} & 3 & \cdot & \mathsf{Adv}^{\textit{coll-res}}_{\mathcal{A}_1,\,\mathsf{H}}(1^\lambda) \\ + & n_i \cdot n_{s_i} & \cdot & \mathsf{EKEM}.\delta \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3 & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A},\,\mathsf{EKEM}}(1^\lambda) \\ + & n_{s_r} \cdot n_i \cdot n_r \cdot \frac{1}{1-\mathsf{IKEM}\cdot\delta} & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A}_4,\,\mathsf{IKEM}}(1^\lambda) \\ + & n_{s_i} \cdot n_i \cdot n_r \cdot \frac{1}{1-\mathsf{RKEM}\cdot\delta} & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A}_4,\,\mathsf{RKEM}}(1^\lambda) \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3 & \cdot & \mathsf{Adv}^{\textit{PRHO}}_{\mathcal{A},\,\mathsf{NHO}}(1^\lambda) \\ + & (n_{s_i} + n_{s_r}) \cdot n_i \cdot n_r & \cdot & \mathsf{Adv}^{\textit{EUF-CMA}}_{\mathcal{A}_6,\,\mathsf{AEAD}}(1^\lambda) \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 2 & \cdot & \mathsf{Adv}^{\textit{PRF}}_{\mathcal{A},\,\mathsf{KDF}}(1^\lambda) \end{array}\right)$$

Proof. This follows directly from adding up the losses in Theorems 3-5, noting that $n_i \cdot n_{s_i} \cdot n_r + n_i \cdot n_r \cdot n_{s_r} + n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \leq n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3$, when summarizing the PRHO-losses and simplifying the result. □

Theorem 2. *There is no polynomial-time quantum-adversary that can win the* eCK-NEC*-game against Dual-KEM, with more than negligible probability. Specifically we find for all such adversaries $\mathcal{A}$:*

$$\mathsf{Adv}^{\textit{eCK-NEC}}_{\mathcal{A},\,\mathsf{2KEM}}(1^\lambda) \leq \left(\begin{array}{rrcl} & 2 & \cdot & \mathsf{Adv}^{\textit{coll-res}}_{\mathcal{A}_1,\,\mathsf{H}}(1^\lambda) \\ + & n_i \cdot n_{s_i} & \cdot & \mathsf{EKEM}.\delta \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3 & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A},\,\mathsf{EKEM}}(1^\lambda) \\ + & n_{s_r} \cdot n_i \cdot n_r \cdot \frac{1}{1-\mathsf{IKEM}\cdot\delta} & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A}_4,\,\mathsf{IKEM}}(1^\lambda) \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 2 & \cdot & \mathsf{Adv}^{\textit{PRHO}}_{\mathcal{A},\,\mathsf{NHO}}(1^\lambda) \\ + & n_{s_r} \cdot n_i \cdot n_r & \cdot & \mathsf{Adv}^{\textit{EUF-CMA}}_{\mathcal{A}_6,\,\mathsf{AEAD}}(1^\lambda) \\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 2 & \cdot & \mathsf{Adv}^{\textit{PRF}}_{\mathcal{A},\,\mathsf{KDF}}(1^\lambda) \\ + & & & \mathsf{Adv}^{\textit{eCK-NEC}_{Case\ A}}_{\mathcal{A},\,\mathsf{2KEM}}(1^\lambda) \end{array}\right)$$

Where $\mathsf{Adv}_{\mathcal{A},\,\mathsf{2KEM}}^{\textit{eCK-NEC}_{\textit{Case A}}}(1^\lambda)$ *refers to the maximum achievable advantage for the adversary to cause an unpeered, complete initiator-session.*

Proof. This follows directly from adding up the losses in Theorems 4 and 5 as well as $\mathsf{Adv}_{\mathcal{A},\,\mathsf{2KEM}}^{\mathsf{eCK\text{-}NEC}_{\text{Case A}}}(1^\lambda)$ and, noting that $n_i \cdot n_r \cdot n_{s_r} + n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \le n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 2$, when summarizing the PRHO-losses and simplifying the result. □

11.5. Proof

There are three separate cases:

1. The initiator-session π_i^s has no peered session.
2. The responder-session π_i^s has no peered session.
3. The session π_i^s has a peered session.

As these cases are defined to be mutually exclusive by definition and cover all possibilities, We analyze them separately and note that the adversarial advantage against our protocol is upper-bounded by the sum of the adversarial advantages that can be achieved in the sub-cases.

11.5.1. Case A: Unpeered Initiator Session

In this first case we treat the situation where the adversary impersonates the responder (aka the satellite).

Theorem 3. *There is no polynomial-time quantum-adversary that can win the eCK-NEC-game of the triple-KEM protocol in the case where the target session is an initiator-session that has no peered session, with more than negligible probability. Specifically we find for all such adversaries $\mathcal{A}$:*

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{3KEM}}^{\textit{eCK-NEC}_{\textit{Case A}}}(1^\lambda) \le \begin{pmatrix} & & & \mathsf{Adv}_{\mathcal{A}_1,\,\mathsf{H}}^{\textit{coll-res}}(1^\lambda) \\ + & n_{s_i} \cdot n_i \cdot n_r \cdot \frac{1}{1-\mathsf{RKEM}.\delta} & \cdot & \mathsf{Adv}_{\mathcal{A}_4,\,\mathsf{RKEM}}^{\textit{IND-CCA}}(1^\lambda) \\ + & n_{s_i} \cdot n_i \cdot n_r & \cdot & \mathsf{Adv}_{\mathsf{NHO},\,\mathcal{A}_5}^{\textit{PRHO}}(1^\lambda) \\ + & n_{s_i} \cdot n_i \cdot n_r & \cdot & \mathsf{Adv}_{\mathcal{A}_6,\,\mathsf{AEAD}}^{\textit{EUF-CMA}}(1^\lambda) \end{pmatrix}$$

Proof. Let ***Game 0*** be the original eCK-NEC game.

In ***Game 1*** we abort if there is ever a hash-collision. To show that this replacement is sound, we initialize a collision-resistance-challenger for H and use whatever collision would trigger an abort to win the collision-resistance-game. Since we always win that game in case of an abort that results from the difference between *Game 0* and *Game 1* we find:

$$\Pr\left[\text{break}_1\right] \leq \Pr\left[\text{break}_0\right] + \mathsf{Adv}^{\text{coll-res}}_{\mathcal{A}_1,\mathsf{H}}\left(1^\lambda\right)$$

In ***Game 2*** we guess the target-session, the party running it and the impersonated party and abort upon a wrong guess. As there are at most n_i possible initiators, n_r possible responders, and up to n_{s_i} initiator-sessions we find:

$$\Pr\left[\text{break}_2\right] \leq n_{s_i} \cdot n_i \cdot n_r \cdot \Pr\left[\text{break}_1\right]$$

In ***Game 3*** we replace the shared secret `k_sat` of the challenge-session with a random string. To show that this replacement is sound we initialize an $\mathsf{IND\text{-}CCA}$-challenger for the responder-KEM and perform the following substitutions:

- Instead of generating its long-term key-pair honestly, the responder will use the challenge-public key.
- Whenever the responder needs to decapsulate a ciphertext, it uses the decapsulation-oracle of the $\mathsf{IND\text{-}CCA}$-game.
- Instead of generating the ciphertext and shared secret honestly in the challenge-session, the initiator uses the challenge-ciphertext and the challenge-key.
- If any honest session recreates the challenge-ciphertext by chance, we will compare the encapsulated key to the challenge-key and use the result to win the $\mathsf{IND\text{-}CCA}$-game and abort. The probability of winning in this case can be lower-bounded to $1 - \delta$, as the probability of two encapsulations of a δ-correct KEM that result in the same ciphertext for the same public key can at most be δ:

  Assume to the contrary that the probability is $\varepsilon > \delta$. Let m_0 be an honestly generated ciphertext whose encapsulation claimed that the encapsulated key was k. By the definition of a δ-correct KEM, there is now a probability of $1 - \delta$ that the decapsulation outputs the same k. By our assumption there is however also a probability ε that another invocation of the encapsulation-algorithm would result in a different shared secret k' during the encapsulation. This means that there is now a $(1 - \delta) \cdot \varepsilon$-probability of the second encapsulation decapsulating incorrectly and a probability of δ that the first decapsulated wrongly. But for $0 < \delta < \varepsilon < 1$ we have $(1 - \delta) > 0$ and thus find that $(1 - \delta) \cdot \varepsilon + \delta > \delta$ which is in contradiction to the assumption that the probability of a decryption-failure is at most δ.

  As a consequence there is a slight additional loss-factor of $\frac{1}{1-\delta}$ in the reduction of this game.
- if the responder receives the challenge-ciphertext, it will use the challenge-key instead of decapsulating. Since this can only happen in a session that does not match the challenge-session (by the definition of case A) and is not partnered with an honest session (by the previous item), there is no need to modify any other session in that case.

If the challenge-bit of the IND-CCA-challenger is 0, then this is a purely conceptual change and we are in *Game 2*. Otherwise the challenge-key is truly random and we are in *Game 3*. Thus we find:

$$\Pr[\text{break}_3] \leq \Pr[\text{break}_2] + \frac{1}{1 - \mathsf{RKEM}.\delta} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A}_4, \mathsf{RKEM}}(1^\lambda)$$

In ***Game 4*** we replace all outputs of the (implicit) hash-object after inputting the shared secret `k_sat` in the challenge-session and all responder-sessions that received the challenge-public-key, with random values. To show that this replacement is sound we initialize a PRHO-challenger for NHO and replace the initiator's direct use of NHO with the oracles in the following way: Whenever the initiator and the responder start a session and would normally initialize a hash-object, they will instead call Create and use the returned identifier i for all oracle invocations in that session. Whenever they would normally use the input/finalize functions of NHO it will instead invoke the In/Fin oracle, with one exception: When they would normally input `k_sat`, they will instead invoke Rand. This substitution is valid since `k_sat` is an independent random value by *Game 4*. If the challenge bit b of the PRHO game is 0, this is a purely conceptual change and we are in *Game 3*. Otherwise, all outputs after inputting `k_sat` get replaced with independent random values and we are in *Game 4*. Thus:

$$\Pr[\text{break}_4] \leq \Pr[\text{break}_3] + \mathsf{Adv}^{\mathsf{PRHO}}_{\mathsf{NHO}, \mathcal{A}_5}(1^\lambda)$$

In ***Game 5*** we abort after receiving a reply to the target-session.

To show that this replacement is sound, we initialize an EUF-CMA-challenger for AEAD and forward `c_2` to it as a potential forgery. This substitution is sound because the AEAD-key `k_1` is truly random by *Game 4* and because `c_2` is a fresh ciphertext:

As there is no matching session to the challenge-session (by the definition of case A), and no collisions for the hash-function (by *Game 1*), and the expected response is the only (and therefore last) message sent by the responder, and because `c_2` cannot be a replay of `c_1`, as they use both a different nonce and a different handshake-hash, and the handshake-hash of the challenge-session is unique among all honest sessions.

This means that the associated data of `c_2` is different from the associated data used in any other honest session, which means that `c_2` has to be fresh.

If the ciphertext is not valid, we would abort in any case and thus the behavior is identical to *Game 4*. Otherwise the ciphertext is a valid forgery and we win the EUF-CMA-game and thus find:

$$\Pr[\text{break}_5] \leq \Pr[\text{break}_4] + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}_6, \mathsf{AEAD}}(1^\lambda)$$

At this point the target-session never successfully completes and we therefore find trivially that:

$$\Pr[\text{break}_5] = 0$$

By summarizing the losses up to this point, we find the adversarial advantage stated in Theorem 3. □

11.5.2. Case B: Unpeered Responder Session

In this second case we treat the situation where an attacker attempts to impersonate the initiator (aka mission-control). This is largely analogous to Case A, with aside from some substitutions of instances the most notable deviation being the need to create a ciphertext for the AEAD-scheme. Furthermore this case applies to both the Dual-KEM and the Triple-KEM protocol without any differences, as all of these differences are in parts of the protocol that have no impact here.

Theorem 4. *There is no polynomial-time quantum-adversary that can win the eCK-NEC-game of the triple-KEM protocol in the case where the target session is a responder-session that has no peered session, with more than negligible probability. Specifically we find for all such adversaries $\mathcal{A}$:*

$$\max\left(\begin{array}{l}\mathsf{Adv}^{\textit{eCK-NEC}_{\textit{Case A}}}_{\mathcal{A},\mathsf{3KEM}}(1^\lambda),\\ \mathsf{Adv}^{\textit{eCK-NEC}_{\textit{Case A}}}_{\mathcal{A},\mathsf{2KEM}}(1^\lambda)\end{array}\right) \leq \left(\begin{array}{cccc} & & & \mathsf{Adv}^{\textit{coll-res}}_{\mathcal{A}_1,\mathsf{H}}(1^\lambda) \\ + & n_{s_r}\cdot n_i\cdot n_r\cdot\frac{1}{1-\mathsf{IKEM}.\delta} & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A}_4,\mathsf{IKEM}}(1^\lambda) \\ + & n_{s_r}\cdot n_i\cdot n_r & \cdot & \mathsf{Adv}^{\textit{PRHO}}_{\mathsf{NHO},\mathcal{A}_5}(1^\lambda) \\ + & n_{s_r}\cdot n_i\cdot n_r & \cdot & \mathsf{Adv}^{\textit{EUF-CMA}}_{\mathcal{A}_6,\mathsf{AEAD}}(1^\lambda)\end{array}\right)$$

Proof. Let ***Game 0*** be the original eCK-NEC game.

In ***Game 1*** we abort if there is ever a hash-collision. To show that this replacement is sound, we initialize a collision-resistance-challenger for H and use whatever collision would trigger an abort to win the collision-resistance-game. Since we always win that game in case of an abort that results from the difference between *Game 0* and *Game 1* we find:

$$\Pr[\text{break}_1] \leq \Pr[\text{break}_0] + \mathsf{Adv}^{\mathsf{coll\text{-}res}}_{\mathcal{A}_1,\mathsf{H}}(1^\lambda)$$

In ***Game 2*** we guess the target-session, the party running it and the impersonated party and abort upon a wrong guess. As there are at most n_i possible initiators, n_r possible responders, and n_{s_r} sessions we find:

$$\Pr[\text{break}_2] \leq n_{s_r}\cdot n_i\cdot n_r\cdot \Pr[\text{break}_1]$$

In ***Game 3*** we replace the shared secret `k_mc` of the challenge-session with a random string. To show that this replacement is sound we initialize an $\mathsf{IND\text{-}CCA}$-challenger for the initiator-KEM and perform the following substitutions:

- Instead of generating its long-term key-pair honestly, the initiator will use the challenge-public key.
- Whenever the initiator needs to decapsulate a ciphertext, it uses the decapsulation-oracle of the $\mathsf{IND\text{-}CCA}$-game.
- Instead of generating the ciphertext and shared secret honestly in the challenge-session, the responder uses the challenge-ciphertext and the challenge-key.
- If any honest session recreates the challenge-ciphertext by chance, we will compare the encapsulated key to the challenge-key and use the result to win the $\mathsf{IND\text{-}CCA}$-game and abort. The probability of winning in this case can be lower-bounded to $1 - \delta$, as the probability of two encapsulations of a δ-correct KEM that result in the same ciphertext for the same public key can at most be δ:

  Assume to the contrary that the probability is $\varepsilon > \delta$. Let m_0 be an honestly generated ciphertext whose encapsulation claimed that the encapsulated key was k. By the definition of a δ-correct KEM, there is now a probability of $1 - \delta$ that the decapsulation outputs the same k. By our assumption there is however also a probability ε that another invocation of the encapsulation-algorithm would result in a different shared secret k' during the encapsulation. This means that there is now a $(1 - \delta) \cdot \varepsilon$-probability of the second encapsulation decapsulating incorrectly and a probability of δ that the first decapsulated wrongly. But for $0 < \delta < \varepsilon < 1$ we find that $(1-\delta)\cdot\varepsilon+\delta > \delta$ which is in contradiction to the assumption that the probability of a decryption-failure is at most δ.

  As a consequence there is a slight additional loss-factor of $\frac{1}{1-\delta}$ in the reduction of this game.
- if the initiator receives the challenge-ciphertext, it will use the challenge-key instead of decapsulating. Since this can only happen in a session that does not match the challenge-session (by the definition of case A) and is not partnered with an honest session (by the previous item), there is no need to modify any other session in that case.

If the challenge-bit of the $\mathsf{IND\text{-}CCA}$-challenger is 0, then this is a purely conceptual change and we are in *Game 2*. Otherwise the challenge-key is truly random and we are in *Game 3*. Thus we find:

$$\Pr\left[\text{break}_3\right] \leq \Pr\left[\text{break}_2\right] + \frac{1}{1 - \mathsf{IKEM}.\delta} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A}_4,\mathsf{IKEM}}\left(1^\lambda\right)$$

In ***Game 4*** we replace all outputs of the (implicit) hash-object after inputting the shared secret `k_mc` in the challenge-session and all responder-sessions that received the challenge-public-key, with random values. To show that this replacement is sound we initialize a PRHO-challenger for NHO and replace the initiator's direct use of NHO with the oracles in the following way: Whenever the initiator and the responder start a session and would normally initialize a hash-object, they will instead call Create and use the returned identifier i for all oracle invocations in that session. Whenever they would normally use the input/finalize functions of NHO it will instead invoke the In/Fin oracle, with one exception: When they would normally input `k_mc`, they will instead invoke Rand. This substitution is valid since `k_mc` is an independent random value by *Game 4*. If the challenge bit b of the PRHO game is 0, this is a purely conceptual change and we are in *Game 3*. Otherwise, all outputs after inputting `k_mc` get replaced with independent random values and we are in *Game 4*. Thus:

$$\Pr\left[\text{break}_4\right] \leq \Pr\left[\text{break}_3\right] + \mathsf{Adv}^{\mathsf{PRHO}}_{\mathsf{NHO},\mathcal{A}_5}\left(1^\lambda\right)$$

In ***Game 5*** we abort after receiving a reply to the target-session's message.

To show that this replacement is sound, we initialize an EUF-CMA-challenger for AEAD perform the following substitutions:

- The responder will disregard `k_3` and instead use the encryption-oracle provided by the EUF-CMA-game to compute `c_4` if necessary (`c_4` encrypts the updated long-term key `pk_sat_new` and is thus never needed in Dual-KEM).
- Upon receiving `c_5`, we forward it to the EUF-CMA-challenger as a potential forgery.

This substitution is sound because the AEAD-key `k_3` is truly random by *Game 4* and because `c_5` cannot be a replay of `c_4`, as they use both a different nonce and a different handshake-hash as associated data by *Game 1*.

As there is no matching session to the challenge-session (by the definition of case A), and no collisions for the hash-function (by *Game 1*), and the expected response is the last message in the handshake, the handshake-hash of the challenge-session is unique among all honest sessions.

If the ciphertext is not valid, we would abort in any case and thus the behavior is identical to *Game 4*. Otherwise the ciphertext is a valid forgery and we win the EUF-CMA-game and thus find:

$$\Pr\left[\text{break}_5\right] \leq \Pr\left[\text{break}_4\right] + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{A}_5,\mathsf{AEAD}}\left(1^\lambda\right)$$

At this point the target-session never successfully completes and we therefore find trivially that:

$$\Pr\left[\text{break}_5\right] = 0$$

By summarizing the losses up to this point, we find the adversarial advantage stated in Theorem 4. □

11.5.3. Case C: Peered Session

This last case intuitively treats adversaries that attempt to break the confidentiality of the protocol.

Theorem 5. *There is no polynomial-time quantum-adversary that can win the* eCK-NEC*-game of the triple-KEM protocol in the case where the target session has a peered session, with more than negligible probability. Specifically we find for all such adversaries $\mathcal{A}$:*

$$\max\left(\begin{array}{c}\mathsf{Adv}^{\textit{eCK-NEC}_{Case\ C}}_{\mathcal{A},\,\mathsf{3KEM}}(1^\lambda),\\ \mathsf{Adv}^{\textit{eCK-NEC}_{Case\ C}}_{\mathcal{A},\,\mathsf{2KEM}}(1^\lambda)\end{array}\right) \leq \left(\begin{array}{crcl} & & & \mathsf{Adv}^{\textit{coll-res}}_{\mathcal{A}_1,\,\mathsf{H}}(1^\lambda)\\ + & n_i \cdot n_{s_i} & \cdot & \mathsf{EKEM}.\delta\\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3 & \cdot & \mathsf{Adv}^{\textit{IND-CCA}}_{\mathcal{A},\,\mathsf{EKEM}}(1^\lambda)\\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} & \cdot & \mathsf{Adv}^{\textit{PRHO}}_{\mathcal{A},\,\mathsf{NHO}}(1^\lambda)\\ + & n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 2 & \cdot & \mathsf{Adv}^{\textit{PRF}}_{\mathcal{A},\,\mathsf{KDF}}(1^\lambda)\end{array}\right)$$

Proof. Let ***Game 0*** be the original eCK-NEC game.

In ***Game 1*** we abort if there is ever a hash-collision. To show that this replacement is sound, we initialize a collision-resistance-challenger for H and use whatever collision would trigger an abort to win the collision-resistance-game. Since we always win that game in case of an abort that results from the difference between *Game 0* and *Game 1* we find:

$$\Pr[\text{break}_1] \leq \Pr[\text{break}_0] + \mathsf{Adv}^{\mathsf{coll\text{-}res}}_{\mathcal{A}_1,\,\mathsf{H}}(1^\lambda)$$

In ***Game 2*** we guess the initiator of the target-interaction as well as the initiator-session of that interaction and abort upon a wrong guess. Because there are n_i possible initiators who initiate up to n_{s_i} sessions each we find:

$$\Pr[\text{break}_2] \leq n_i \cdot n_{s_i} \cdot \Pr[\text{break}_1]$$

In ***Game 3*** we abort if there is ever an honest session that shares the public key with the initiator-session. To show that this substitution is valid we initialize an $\mathsf{IND\text{-}CCA1}$-challenger for EKEM and embed the challenge public-key into the initiators target-session, using the decapsulation-oracle for decapsulation. If we now honestly recreate that public key in a different session, we can use the secret key to break the security of EKEM: Because we assume that the probability of an decapsulation-failure with honestly generated ciphertexts (such as the challenge-ciphertext) of EKEM is at most $\mathsf{EKEM}.\delta$ and because the ciphertexts depends only on the public key and independent randomness, the secret key of the colliding public key is a (or sometimes: the) matching secret key for the challenge public key. (We remark that this doesn't even just break IND, but is in fact a full key-extraction attack.) We thus find:

$$\Pr[\text{break}_3] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA1}}_{\mathcal{B},\,\mathsf{EKEM}}(1^\lambda) + \mathsf{EKEM}.\delta + \Pr[\text{break}_2]$$

In ***Game 4*** we guess the responder of the target-interaction and the targeted responder-session and abort upon a wrong guess. As there are n_r potential responders and n_{s_r} responder-sessions we find:

$$\Pr[\text{break}_4] \leq n_r \cdot n_{s_r} \cdot \Pr[\text{break}_3]$$

In ***Game 5*** we abort if there is ever an honest responder-session that is not part of the target-interaction that produces the same ciphertext for the EKEM that is used in the challenge-interaction for the same public-key that is used in the challenge-interaction. (By *Game 3* there is only one honest initiator-session that uses that public key, but there may be dishonest ones in addition.) To show that this substitution is sound we initialize an IND-CPA-challenger for EKEM and again replace the public key used in the initiator's challenge-session with the challenge public-key. Furthermore we replace the ciphertext used in the challenge-session with the challenge-ciphertext and the resulting key with the challenge-key. If any later honest session recreates the challenge-ciphertext, then the contained key can be used to trivially win the IND-CPA-game. (We remark that this is in fact a full plaintext-recovery attack.) We thus find:

$$\Pr[\text{break}_5] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B},\mathsf{EKEM}}(1^\lambda) + \Pr[\text{break}_4]$$

In ***Game 6*** we abort if there is ever an honest session that is not part of the target-interaction but shares the handshake-hash `h_6` with it.

This is perfectly indistinguishable from *Game 5*, because *Game 1* ensures that there are no hash-collisions, *Game 3* ensures that the EKEM public key of the target-interaction is different from that in all other honest initiator-sessions, and *Game 5* ensures that the EKEM-ciphertext in the challenge interaction is different from that in all other honest responder-sessions. Since both of these values make their way into the handshake-hash and since any interaction that involves at least one honest party will therefore add at least one value that is different from the one in the challenge-interaction we can conclude that the handshake-hash of the challenge-interaction is not repeated in any honest sessions.

Because this is a purely conceptual change at this point we find:

$$\Pr[\text{break}_6] = \Pr[\text{break}_5]$$

In ***Game 7*** we replace the shared secret `k_e` that is encapsulated for the target-session's ephemeral key and fed into the pseudo-random hash-object with a random string. To show that this replacement is sound we initialize an IND-CCA-challenger for the ephemeral KEM and perform the following substitutions:

Instead of generating the ephemeral keypair for the target-session himself, the initiator will use the challenge-public key of the IND-CCA-game.

Instead of generating the KEM-ciphertext honestly, the responder will use the challenge-ciphertext of the IND-CCA-game.

Both parties will use the challenge-key of the IND-CCA-game in place of any encapsulated keys.

In the event that the initiator receives a different ciphertext than the challenge-ciphertext (due to adversarial actions), he will use the decapsulation-oracle of the IND-CCA-game.

If the challenge-bit of the IND-CCA-game is 0, this is a purely conceptual change and we are in *Game 6*. Otherwise the challenge-key is a truly random string and we are in *Game 7* and thus find:

$$\Pr[\text{break}_7] \leq \Pr[\text{break}_6] + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{EKEM}}(1^\lambda)$$

In ***Game 8*** we replace the outputs of the pseudo-random hash-object, after feeding it the EKEM-key, with random strings. To show that this replacement is sound we initialize a PRHO-challenger for NHO and perform the following substitutions:

Whenever the initiator and the responder start a session and would normally initialize a hash-object, they will instead call Create and use the returned identifier i for all oracle invocations in that session. Whenever they would normally use the input/finalize functions of NHO it will instead invoke the In/Fin oracle, with one exception: When they would normally input `k_e`, they will instead invoke Rand. This substitution is valid since `k_e` is an independent random value by *Game 7*. If the challenge bit b of the PRHO game is 0, this is a purely conceptual change and we are in *Game 7*. Otherwise, all outputs after inputting `k_e` get replaced with independent random values and we are in *Game 8*. Thus:

$$\Pr[\text{break}_8] \leq \Pr[\text{break}_7] + \mathsf{Adv}^{\mathsf{PRHO}}_{\mathcal{A},\mathsf{NHO}}(1^\lambda)$$

In ***Game 9*** we replace the final keys with random strings. To show that this replacement is sound we will use two sub-games.

In ***Game 9.A*** we replace `key_mc` with a random string. To show that this replacement is sound we initialize a PRF-challenger for KDF and replace the evaluation of KDF using `pre_key_mc` as key with an invocation of the challenge-oracle. This substitution is sound because `pre_key_mc` is truly random by *Game 8* and because `h_6` is a different message than used in any other session by *Game 6*, including sessions where the peer is not honest. If the challenge-bit is 0, then this is therefore a purely conceptual change and we are in *Game 8*. Otherwise `key_mc` is a truly random string and we are in *Game 9.A* and find:

$$\Pr[\text{break}_{9.A}] \leq \Pr[\text{break}_8] + \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{KDF}}(1^\lambda)$$

In ***Game 9.B*** we replace `key_sat` with a random string. To show that this replacement is sound we initialize a PRF-challenger for KDF and replace the evaluation of KDF using `pre_s_g` as key with an invocation of the challenge-oracle. This substitution is sound because `pre_s_g` is truly random by *Game 8* and because `h_6` is a different message than used in any other session by *Game 6*, including sessions where the peer is not honest. If the challenge-bit is 0, then this is therefore a purely conceptual change and we are in *Game 9.A*. Otherwise `key_mc` is a truly random string and we are in *Game 9.B* and find:

$$\Pr\left[\text{break}_{9.B}\right] \leq \Pr\left[\text{break}_{9.A}\right] + \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\,\mathsf{KDF}}\left(1^{\lambda}\right)$$

By noting that *Game 9.B*=*Game 9* and summarizing the losses in the sub-games we find:

$$\Pr\left[\text{break}_{9}\right] \leq \Pr\left[\text{break}_{8}\right] + 2 \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\,\mathsf{KDF}}\left(1^{\lambda}\right)$$

At this point we note that the final keys are originally outputs of the PRHO-challenger and thus random; Therefore the adversary has to distinguish a random string from a random string which trivially limits the adversarial to $\frac{1}{2}$ with an advantage of 0:

$$\Pr\left[\text{break}_{9}\right] = 0$$

By summarizing the security-losses in the previous games and noting that $n_i \cdot n_{s_i} \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}CCA1}}_{\mathcal{B},\,\mathsf{EKEM}}\left(1^{\lambda}\right) + n_r \cdot n_{s_r} \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B},\,\mathsf{EKEM}}\left(1^{\lambda}\right) + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\,\mathsf{EKEM}}\left(1^{\lambda}\right)\right)\right) \leq n_i \cdot n_{s_i} \cdot n_r \cdot n_{s_r} \cdot 3 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\,\mathsf{EKEM}}\left(1^{\lambda}\right)$ to simplify the resulting equation, we find the adversarial advantage claimed in Theorem 5. □

References

[1] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`.

[2] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Technical report, Sept 2022.

[3] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions`.

[4] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Fiona Johanna Weber. Post quantum noise. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 97–109. ACM Press, November 2022.

[5] ANSI X9.62-1999. *Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*, 1999.

[6] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. BIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[7] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-Wing: The hybrid KEM you've been looking for. Cryptology ePrint Archive, Paper 2024/039, 2024. https://eprint.iacr.org/2024/039.

[8] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[9] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, Heidelberg, September / October 2011.

[10] Daniel J. Bernstein, Andreas Hülsing, Tanja Lange, and Evangelos Rekleitis. Post-quantum cryptography - integration study, 2022. https://www.enisa.europa.eu/publications/post-quantum-cryptography-integration-study.

[11] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 464–479. Springer, Heidelberg, August 2022.

[12] Ward Beullens, Jan-Pieter D'Anvers, Andreas Hülsing, Tanja Lange, Lorenz Panny, Cyprien de Saint Guilhem, Nigel P. Smart, Evangelos Rekleitis, Angeliki Aktypi, and Athanasios-Vasileios Grammatopoulos. Post-quantum cryptography: Current state and quantum mitigation, 2021. https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation/@@download/fullReport.

[13] Nina Bindel and Britta Hale. A note on hybrid signature schemes. Cryptology ePrint Archive, Paper 2023/423, 2023. https://eprint.iacr.org/2023/423.

[14] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[15] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 423–447. Springer, Heidelberg, April 2023.

[16] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, Heidelberg, December 2018.

[17] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[18] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291.

[19] Eric Crockett, Christian Paquin, and Douglas Stebila. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Cryptology ePrint Archive, Report 2019/858, 2019. https://eprint.iacr.org/2019/858.

[20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[21] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394. Springer, Heidelberg, December 2018.

[22] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias J. Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nis

t.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[23] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 190–218. Springer, Heidelberg, March 2018.

[24] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, Heidelberg, November 2017.

[25] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS$^+$. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[26] Andreas Hülsing, Denise Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, RFC Editor, 05 2018.

[27] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321. IEEE Computer Society Press, May 2021.

[28] ISE Crypto PQC working group. Securing tomorrow today: Why Google now protects its internal communications from quantum threats, 2022. https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms.

[29] Stavros Kousidis, Falko Strenzke, and Aron Wussler. Post-quantum cryptography in openpgp draft-wussler-openpgp-pqc-00, 12 2022. https://datatracker.ietf.org/doc/draft-wussler-openpgp-pqc/.

[30] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073, 2006. https://eprint.iacr.org/2006/073.

[31] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[32] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Heidelberg, May / June 2010.

[33] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 448–471. Springer, Heidelberg, April 2023.

[34] David A. McGrew, Michael Curcio, and Scott R. Fluhrer. Hash-Based Signatures. RFC 8554, RFC Editor, 2019.

[35] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[36] OpenSSH team. Openssh 9.0 release note, 2022. `https://www.openssh.com/txt/release-9.0`.

[37] Mike Ounsworth, Aron Wussler, and Stavros Kousidis. Combiner function for hybrid key encapsulation mechanisms (Hybrid KEMs). Internet-Draft draft-ounsworth-cfrg-kem-combiners-04, Internet Engineering Task Force, July 2023. Work in Progress.

[38] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in TLS. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 72–91. Springer, Heidelberg, 2020.

[39] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[40] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 472–503. Springer, Heidelberg, April 2023.

[41] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `https://eprint.iacr.org/2006/145`.

[42] John M. Schanck, William Whyte, and Zhenfei Zhang. Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.2. Internet-Draft draft-whyte-qsh-tls12-02, Internet Engineering Task Force, July 2016. Work in Progress.

[43] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[44] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1461–1480. ACM Press, November 2020.

[45] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. Efficient key recovery for all HFE signature variants. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 70–93, Virtual Event, August 2021. Springer, Heidelberg.

[46] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

A. Detailed KEM+Signature Protocol

The detailed protocol for the KEM+Signature-update contains protection against a power-outage; this is especially necessary for one-time-signatures as they must not be used to sign a different message, even if a handshake is interrupted.

The idea here is to derandomize the protocol by sampling a random seed before anything else and storing it in safe memory. In the event of a power-outage this seed will not be regenerated, but just loaded from storage. After the completion of the handshake it will on the other hand be deleted.

This seed is then fed into a cryptographically secure random-number-generator whose output is the sole randomness used in the exchange from that point onwards. If the satellite has a power-outage during the exchange it will restart the exchange from the beginning waiting for mission control to resend its initial packet. Mission control must resend the exact same packet until it arrives and the satellite can verify it. In the case of a signature-chain there is no risk of a different message arriving since the signing-key of mission control is only used once for the specific message that the satellite should receive. The satellite will then recompute its response deterministically and resend it, until mission control receives it.

In the case of a power-outage at mission control, the recreatable randomness allows to recreate the exact initial message and update in a similar manner.

We remark that this protection is possible in general, but possibly less relevant for KEM-based challenge-response protocols, since they can more easily be rerun independently in case of a power-failure.

```python
def send_1():
    if seed is None: # Possibly set from aborted earlier interaction
        seed = true_rng()
    if ctr_mc is None:
        ctr_mc = 0
    prng = csprng(seed)
    prho_state_mc = NoiseHashObject.gen()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN, pk_mc, pk_sat)
    k_0 = prho_state_mc.input(psk)
    pk_e, sk_e = EKEM.gen(prng())
    h_1 = H(h_0, pk_e)
    ctr_mc += 1
    payload = ctr_mc
    if update_longterm:
        pk_mc_new, sk_mc_new = Sig.gen(prng())
        payload += pk_mc_new
    c_0 = AEAD.enc(k_0, payload, 0, h_0)
    h_2 = H(h_1, c_0)
    sig_mc = Sig.sign(sk_mc, h_2) # optionally encrypt
    sk_mc = sk_mc_new # Optional: wait until response for key-confirmation
    h_3 = H(h_2, sig_mc)
    send(pk_e, c_0, sig_mc)
```

```

def receive_1((pk_e, c_0, sig_mc)):
    if seed is None: # Possibly set from aborted earlier interaction
        seed = true_rng()
    if ctr_sat is None:
        ctr_sat = 0
    prng = csprng(seed)
    prho_state_sat = NoiseHashObject.gen()
    psk = (key_mc + key_sat) if key_mc is not None else b'\0\0...'
    h_0 = H(PROTOCOL_TOKEN, pk_mc, pk_sat)
    k_0 = prho_state_sat.input(psk)
    h_1 = H(h_0, pk_e)
    ctr_mc, pk_mc_new = AEAD.dec(k_0, c_0, 0, h_0)
    h_2 = H(h_1, c_0)
    if not Sig.verify(pk_mc, sig_mc, h2) or ctr_mc != ctr_sat + 1:
        error()
    ctr_sat = ctr_mc
    h_3 = H(h_2, sig_mc)
    if update_longterm:
        pk_mc = pk_mc_new

def send_2():
    c_e, k_e = EKEM.enc(prng())
    # c_e may optionally be encrypted with k_0
    h_4 = H(h_3, c_e)
    k_1 = prho_state_sat.input(k_e)
    payload = ""
    if update_longterm:
        pk_sat_new, sk_sat_new = Sig.gen(prng())
        payload += pk_sat_new
    c_1 = AEAD.enc(k_1, payload, 0, h_4)
    h_5 = H(h_4, c_1)
    sig_sat = Sig.sign(sk_sat, h_5) # optionally encrypt
    sk_sat = sk_sat_new # Optional: wait until response for key-confirmation
    h_6 = H(h_5, sig_sat)
    send(c_e, c_1, sig_sat)
    pre_key_mc,pre_key_sat = prho_state_sat.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, k_s

def receive_2((c_e, c_1, sig_sat)):
    k_e = EKEM.dec(sk_e, c_e)
    h_4 = H(h_3, c_e)
    k_1 = prho_state_mc.input(k_e)
    pk_sat_new = AEAD.dec(k_1, c_1, 0, h_4)
    h_5 = H(h_4, c_1)
    h_6 = H(h_5, sig_sat)
    if not Sig.verify(pk_sat, sig_sat, h_4):
        error()
    if update_longterm:
        pk_sat = pk_sat_new
    pre_key_mc,pre_key_sat = prho_state_mc.finalize()
    key_mc = KDF(pre_key_mc, h_6)
    key_sat = KDF(pre_key_sat, h_6)
    return key_mc, key_sat
```

B. Packet-Sizes

The following packet sizes assume the existence of `psk` and include the corresponding authentication tag.

- Triple-KEM:
  - Packet 1: RKEM-ct + AEAD-tag + EKEM-pk + IKEM-pk[opt1] + AEAD-tag
  - Packet 2: EKEM-ct + AEAD-tag + IKEM-ct + AEAD-tag + RKEM-pk[opt2] + AEAD-tag[opt2]
  - Packet 3: AEAD-tag
  - TK(Kyber512,McEliece348864)
    - Packet 1: 96 + 16 + 800 + 261120[opt1] + 16 = 928/262048
    - Packet 2: 768 + 16 +96 + 16 + 261120[opt2] + 16[opt2] = 896/262032
    - Packet 3: 16
  - TK(Kyber512+X25519)
    - Packet 1: 800 + 16 + 832 + 832[opt1] + 16 = 1664/2496
    - Packet 2: 800 + 16+ 800 + 16 + 832[opt2] + 16[opt2] = 1632/2480
    - Packet 3: 16
  - TK(Kyber768+X25519)
    - Packet 1: 1120 + 16 + 1216 + 1216[opt1] + 16 = 2368/3584
    - Packet 2: 1120 + 16 + 1120 + 16 + 1216[opt2] + 16[opt2] = 2272/3504
    - Packet 3: 16
  - TK(Kyber1024+X25519)
    - Packet 1: 1600 + 16 + 1600 + 1600[opt1] + 16 = 3232/4832
    - Packet 2: 1600 + 16 + 1600 + 16 + 1600[opt2] + 16[opt2] = 3232/4848
    - Packet 3: 16
- Sign+KEM:
  The following packet sizes assume the existence of `psk` and include the corresponding authentication tag.
  - Packet 1: ctr + EKEM-pk + Sig-pk[opt1] + AEAD-tag + Sig
  - Packet 2: EKEM-ct, Sig-pk[opt2] + AEAD-tag + Sig
  - SK(Kyber512+X25519, Dilithium2+ECDSA)
    - Packet 1: 16 + 832 + 1344[opt1] + 16 + 2484 =3348/4692
    - Packet 2: 800 + 1344[opt2] + 16 + 2484 = 3300/4644
  - SK(Kyber512+X25519, Falcon512+ECDSA)

    - Packet 1: 16 + 832 + 929[opt1] + 16 + 730 = 1594/2523
    - Packet 2: 800 + 929[opt2] + 16 + 730 = 1546/2475
  - SK(Kyber512+X25519, XMSS-SHA2_10_256)
    - Packet 1: 16 + 832 + 64[opt1] + 16 + 2500 = 3364/3428
    - Packet 2: 800 + 64[opt2] + 16 + 2500 = 3316/3380

- Signature-Chain
  - Packet 1: EKEM-pk + Sig-pk + AEAD-tag + Sig
  - Packet 2: EKEM-ct, Sig-pk + AEAD-tag + Sig
  - SC(Kyber512+X25519,WOTS+(32,16))
    - Packet 1: 832 + 32 + 16 + 2144 = 3024
    - Packet 2: 800 + 32 + 16 + 2144 = 2992
  - SC(Kyber768+X25519,WOTS+(32,16))
    - Packet 1: 1216 + 32 + 16 + 2144 = 3408
    - Packet 2: 1120 + 32 + 16 + 2144 = 3312
  - SC(Kyber1024+X25519,WOTS+(32,16))
    - Packet 1: 1600 + 32 + 16 + 2144 = 3792
    - Packet 2: 1600 + 32 + 16 + 2144 = 3792
  - SC(Kyber1024+X25519,WOTS+(64,16))
    - Packet 1: 1600 + 32 + 16 + 8384 = 10032
    - Packet 2: 1600 + 32 + 16 + 8384 = 10032