Post-Quantum Secure Channel Protocols for eSIMs

Design, Validation and Performance Analysis

Luk Bettale[1], Emmanuelle Dottax[2], and Laurent Grémy[2]

[1] IDEMIA Secure Transactions, Courbevoie, France
[2] IDEMIA Secure Transactions, Pessac, France
{luk.bettale,emmanuelle.dottax,laurent.gremy}@idemia.com

Abstract. The transition to Post-Quantum (PQ) cryptography is increasingly mandated by national agencies and organizations, often involving a phase where classical and PQ primitives are combined into hybrid solutions. In this context, existing protocols must be adapted to ensure quantum resistance while maintaining their security goals. These adaptations can significantly impact performance, particularly on embedded devices.

In this article, we focus on standardized protocols which support application management on eSIMs across different modes. This is a complex use-case, involving constrained devices with stringent security requirements. We present PQ adaptations, including both hybrid and fully PQ versions, for all modes. Using ProVerif, we provide automated proofs that verify the security of these PQ variants. Additionally, we analyze the performance impact of implementing PQ protocols on devices, measuring runtime and bandwidth consumption. Our findings highlight the resource overhead associated with achieving post-quantum security for eSIM management.

Keywords: Post-Quantum Cryptography · Secure Channel · Protocol Design · Formal Proof · Embedded Device

1 Introduction

With quantum computing advances, the security of widely used cryptographic systems is increasingly threatened, making it essential to plan for migration to quantum-resistant solutions. With the recent publication of definitive Post-Quantum Cryptography (PQC) standards by NIST, the groundwork is set for such transitions. For some sectors, this transition is more pressing. Industries that rely on long-lived equipment, such as automotive systems, smart meters, and critical infrastructure sensors, often deploy devices that remain operational for many years and rely on connectivity based on an embedded Subscriber Identity Module (eSIM), a digital version of a SIM card embedded directly within the device hardware. Ensuring these devices are secure against quantum-based

attacks is a priority, especially as the cost of upgrading hardware in the field can be significant.

Given this context, the migration of embedded elements and their associated remote management protocols is critical. In this work, we focus on protocols that enable remote management of eSIMs (and, more generally, embedded secure elements), as defined by standards from Global System for Mobile communications Association (GSMA) and GlobalPlatform (GP). Specifically, we investigate a protocol supporting a "scripting" mode, where communications are pregenerated and executed as a sequence, which is useful for deploying commands in environments without a continuous connection. The structure of this protocol presents complexities for post-quantum migration, as seen in other cases such as the Signal protocol [10,32,9,19,4,5,38]. Notably, a Key Encapsulation Mechanism (KEM) does not always seamlessly replace ECDH, requiring adaptations.

This challenge is further amplified by the resource constraints typical of secure embedded elements, including limited memory and processing power, making this an instructive case study in post-quantum migration difficulties. Although the considered protocols originate from a specific use case, they may have potential applications beyond this domain.

In this paper, we first present the background and original versions of the target protocols in Sect. 2. We then propose PQ and hybrid adaptations of these protocols in Sect. 3, designed to retain their original security properties. In Sect. 4, we apply formal verification tools, mainly ProVerif, to validate that these properties are achieved, including in the original, classical cryptography-based protocols. Finally, Sect. 5 examines the performance impact of migrating to PQ cryptographic mechanisms.

2 Protocols for eSIM

2.1 Context

The eSIM is a technological evolution of the traditional SIM card. Unlike physical SIM cards, eSIMs are programmable secure chips soldered directly onto the device's motherboard[3]. A further evolution is the Integrated SIM (iSIM), which is incorporated into the device's processor. Both technologies enable the management of mobile subscriptions and various services without requiring physical intervention.

The GSMA defines the interactions between the various stakeholders involved, including Mobile Network Operators (MNOs), Original Equipment Manufacturers (OEMs), and eSIM providers, across different use-cases. For instance, in the process of downloading a SIM *profile* (when a user acquires a new subscription), the GSMA specifies how the new profile is transferred from the MNO to the eSIM, via the mobile device [28,29].

[3] Strictly speaking, the term eSIM refers to the service, while eUICC (Embedded Universal Integrated Circuit Card) denotes the physical device that operates the mobile functionality. However, these terms are frequently used interchangeably.

Since eSIMs function as secure elements, they can also host other services. For instance, a banking mobile application could benefit from a counterpart on the eSIM for secure operations. In these scenarios, the involved parties differ: indeed, the *service provider*, as owner of the application, enters in the process. A distinct set of documents governs these cases: the *Secured Applications for Mobile* specifications [30] outlines how applications can be installed on eSIMs and managed. A secure link between the eSIM and the entity responsible for installation is established using the *Secure Channel Protocol '11'* (SCP11). This protocol is specified by GP, the organization in charge of standards for digital services that rely on secure elements [26]. This document specifies a secure channel protocol based on Elliptic Curve Cryptography (ECC). We explore it in more detail in the next section.

2.2 Protocol of GP SCP11

SCP11 is a secure channel protocol widely used in various secure element-based products, including but not limited to eSIMs. It employs ECC for mutual authentication and secure channel initiation, and AES for secure messaging. It defines three variants (referred to as *modes*), each tailored to a specific use case. The protocol involves two parties: the Card (e.g., an eSIM) and the Off-Card Entity (OCE, such as a terminal or sever). Table 1 summarizes the security properties claimed in [26], which include authentication of one party to the other, confidentiality and integrity of exchanges, Perfect Forward Secrecy (PFS) and Session Replay (these properties are discussed in greater depth in Sect. 4). The different modes are detailed hereafter.

Table 1. Properties of the different SCP11 modes

Property	Mode A	Mode B	Mode C
Authentication OCE to Card	✔		✔
Authentication Card to OCE	✔	✔	✔
Message Integrity	✔	✔	✔
Data Confidentiality	✔	✔	✔
Perfect Forward Secrecy	✔	✔	
Session Replay			✔

SCP11 Mode A. Mode A establishes mutual authentication between the Card and the OCE; it is illustrated by Fig. 1. Each party holds an ECC key pair—$(\mathsf{EC.sk_{OCE}}, \mathsf{EC.pk_{OCE}})$ for the OCE and $(\mathsf{EC.sk_C}, \mathsf{EC.pk_C})$ for the Card—along with associated certificates, $\mathsf{Cert_{OCE}}$ and $\mathsf{Cert_C}$, signed by an authority CA. These keys enable a key agreement based on ECDH.

The process begins with the OCE retrieving the Card's certificate through a GET_DATA command, then verifying it with the CA's public key $\mathsf{EC.pk_{CA}}$. The OCE then sends its own certificate via a Perform Security Operation (PSO) command, which the Card verifies. Next, the OCE generates an ephemeral key pair $(\mathsf{EC.epk_{OCE}}, \mathsf{EC.esk_{OCE}})$ and sends the public component to the Card. The Card, in turn, creates its own ephemeral key pair $(\mathsf{EC.epk_C}, \mathsf{EC.esk_C})$ and performs two elliptic curve key agreements (EC.KeyAgr, standard ECDH as specified in [11, §4.3]): one using the certified keys and another with the ephemeral keys, ensuring PFS.

The shared secrets, Sss and See, are fed into a key derivation function (KDeriv) to generate a receipt key $\mathsf{SK_{Receipt}}$ and session keys for secure channel $\mathsf{SK_{Session}}$[4]. The KDeriv function uses SHA-256, in compliance with [11]. The message Receipt is a CMAC [21] computed over $\mathsf{EC.epk_{OCE}}$ and $\mathsf{EC.epk_C}$, and sent to the OCE along with the Card's ephemeral key, allowing the OCE to compute and verify the MAC value.

If verified, the OCE and the Card continue with the *Secure Channel Protocol '03'* (SCP03) [25], using $\mathsf{SK_{Session}}$, which includes one encryption/decryption key and two MAC keys, one for each direction. All messages are encrypted and MAC-ed.

It can be noted that the Card cannot authenticate the OCE until it receives an SCP03 command with a valid MAC, as prior messages could come from an attacker without access to $\mathsf{EC.sk_{OCE}}$. Additionally, the session cannot be replayed.

SCP11 Mode B. The Mode B provides authentication of the Card to the OCE only; it is illustrated by Fig. 2. This mode is useful when the OCE lacks certified keys, but can achieve a limited level of authentication through alternative means. For instance, if the OCE is a card terminal, a user-entered PIN verified by the Card [26, Appendix A] offers a weak, indirect form of OCE authentication. However, these considerations are out of the scope of [26], and are not discussed further here.

Mode B is similar to Mode A, with two distinctions:

- The OCE does not send a certificate or use static keys; an INTERNAL_AUTH command replaces the MUTUAL_AUTH one.
- The two key agreements involve the OCE's ephemeral key and both the static and ephemeral keys of the Card.

The rest of the protocol remains the same, ensuring PFS due to the use of ephemeral keys, and anti-replay.

SCP11 Mode C. Mode C is a variant designed for *offline scripting*, enabling the OCE to prepare a sequence of commands in advance and send them to a third-party entity, which will then execute the script on the Card, as shown in

[4] In the SCP11 document [26, Table 6.18], $\mathsf{SK_{Receipt}}$ is the *receipt key* and $\mathsf{SK_{Session}}$ encompasses the keys called S-ENC, S-MAC, S-RMAC and S-DEK.

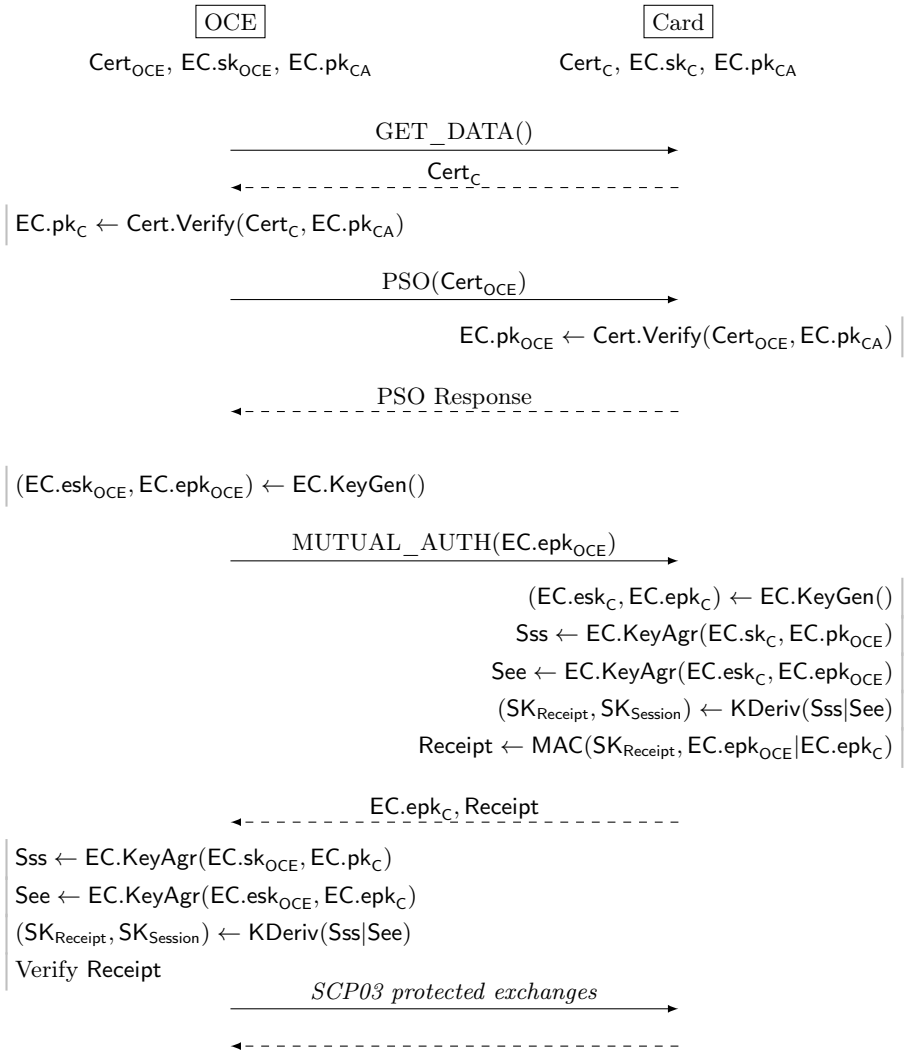

Fig. 1. SCP11 Mode A

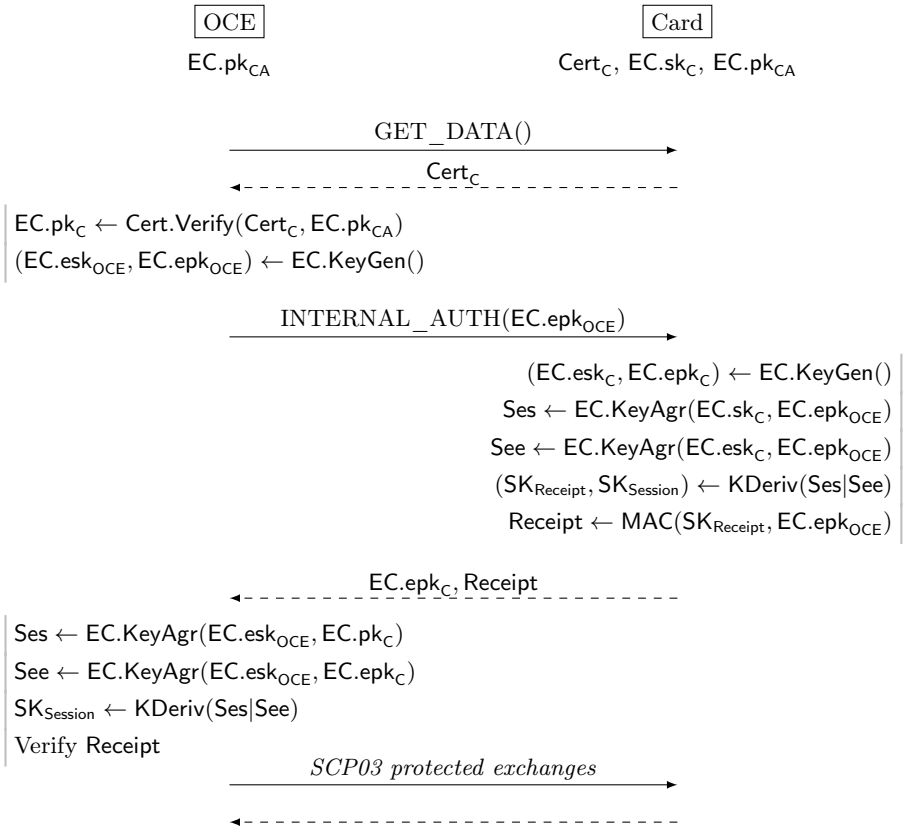

Fig. 2. SCP11 Mode B

Fig. 3. Typically, this mode assumes the Card is an eSIM in a mobile device, which plays the script. For instance, the script might be distributed as part of a rich environment application (e.g., an Android or iOS application) and executed during the application installation. Importantly, no secrets are handled by the mobile application, enabling the safe installation of services on the eSIM to secure functions such as those of a banking application. This mode also allows for remote administration of secure elements in general, without requiring a continuous, direct connection, which may be impractical in certain scenarios.

Mode C resembles Mode A with the following modifications:

- The OCE retrieves the eSIM's public key from a database.
- Only the eSIM's static keys are used, as the *offline* nature of this mode precludes the use of ephemeral data on eSIM side.

Mutual authentication, key derivation and command wrapping are performed by the OCE in advance. The $\mathsf{Receipt}$ is based on $\mathsf{EC.epk_{OCE}}$ and $\mathsf{EC.pk_C}$.

Mode C provides mutual authentication between the OCE and the eSIM but lacks PFS for the eSIM's secrets. Furthermore, as the session relies only on static data from the eSIM, it can be replayed, unlike in Modes A and B. This is acknowledged by [26], and as a consequence some sensitive commands—like inserting new keys—are disallowed in Mode C.

2.3 Migration towards Quantum-Resistant Versions

The protocols currently in use rely extensively on ECC, which lacks resistance to quantum attacks. This vulnerability requires migration to quantum-resistant alternatives, particularly for two reasons. First, any data exchanged today may be decrypted in the future, creating a significant risk if long-term sensitive data are transmitted. Second, eSIMs are often embedded in devices intended for extended deployment in the field (like vehicles or smart-meters), making it essential to equip them as soon as possible with quantum-resistant protocols to ensure their long-term security.

In the following section, we address the migration of these existing protocols. As we will discuss, this process involves certain complexities and requires dedicated adaptation efforts.

3 Quantum-Resistant Versions

3.1 Quantum-Resistant Protocol for Mode C

We begin with an analysis of Mode C, the variant used for scripting. In this mode, as previously described, the off-card entity OCE prepares a script in advance for the eSIM. The protocol achieves mutual authentication, ensuring that only the intended eSIM, with the appropriate ECDH key, can read the script, while the eSIM is assured that the script originates from an authorized OCE. Our goal is to preserve these properties against a quantum-capable adversary.

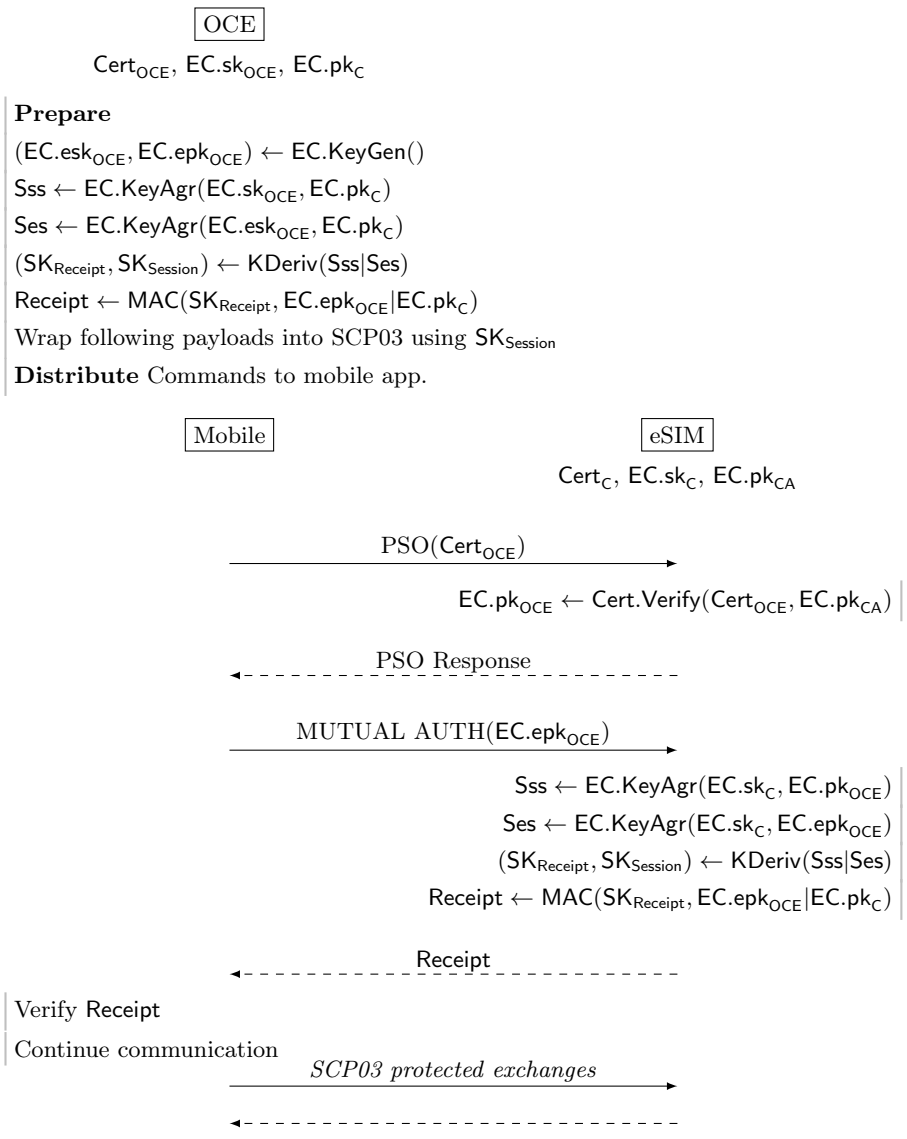

Fig. 3. SCP11 Mode C

A first set of standardized post-quantum algorithms is available, covering both signature [41,43] and Key Encapsulation Mechanism (KEM) [42]. Since the original protocol relies on ECDH, a natural approach is to use a KEM to construct a quantum-resistant version. Assuming the OCE has the eSIM's long-term KEM public key (similar to the original protocol using the eSIM's ECDH key), it can prepare a script for the eSIM. However, using only KEM keys does not allow the OCE to prove that it authored the script, as an interaction would be required for the eSIM to produce a ciphertext for the OCE. This limitation is well-documented in similar adaptations, such as PQ versions of Signal (see references in Sect. 1): while KEMs work for key establishment, they cannot replace ECDH in protocols requiring asynchronous authentication.

To address this, the OCE needs a signature key to enable the eSIM to verify the script's origin. Replacing the eSIM's long-term KEM key with a signature key is not feasible here, as it would again require interaction with the eSIM, allowing it to sign an ephemeral KEM key.

Thus, we construct a protocol with a signature key on the OCE side, and a KEM key on the eSIM side. To the best of our knowledge, this is the first approach of its kind to leverage such a combination. The protocol is illustrated by Fig. 4. The OCE is now equipped with a signature key $\mathsf{SIG.sk_{OCE}}$ and the corresponding certificate $\mathsf{Cert_{OCE}}$. As in the classic case, the OCE already knows the eSIM's public key, which is now a KEM key $\mathsf{KEM.pk_C}$. The OCE begins by performing an encapsulation on this key to produce a shared secret Ss and its ciphertext $\mathsf{c.Ss}$, which are used to derive the secure channel and receipt keys. The ciphertext is signed, to authenticate the origin of the script. The script is sent to the mobile device and played on the eSIM. The eSIM verifies the OCE's certificate, checks the signature, and decapsulates the ciphertext to obtain the shared secret and derive the session keys. The remainder of the process aligns with the classic protocol.

Regarding PFS, the situation is the same as in the classical case: it is not ensured on the eSIM's side (recovering $\mathsf{KEM.sk_C}$ allows to decrypt past communications), but it is on the OCE's one (the shared secret Ss is ephemeral). Similarly, replay is possible.

3.2 Quantum-Resistant Protocols for Other Modes

For completeness and coherence, we explore how PQ versions of Modes A and B can be constructed using the same credentials as Mode C: a signature key for the OCE and a KEM key for the eSIM. While these protocols could certainly rely exclusively on either signature keys or KEM keys, using the same functions as mode C would streamline the process. This approach removes the need for additional credentials and avoids introducing extra functionalities on the eSIM.

For Mode A, we want to maintain the original mutual authentication and PFS properties. This can be achieved following the process illustrated by Fig. 5. Compared to the classic protocol, we need one more command, that we call MUTUAL_AUTHENTICATE2. This is again due to the usage of a KEM in place of an ECDH key agreement.

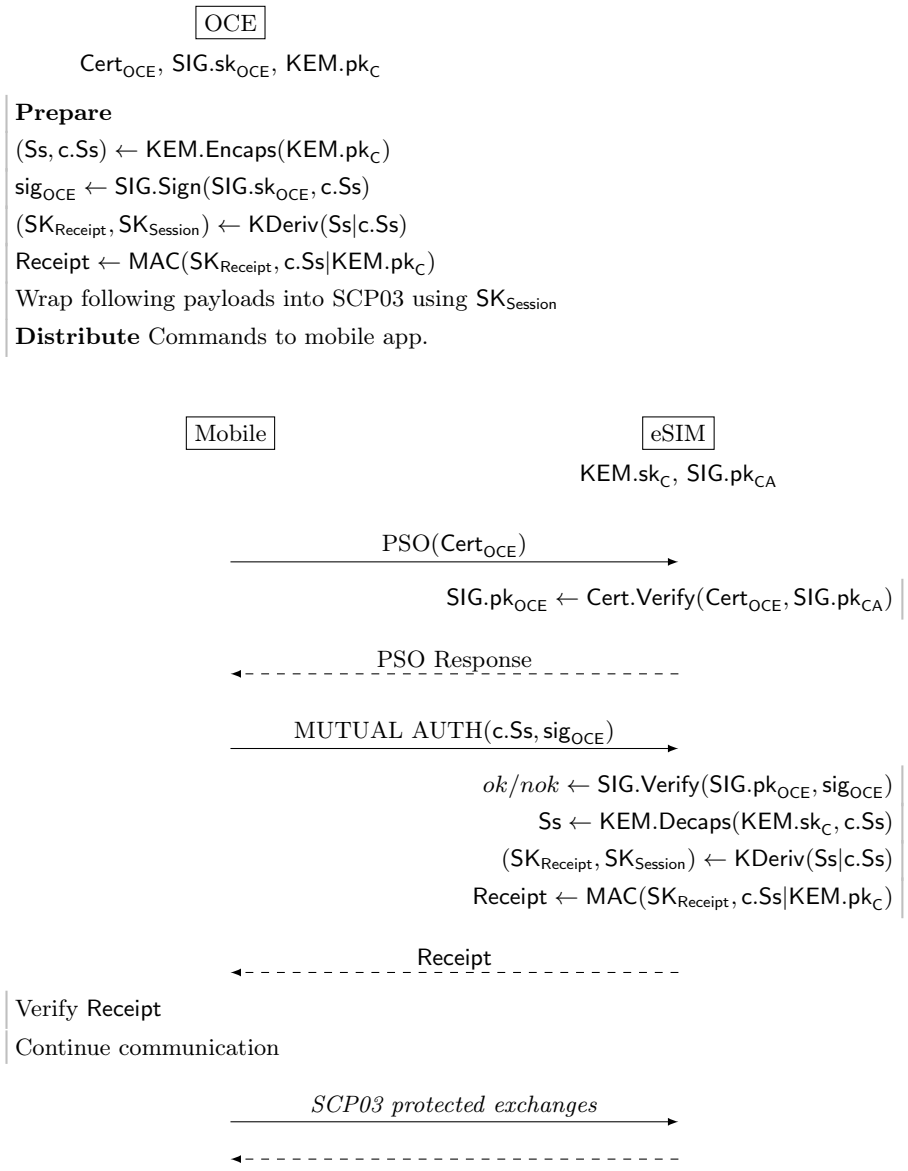

Fig. 4. PQ version for Mode C

The beginning of the protocol is the same: the Card and the OCE exchange their certificates, verify them using $\mathsf{SIG.pk_{CA}}$ and extract the public keys $\mathsf{SIG.pk_{OCE}}$ and $\mathsf{KEM.pk_C}$. Of course, the certificates are signed with a PQ signature algorithm. The first MUTUAL_AUTHENTICATE command is used to make the eSIM generate a KEM ephemeral key and send the public part to the OCE. The OCE can then use this ephemeral key and the static one to generate two shared secrets Se and Ss, and their respective ciphertexts $\mathsf{c.Se}$ and $\mathsf{c.Ss}$. The OCE signs these ciphertexts, together with the eSIM's ephemeral key. The resulting signature and the ciphertexts are sent to the eSIM via the new command. The eSIM can then verify the signature, decapsulate both ciphertexts and derive secret material from the concatenation of shared secrets and ciphertexts (as advised by [22]). The rest of the protocol is unchanged: the eSIM computes an authentication value $\mathsf{Receipt}$, and both parties use the session keys $\mathsf{SK_{Session}}$ for further exchanges through SCP03.

Note that the ephemeral key could be generated by the OCE, this is equivalent for the PFS (assuming both parties correctly manage ephemeral values), and both options need two commands for the mutual authentication.

The PQ version of Mode B is similar to the one of Mode A, with the difference that the OCE does not send any certificate, nor perform any signature. It is illustrated by Fig. 6.

3.3 Hybrid Versions

Hybrid versions of protocols can be constructed by combining classical algorithms with PQ ones to address the limited maturity of PQ algorithms. This approach is recommended by several European cybersecurity agencies [1,12] and will be accommodated by NIST [40].

Several options are available for constructing hybrid protocols. ETSI provides two key agreement constructions in its technical specification [22]:

- *Concatenate* hybrid key agreement, where both cryptographic methods are combined in single messages by concatenating their respective data.
- *Cascade* hybrid key agreement, where the first key agreement is executed before the second.

With the first option, each command conveys more data, while the second one augments the number of commands. We opted for the first option—the second one can easily be derived.

Figure 7 illustrates the hybrid protocol for Mode A. In this set-up, the OCE and the eSIM use hybrid certificates, denoted $\mathsf{Cert^H_{OCE}}$ and $\mathsf{Cert^H_C}$. Multiple formats for hybrid certificates exist (see [27] for an overview), but all include both classical and PQ keys and are signed by the classical and PQ keys of the CA, denoted $\mathsf{pk^H_{CA}}$. The $\mathsf{Cert.Verify}$ function returns both classical and PQ public keys: for the OCE, $\mathsf{EC.pk_{OCE}}$ and $\mathsf{SIG.pk_{OCE}}$; for the eSIM, $\mathsf{EC.pk_C}$ and $\mathsf{KEM.pk_C}$.

The hybrid protocols incorporate both classical and PQ computations within the same number of commands as the PQ-only version. Shared secrets are derived

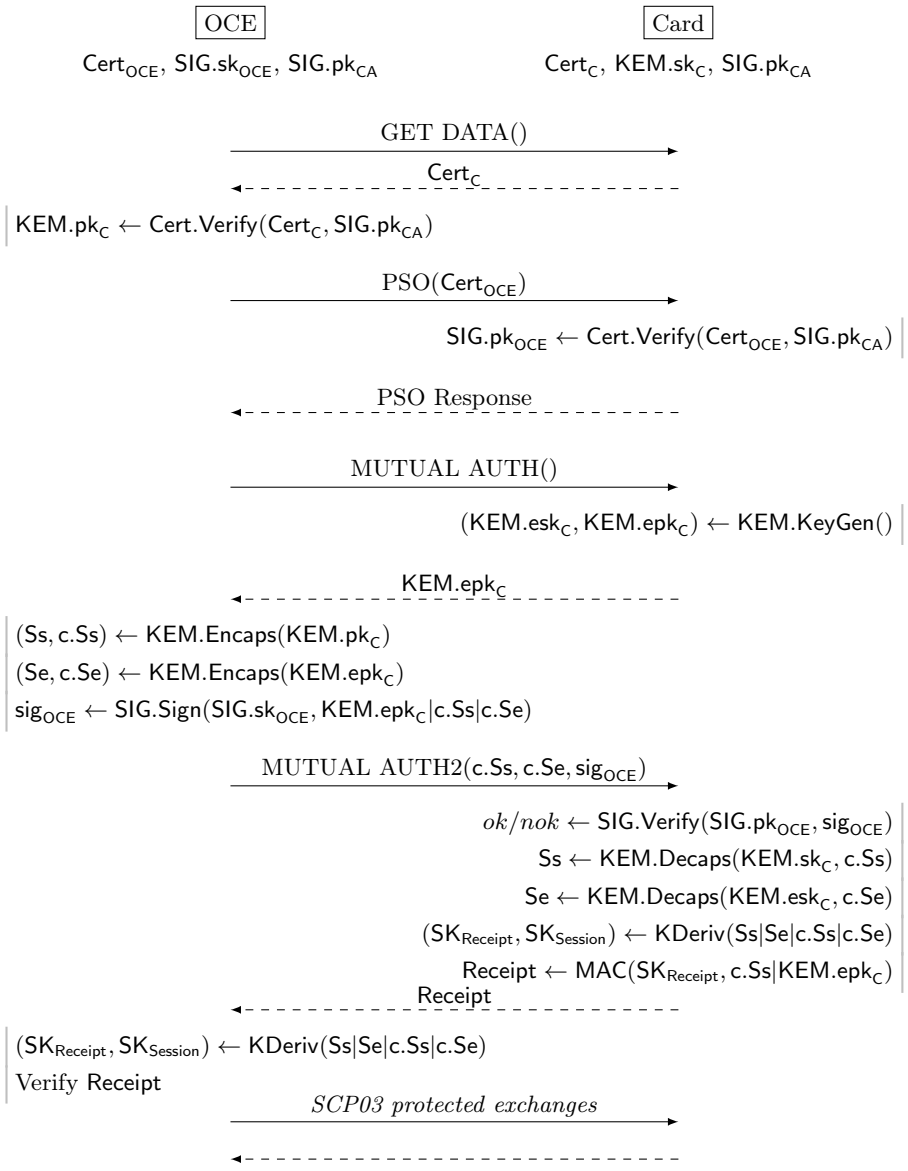

Fig. 5. PQ version for Mode A

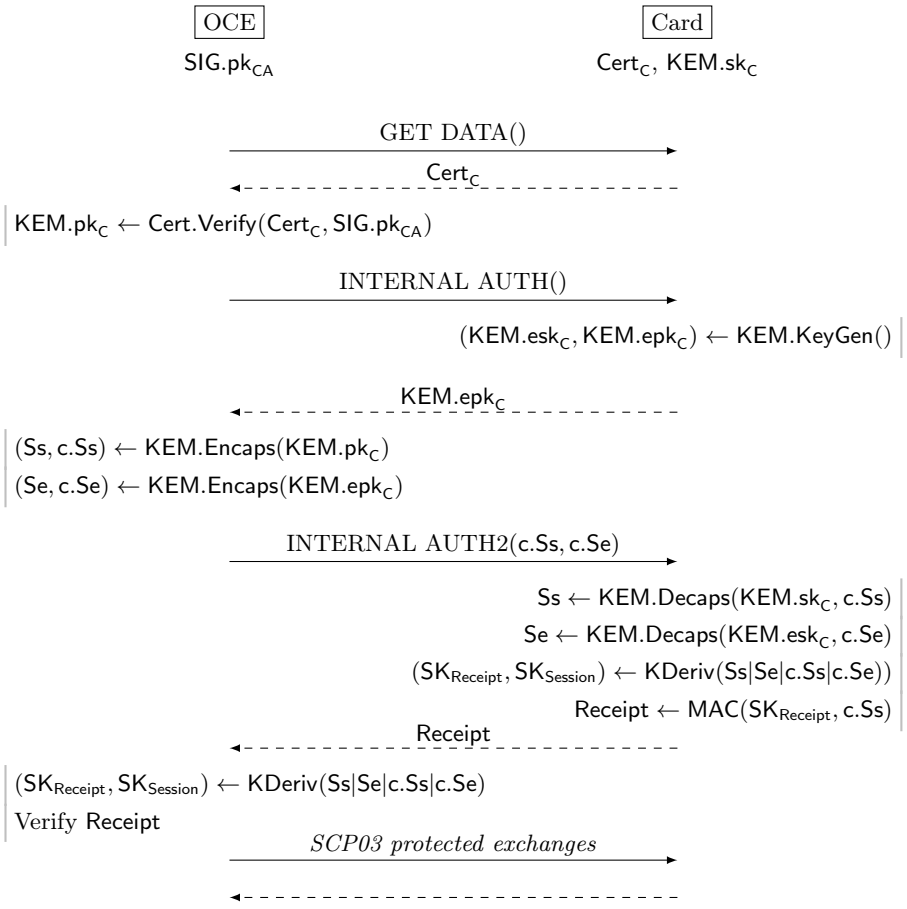

Fig. 6. PQ version for Mode B

from both classical and PQ key agreements and are carefully combined using the KDeriv function, in line with [22] recommendations.

Hybrid versions of Modes B and C are based on these same principles and are detailed in Annex A.

4 Formal Proofs

The security properties of the different modes of SCP11 are listed in Sect. 2.2 and summarized in Table 1. However, no formal security proof exists in the literature confirming that these properties are achieved. We therefore propose a formal verification of the different modes of SCP11, and then consider the quantum-resistant and hybrid variants.

4.1 Symbolic Formal Verification of SCP11 Protocols

Our proofs are conducted in the formal model, also known as the Dolev–Yao attacker model [20] or symbolic model. In this setting, the protocol participants communicate over a public channel and employ *perfect* cryptographic primitives, where, for instance, decryption is only possible with the corresponding key. An active attacker in this model has full control over the public channel, enabling her to read, drop, modify, replay messages, and inject new messages. Additionally, the attacker may selectively compromise participants (e.g., through long term key leaks), or gain access to oracles that break cryptographic assumptions (e.g., retrieving a private key from a public key). In this setting, we can model a protocol and determine wether certain (mathematical) properties hold despite such an adversary. For SCP11, we designate the OCE, the Card and the attacker as $\mathcal{O}$, $\mathcal{C}$ and $\mathcal{A}$, respectively.

Authentication. Formalizing cryptographic properties into precise mathematical formulations can be challenging, as illustrated by the various definitions of authentication [39]. Interested readers can find a near-mathematical formalization of these properties in [46, pp.82–83]. We aim to prove the two strongest forms of agreement for classical SCP11: the *non-injective agreement* [39, §2.3] and *injective agreement* [39, §2.4]. The authentication of $\mathcal{C}$ to $\mathcal{O}$ may represent an injective agreement upon Receipt. The authentication of $\mathcal{O}$ to $\mathcal{C}$, which is unattainable in Mode B, may represent an injective agreement on the SCP commands in Mode A, but a non-injective agreement in Mode C, since $\mathcal{O}$ may replay a previous session.

Message Integrity. Message integrity is required during SCP03 exchanges. For any command a sent by $\mathcal{O}$ and any command b received by $\mathcal{C}$, integrity is achieved if a and b are identical. However, this condition may not hold for the Mode B, where any $\mathcal{A}$ may impersonate an $\mathcal{O}$. Message integrity is verified across all modes if, for any command a whose authenticated encryption e is sent by $\mathcal{O}$ and for any command b decrypted after the reception of e by $\mathcal{C}$, the commands a and b are the same.

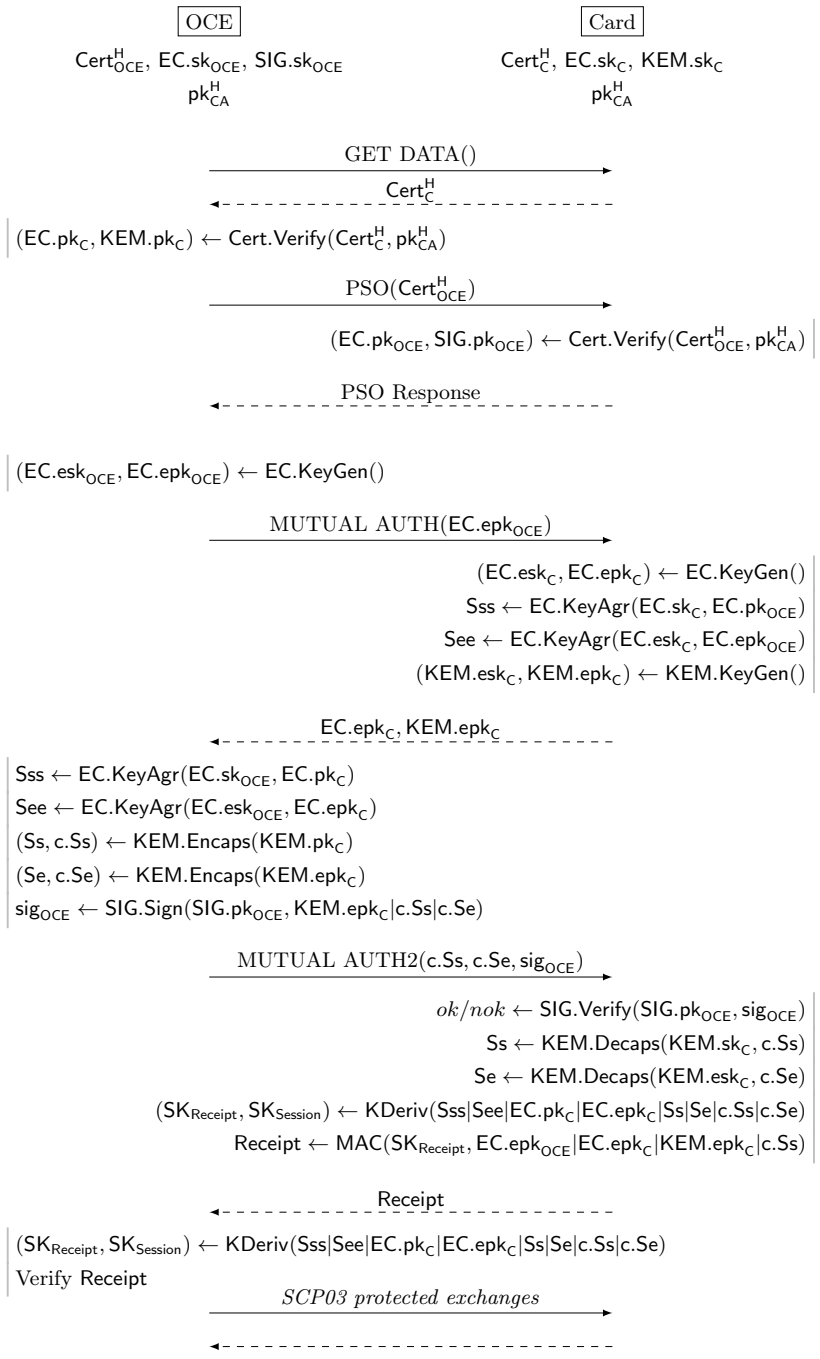

Fig. 7. Hybrid version for Mode A

Data Confidentiality and PFS. Like message integrity, data confidentiality applies only to SCP03 exchanges. A command is confidential if $\mathcal{A}$ cannot access it at any point. Proving PFS adds a temporal condition: we assume $\mathcal{A}$ obtains the long-term secret key of $\mathcal{C}$ or $\mathcal{O}$ after the protocol's completion. A protocol is perfect forward secure [31,18] if $\mathcal{A}$ cannot compute the symmetric keys, such as $\mathsf{SK}_{\mathsf{Session}}$, exchanged before impersonating $\mathcal{C}$ or $\mathcal{O}$.

Session Replay. The session replay is addressed by the (non-)injection agreement discussed above. To validate the negation of this property, session uniqueness, we need to confirm that if two tuples of keys ($\mathsf{SK}_{\mathsf{Receipt}}$, $\mathsf{SK}_{\mathsf{Session}}$) established by $\mathcal{C}$ and $\mathcal{O}$ are identical at two different time periods, those periods must be the same.

To be automatically proved, all these (mathematical) properties need to be encoded in the formalism of software programs designed for the formal verification of cryptographic protocols.

4.2 Security Protocol Analysis Tools

The two leading tools for formally proving protocols in the symbolic model are ProVerif [6] and Tamarin [45]. A comprehensive comparison of these tools, along with others, can be found in [2, §II.A-C]. Two newer software options may also be considered: Verifpal [37] and PQ-Squirrel [17]. Verifpal is designed as a user-friendly tool with a limited, though sufficient for our case, number of cryptographic primitives and properties, while PQ-Squirrel is in between a symbolic and a computational tool. All these tools generally involve two stages: the first models the protocol, and the second assesses the security properties, expressed as (mathematical) queries.

In this work, we chose to use ProVerif (version 2.05) as our primary tool, supplemented with Verifpal (version 2.72) for its simplicity. Notably, Verifpal models can be exported to ProVerif models, a feature we anticipated but could not use. In the remainder of this section, we will describe how we model the classical SCP11 modes outlined in Sect. 2.2. Despite their differences, both ProVerif and Verifpal share a crucial concept in the query stage: the notion of *phase*. This concept is essential for proving PFS in our context, as detailed in [36, §2.5] and [8, §4.1.6]. The complete execution of the protocol is considered phase 0, followed by phase 1, which necessarily occurs after the completion of phase 0, during which the long term keys are leaked. The proof of PFS is done for phase 1.

The Verifpal Prover. This section provides a brief overview of our Verifpal model; for detailed information, refer to the documentation [36, §2-3].

Model. A Verifpal model starts by characterizing the attacker, *active* in our case to match the Dolev–Yao attacker described in Sect. 4.1. The two actors then perform a sequence of actions using predefined cryptographic primitives, organized within blocks defining a specific *principal*, interleaved with the messages

exchanged between the principals. Principals with the same name share a global state, meaning that elements within a block are accessible to all subsequent blocks of that principal. The CA is replaced by public key exchanges which are *guarded*, i.e., public exchanges the attacker cannot modify.

Queries. At the end, we formulate *queries* to assess the protocol's security goals. In our context, these queries focus on *authentication*, more precisely the non-injective agreement on certain messages exchanged between the two principals, as well as *confidentiality* of data and *freshness* of some keys to evaluate session replay. Message integrity is checked using an `ASSERT` directly in the model step.

Currently, we cannot export our Verifpal models to ProVerif, since the export feature does not support freshness queries and the different phases of the protocol. We therefore build our ProVerif models from scratch.

The ProVerif Prover. In this section, we discuss our modeling choices in the ProVerif language. For comprehensive details, refer to the ProVerif manual [8].

Model. Unlike Verifpal, ProVerif requires explicit definitions of the cryptographic primitives used in the protocol. These definitions resemble an API description, accompanied by the mathematical equations each primitive verifies [8, §4.2]. For instance, a message authentication code (MAC) is illustrated in Listing 1. To compute a MAC, the function Mac takes a key k and a message m to produce the MAC c of m. To verify if c is the MAC of m under the key k, the function MacVerify takes the key k, the message m and the alleged MAC c and outputs *true* if $c = \mathrm{Mac}(k, m)$ (i.e., $\mathrm{MacVerify}(k, m, \mathrm{Mac}(k, m)) = 1$) and *false* otherwise (i.e., $\mathrm{MacVerify}(k, m, c \neq \mathrm{Mac}(k, m)) = 0$).

Listing 1. ProVerif model of the MAC computation and verification

```
type mackey. type macs. fun mac(mackey, bitstring): macs.
fun macverify(mackey, bitstring, macs): bool
reduc forall k: mackey, m: bitstring;
  macverify(k, m, mac(k, m)) = true
otherwise forall k: mackey, m: bitstring, c: macs;
  macverify(k, m, c) = false.
```

The primitives are embedded within two *process macros* representing the Card and the OCE. Unlike Verifpal's export to ProVerif, which relies on state sharing similar to Tamarin, we define a single process macro for each actor. The two actors communicate over an open channel, vulnerable to a Dolev–Yao attacker (see Sect. 4.1). Each process macro is initialized in the main process with the public and private keys required for the chosen mode, with public keys also broadcasted over the public channel, making them accessible to the attacker. Additionally, the certificate authority is modeled as a separate macro process that signs trusted public keys stored in an immutable table.

Queries. To formalize queries in the ProVerif model, we first create *events*, although not all queries require events. An example lies in Listing 2, where we define a command and query wether the attacker obtains this command.

Listing 2. ProVerif model of a confidentiality query

```
free commands: bitstring [private].
query attacker(commands).
```

An *event* denotes a significant sequence of actions. For instance, the authentication of the Card to the OCE via injective agreement on the receipt message is briefly formalized in Listing 3. The issuance and sending of the receipt message by the Card to the OCE is the goal of `sendReceipt`, which takes as arguments the receipt message and the two actors, first the sender and second the receiver. To prove the authentication query, we must define another event to confirm that the MAC verification is correctly performed: this is the goal of `acceptReceipt`, which takes as arguments the receipt message and the two actors, first the receiver and second the issuer. Once events are defined and placed correctly in the model, we use the injective correspondence with the `inj-event` keyword.

Listing 3. ProVerif model of the two process macros

```
...
event sendReceipt(macs, host, host).
event acceptReceipt(macs, host, host).
...
query receipt: macs, hostOCE: host, hostCard: host,
  i: time, j: time;
  event(acceptReceipt(receipt, hostOCE, hostCard))@i ==>
  inj-event(sendReceipt(receipt, hostCard, hostOCE))@j &&
  j < i.
...
let pOCE(pkS: signCApub, SKOCEECKA: eckapriv) =
  ...
  in(c, (ePKSDECKA: eckapub, receipt: macs));
  ...
  let (rkey: mackey, senc: symkey, smac: mackey) =
    kdf((ShSss, ShSee)) in
  if macverify(rkey, (ePKOCEECKA, ePKSDECKA), receipt) then (
    event acceptReceipt(receipt, OCE, Card);
    ...
  ).
let pCard(pkS: signCApub, SKSDECKA: eckapriv) =
  ...
  let (rkey: mackey, senc: symkey, smac: mackey) =
    kdf((ShSss, ShSee)) in
  let receipt = mac(rkey, (ePKOCEECKA, ePKSDECKA)) in
  event sendReceipt(receipt, Card, OCE);
  out(c, (ePKSDECKA, receipt));
  ...
```

Regarding the integrity query in Listing 4, events relate to the (encrypted) commands sent over the SCP03 channel. The query states that if an actor sends a command a through an encrypted message e with an authentication tag m, and another actor receives a command b from the decryption of message e with tag m, then a and b are the same.

Listing 4. ProVerif model of an integrity query

```
event sendCom(bitstring, bitstring, macs).
event readCom(bitstring, bitstring, macs).
query a: bitstring, b: bitstring, e: bitstring, m: macs;
  event(sendCom(a, e, m)) && event(readCom(b, e, m)) ==>
  a = b.
```

Finally, Listing 5 formalizes the session replay query. After the Card derives the key, the event `SCP03SK` is triggered. This property states that if the three keys established by the Card with the (believed) OCE are identical for different time periods i and j, then i and j must match, i.e., $i = j$.

Listing 5. ProVerif model of a replay query

```
event SCP03SK(mackey, symkey, mackey, host, host).
query rkey: mackey, senc: symkey, smac: mackey,
  i: time, j: time;
  event(SCP03SK(rkey, senc, smac, Card, OCE))@i &&
  event(SCP03SK(rkey, senc, smac, Card, OCE))@j ==> i = j.
```

4.3 Verification of the Models

We model all three modes in both tools. In our ProVerif models, we include a sanity check to verify that at least one trace allows the Card to receive the commands sent by the OCE. Without this check, a query may be considered as verified only because the event(s) of the query are not reached; for instance, if the first event is not reached, both true and false can be assumed for the second event, see Listing 3. We summarize the results of our proofs for the different modes in both tools in Table 2, which can be compared to Table 1. All properties are verified with both tools, with semantic differences discussed in Sect. 4.2. In addition, we present the runtime for each tool when verifying the different models: ProVerif performs significantly faster than Verifpal on a personal computer.

4.4 Hybrid and Post-Quantum Protocols

With the ProVerif models established for the classical modes of SCP11, we now aim to prove that the quantum-resistant versions introduced in Sect. 3 achieve the same properties as their classical counterparts in the symbolic model.

Table 2. Verification of the different SCP11 modes.

Property	Verifpal			ProVerif		
	Mode A	Mode B	Mode C	Mode A	Mode B	Mode C
Authent. OCE to Card	✔		✔	✔		✔
Authent. Card to OCE	✔	✔	✔	✔	✔	✔
Message Integrity	✔	✔	✔	✔	✔	✔
Data Confidentiality	✔	✔	✔	✔	✔	✔
Perfect Forward Secrecy	✔	✔		✔	✔	
Session Replay			✔			✔
Time (rounded)	11 h	2.5 h	50 m	3 s	3 s	2 s

Previous Works on Symbolic Proofs for Post-Quantum Protocols. Unlike computational tools [3,7], symbolic tools typically do not require adaption to take into account quantum adversaries. Consequently, we can utilize ProVerif directly, given an appropriate model for the new primitives introduced in the post-quantum variants. Several studies have already explored automated proofs in the symbolic model for post-quantum versions of various protocols, as summarized in Table 3.

Table 3. Post-quantum protocols with security proofs in the formal model with an automated tool.

(Sub)Protocol	Article	ProVerif	Tamarin	Verifpal	PQ-Squirrel
OPC UA	[44]	✔		✔	
PQ-Wireguard	[33]		✔		
PQ IKEv2	[24]		✔		
KEMTLS	[13]		✔		
PQ IKEv1	[17]				✔
PQ IKEv2	[17]				✔
PQ X3DH	[17]				✔
PQ Signal	[4]		✔		
PQXDH	[5]	✔			
iMessage PQ3	[38]		✔		
PQ SCP11	This work	✔			

Modeling PQ SCP11. The main distinction from classical protocols is the introduction of a KEM algorithm. Following the approach in [5], the generic KEM encapsulation is modeled as the generation of a secret value m, which is asymmetrically encrypted with the recipient's public key, and m serves as the shared secret. The ML-KEM standard [42] uses a shared secret which can be modeled as (the hash of) the random m concatenated with the public key, as illustrated in Listing 6. This approach differs from a generic KEM.

Listing 6. ProVerif model of a simplified version of ML-KEM

```
type kempriv. type kempub. fun kempk(kempriv): kempub.
fun h(bitstring, kempub): bitstring.
fun pkeenc(kempub, bitstring): bitstring.
fun pkedec(kempriv, bitstring): bitstring
reduc forall sk: kempriv, m: bitstring;
  pkedec(sk, pkeenc(kempk(sk), m)) = m.
letfun kemenc(pk: kempub) = new m: bitstring;
  let k = h(m, pk) in (pkeenc(pk, m), k).
letfun kemdec(sk: kempriv, ct: bitstring) =
  let m = pkedec(sk, ct) in h(m, kempk(sk)).
```

To avoid potential public key confusion attacks [5, §4.1.1], we incorporate an identifier into the signature provided by the certificate authority, facilitating domain separation between the public keys. We model the PQ signature algorithm as the standard signature model [35, §2.2].

Quantum-only versions. The events and queries from the classical models can be directly reused in the models for the PQ SCP11 versions. Our models for the PQ modes validate the expected proofs outlined for the classical versions, as summarized in Table 1. Notably, the PQ versions of Modes A and C allow to reach the authentication of the OCE to the Card sooner than in the classical version through the use of signatures. More precisely, the sequence involving the issuance of a signature by the OCE followed by its verification by the Card guarantees *aliveness* of the OCE [39, §2.1] in the Mode C. This same sequence guarantees to the Card *agreement* [39, §2.4] with the OCE on the signature. However, in Mode A, the omission of the ephemeral key in the signed the message reduces the authentication to *aliveness*, akin to Mode C. This reduction may be acceptable if the PQ protocol aims to align with the security properties achieved in the classical protocol.

During the modeling stage of the PQ protocols, we model firstly a simple rough key derivation step, by only considering the concatenation of the shared secret *without* taking into account the ciphertexts. This is in contrast with what is described for example in [22, §8]. While assessing the properties related to SCP11, we found that our initial (incorrect) model did not significantly diverge from expectations, except for the replay property in Mode B. In this case, since the OCE is not authenticated, an attacker could send the same shared secret, for two runs of the protocol. Interestingly, using ML-KEM [42] defeats this attack since the public key used for the encapsulation is embedded into the shared

secret construction, but this cannot be considered as a perennial solution. By aligning our model on the protocol described in Fig. 6, the attack is defeated for a generic KEM.

Hybrid versions. Hybrid protocols are of first importance today to prepare the migration between pure-classical and pure-post-quantum protocols. In addition to ensuring that an SCP11 mode provides the requisite security when its primitives are secure, hybrid protocols must also remain secure if either of the combined primitives—classical or else PQ—is compromised. To model this, we introduce oracles for the attacker via ProVerif *process macros*, allowing her to submit public keys and obtain the associated secret keys, but only for either PQ or classical primitives, as shown in Listing 7.

Listing 7. ProVerif model of the oracle which breaks the public key of a KEM

```
free att: channel. (* A channel for the attacker *)
fun recoverKEMpriv(kempub): kempriv
reduc forall sk: kempriv;
  recoverKEMpriv(kempk(sk)) = sk [private].
let kem_attacks() =
  in(att, pk: kempub); out(att, recoverKEMpriv(pk)).
```

We can verify that the hybrid protocols ensure the same security properties as the classical protocols when PQ primitives are broken, and conversely the same security as PQ protocols when the classical primitives are compromised. Note that, since the signatures issued by the certificate authority are also hybrid, we model it as a single unbroken signature algorithm rather than a combination of two signature algorithms—one compromised and one secure.

5 Instantiation and Performance Analysis

When implementing cryptographic protocols in embedded devices, a lot of constraints come into play. First, the amount of available resources (RAM, CPU frequency) is usually very low. Implementing post-quantum algorithms can be challenging. This is even more the case as implementations on embedded devices have to be resistant against physical attacks. The cost of securing an implementation against side-channel and fault attacks can drastically increase the required amount of RAM, as well as the execution time. On top of that, protocols are designed for two parties to communicate with each other. The amount of data exchanged by the parties would directly impact the overall performance of the protocols.

In this section, we study these metrics for an implementation on a typical embedded device (32-bit Cortex-M3 CPU at 100 MHz) with a hardware accelerator for ECC. The chip supports a baud-rate of 600 Mbit/s. We discuss the impact of the choice of the PQ algorithm on the execution time of a complete protocol execution.

5.1 Implementation

For our experiments, we naturally chose to implement algorithms that were standardized by NIST, namely ML-KEM [42] for key encapsulation, and ML-DSA [41], SLH-DSA [43] and FN-DSA for signature. As the FN-DSA specifications are not yet available, we implemented the latest version submitted to the NIST competition, that is the Round 3 version of Falcon [23]. In the rest of the paper, we will use the name FN-DSA. The ML-KEM primitives were implemented with side-channel countermeasures such as [15,16].

To align the security levels with the post-quantum algorithms, we implemented the ECC component with three distinct security levels: 128, 192, and 256 bits, corresponding to NIST categories 1, 3, and 5, respectively. We used the `P-256`, `P-384` and `P-521` Elliptic Curve domain parameters as specified in SP800-186 [14]. Since we simulated the OCE using a desktop computer, the timing results for the OCE are not relevant. However, the OCE typically has significantly greater processing power than the chip, making its timing negligible in this context.

5.2 Performance Analysis

In Table 4, we present the communication and processing time of the chip, when classic SCP11 using ECC is used. Note that we measure only the secure channel establishment part, before any SCP03 exchanges. As the communication time may depend on the negotiated baud-rate and transmission protocol, it is clearly separated from the processing time of the chip. This communication time includes both the sending and receiving of data by the chip.

In Tables 5, 6 and 7, we give the same figures for the hybrid version using the three standardized post-quantum signatures, respectively ML-DSA, SLH-DSA and FN-DSA. For SLH-DSA, we chose the *small* variants, as our context only requires verification. This variant provides the shortest signatures and the fastest verification times, though it does come with a much slower signing process. As the signatures are generated by the OCE and can even be done completely offline in Mode C, this option is optimal for our protocols. Additionally, the tables include the ratio with respect to the classic version to see the overhead of hybridation.

The first observation from Table 6 is that even with the small variant of SLH-DSA, the overall performances are very slow, with processing time reaching up to 5 seconds whenever a verification is required. Also the larger signatures significantly impact the communication time, with a factor as high as 155 for Mode C at security level 256. Because SLH-DSA is a hash-based signature, it could be used on its own, without ECDSA according to [12,1]. However, removing the ECDSA verification would not lead to substantial improvement; the timings would remain on the same order of magnitude.

For ML-DSA in Table 5, it is worth noticing that while the impact on the processing time is non-negligible, it is not more than a factor of 8 at worst. However, the communication time is more than 40 times slower. Despite this, the total time remains less than one second for security level 128.

The results for FN-DSA in Table 7 are much better due to the relatively small signature size and the efficient verification process. At security level 128, all modes run in less than 500 ms. In a context where the constrained device only performs verifications, it is advantageous to choose a signature algorithm with a short signature and fast verification, such as FN-DSA.

Table 4. Chip-based measurements of classic (ECC) SCP11 protocols execution.

Protocol	Sec. level	Communication (ms)	Processing (ms)	Total (ms)
Mode A	128	8	67	75
	192	12	130	142
	256	16	225	241
Mode B	128	6	48	54
	192	8	93	101
	256	11	158	169
Mode C	128	4	49	53
	192	6	97	103
	256	8	169	177

6 Conclusion

Through our exploration of the SCP11 protocol's different modes, we identify challenges in the transition from classical to post-quantum cryptography. Beyond proving that the original SCP11 protocol achieves its claimed security properties, we also designed new PQ-only and hybrid protocols that uphold the same security guarantees as the classical modes. Our formal proofs validate that these PQ and hybrid designs meet the rigorous requirements for secure deployment in embedded environments.

By leveraging a range of standardized PQ cryptographic primitives, we measure the impact of these post-quantum adaptations when integrated into embedded systems, and show that an algorithm with short signatures and fast verification like FN-DSA is most suited.

To further optimize PQ protocol performance, chip manufacturers may incorporate specialized accelerators, increase clock speeds, or expand memory, which would notably benefit embedded use cases. Improving communication performance presents greater challenges, requiring coordination among stakeholders like mobile and chip manufacturers. As this process will take time, performance gains in communication may be slower to realize.

Table 5. Chip-based measurements of hybrid SCP11 protocols execution, with ratio relative to classic versions, using ML-DSA signature.

Protocol	Sec. level	Communication		Processing	Total
		(ms)	(ratio)	(ms) (ratio)	(ms) (ratio)
Mode A	128	267	(×35)	516 (×7.7)	783 (×11)
	192	380	(×33)	796 (×6.1)	1176 (×8.3)
	256	515	(×32)	1184 (×5.3)	1698 (×7.1)
Mode B	128	117	(×23)	290 (×6.0)	406 (×7.6)
	192	165	(×21)	427 (×4.6)	591 (×5.9)
	256	228	(×22)	629 (×4.0)	857 (×5.1)
Mode C	128	167	(×43)	348 (×7.1)	516 (×9.8)
	192	239	(×42)	559 (×5.8)	797 (×7.8)
	256	321	(×40)	837 (×5.0)	1157 (×6.5)

Table 6. Chip-based measurements of hybrid SCP11 protocols execution, with ratio relative to classic versions, using SLH-DSA signature (small).

Protocol	Sec. level	Communication		Processing	Total
		(ms)	(ratio)	(ms) (ratio)	(ms) (ratio)
Mode A	128	544	(×71)	2291 (×34)	2835 (×38)
	192	1079	(×93)	3398 (×26)	4477 (×32)
	256	1932	(×122)	5002 (×22)	6934 (×29)
Mode B	128	226	(×44)	290 (×6.0)	516 (×9.7)
	192	424	(×55)	427 (×4.6)	851 (×8.4)
	256	733	(×69)	629 (×4.0)	1362 (×8.1)
Mode C	128	335	(×87)	2123 (×43)	2458 (×47)
	192	679	(×118)	3160 (×33)	3839 (×37)
	256	1234	(×155)	4655 (×27)	5889 (×33)

Table 7. Chip-based measurements of hybrid SCP11 protocols execution, with ratio relative to classic versions, using FN-DSA (Round 3 Falcon) signature. Note that FN-DSA does not specify a parameter set for security level 192.

Protocol	Sec. level	Communication		Processing	Total
		(ms)	(ratio)	(ms) (ratio)	(ms) (ratio)
Mode A	128	136	(×18)	341 (×5.1)	477 (×6.4)
	256	265	(×17)	750 (×3.3)	1014 (×4.2)
Mode B	128	82	(×16)	290 (×6.0)	371 (×7.0)
	256	162	(×15)	629 (×4.0)	791 (×4.7)
Mode C	128	72	(×19)	173 (×3.5)	244 (×4.6)
	256	137	(×17)	403 (×2.4)	540 (×3.0)

Our future research will broaden the focus to additional protocols in the embedded ecosystem, providing formal proofs for their PQ adaptations. We also plan to collaborate with GlobalPlatform working groups to support PQ protocol standardization and ensure smooth PQ migration across diverse embedded applications.

References

1. ANSSI: ANSSI views on the Post-Quantum Cryptography transition (December 2023), https://cyber.gouv.fr/sites/default/files/document/follow_up_position_paper_on_post_quantum_cryptography.pdf
2. Barbosa, M., Barthe, G., Bhargavan, K., Blanchet, B., Cremers, C., Liao, K., Parno, B.: SoK: Computer-aided cryptography. In: IEEE S&P 2021 [34], pp. 777–795. https://doi.org/10.1109/SP40001.2021.00008
3. Barbosa, M., Barthe, G., Fan, X., Grégoire, B., Hung, S.H., Katz, J., Strub, P.Y., Wu, X., Zhou, L.: EasyPQC: Verifying post-quantum cryptography. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2564–2586. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484567
4. Beguinet, H., Chevalier, C., Ricosset, T., Senet, H.: Formal verification of a post-quantum Signal protocol with Tamarin. In: Ben Hedia, B., Maleh, Y., Krichen, M. (eds.) VECoS '23. LNCS, vol. 14368, pp. 105–121. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-49737-7_8
5. Bhargavan, K., Jacomme, C., Kiefer, F., Schmidt, R.: Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (Aug 2024)
6. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming **75**(1), 3–51 (2008). https://doi.org/10.1016/j.jlap.2007.06.002

7. Blanchet, B., Jacomme, C.: CryptoVerif: a computationally-sound security protocol verifier. Tech. Rep. RR-9526, Inria (Oct 2023), https://inria.hal.science/hal-04253820
8. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 2.05: Automatic cryptographic protocol verifier, user manual and tutorial (Oct 2023), https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf
9. Brendel, J., Fiedler, R., Günther, F., Janson, C., Stebila, D.: Post-quantum asynchronous deniable key exchange and the Signal handshake. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 3–34. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97131-1_1
10. Brendel, J., Fischlin, M., Günther, F., Janson, C., Stebila, D.: Towards post-quantum security for Signal's X3DH handshake. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 404–430. Springer, Cham (Oct 2020). https://doi.org/10.1007/978-3-030-81652-0_16
11. BSI: Elliptic Curve Cryptography (June 2018), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_V-2-1_pdf
12. BSI: Cryptographic Mechanisms: Recommendations and Key Lengths (February 2024), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf
13. Celi, S., Hoyland, J., Stebila, D., Wiggers, T.: A tale of two models: Formal verification of KEMTLS via Tamarin. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS 2022, Part III. LNCS, vol. 13556, pp. 63–83. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-17143-7_4
14. Chen, L., Moody, D., Randall, K., Regenscheid, A., Robinson, A.: Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters. NIST (Feb 2023), https://doi.org/10.6028/NIST.SP.800-186, Special Publication 800-186
15. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order table-based conversion algorithms and masking lattice-based encryption. IACR TCHES **2022**(2), 1–40 (2022). https://doi.org/10.46586/tches.v2022.i2.1-40
16. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order polynomial comparison and masking lattice-based encryption. IACR TCHES **2023**(1), 153–192 (2023). https://doi.org/10.46586/tches.v2023.i1.153-192
17. Cremers, C., Fontaine, C., Jacomme, C.: A logic and an interactive prover for the computational post-quantum security of protocols. In: 2022 IEEE Symposium on Security and Privacy. pp. 125–141. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833800
18. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. DCC **2**(2), 107–125 (1992). https://doi.org/10.1007/BF00124891
19. Dobson, S., Galbraith, S.D.: Post-quantum signal key agreement from SIDH. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022. pp. 422–450. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-17234-2_20
20. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–207 (1983). https://doi.org/10.1109/TIT.1983.1056650
21. Dworkin, M.: Recommendation for Block Cipher – Modes of Operation: The CMAC Mode for Authentication (May 2005). https://doi.org/10.6028/NIST.SP.800-38B

22. ETSI: Quantum-safe Hybrid Key Exchanges (Dec 2020), https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf, ETSI TS 103 744 V1.1.1
23. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU (Oct 2020), https://falcon-sign.info, specification v1.2
24. Gazdag, S.L., Grundner-Culemann, S., Guggemos, T., Heider, T., Loebenberger, D.: A formal analysis of IKEv2's post-quantum extension. In: ACSAC '21. p. 91–105. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3485832.3485885
25. GlobalPlatform: Secure Channel Protocol '03' – Card Specification v2.3 — Amendment D – Version 1.2 (April 2020), https://members.globalplatform.org/viewdocument/secure-channel-protocol-03-amen
26. GlobalPlatform: Secure Channel Protocol '11' – Card Specification v2.3 – Amendment F – Version 1.4 (March 2023), https://globalplatform.org/specs-library/secure-channel-protocol-11-amendment-f/
27. Gray, J., Onsworth, M.: Certificate mechanisms for transitioning to post-quantum cryptography. International Cryptographic Module Conference (ICMC) 2024, https://icmconference.org/?session=certificate-mechanisms-for-transitioning-to-post-quantum-cryptography-q01c
28. GSMA: RSP Architecture SGP.21 v3.1 (December 2023), https://www.gsma.com/solutions-and-impact/technologies/esim/wp-content/uploads/2023/12/SGP.21-V3.1.docx
29. GSMA: RSP Technical Specification SGP.22 v3.1 (December 2023), https://www.gsma.com/solutions-and-impact/technologies/esim/wp-content/uploads/2023/12/SGP.22-V3.1.docx
30. GSMA: Secured Applications for Mobile v1.1 (November 2023), https://www.gsma.com/get-involved/working-groups/wp-content/uploads/2023/11/SAM.01-v1.1-1.pdf
31. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, pp. 29–37. Springer, Berlin, Heidelberg (Apr 1990). https://doi.org/10.1007/3-540-46885-4_5
32. Hashimoto, K., Katsumata, S., Kwiatkowski, K., Prest, T.: An efficient and generic construction for Signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 410–440. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75248-4_15
33. Hülsing, A., Ning, K.C., Schwabe, P., Weber, F.J., Zimmermann, P.R.: Post-quantum WireGuard. In: IEEE S&P 2021 [34], pp. 304–321. https://doi.org/10.1109/SP40001.2021.00030
34. 2021 IEEE Symposium on Security and Privacy. IEEE Computer Society Press (May 2021)
35. Jackson, D., Cremers, C., Cohn-Gordon, K., Sasse, R.: Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2165–2180. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339813
36. Kobeissi, N.: Verifpal user manual (Sep 2023), https://verifpal.com/res/pdf/manual.pdf
37. Kobeissi, N., Nicolas, G., Tiwari, M.: Verifpal: Cryptographic protocol analysis for the real world. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) IN-

DOCRYPT 2020. LNCS, vol. 12578, pp. 151–202. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-65277-7_8
38. Linker, F., Sasse, R., Basin, D.: A formal analysis of Apple's iMessage PQ3 protocol. Cryptology ePrint Archive, Paper 2024/1395 (2024), https://eprint.iacr.org/2024/1395
39. Lowe, G.: A hierarchy of authentication specification. In: Computer Security Foundations Workshop. pp. 31–44. IEEE Computer Society (1997). https://doi.org/10.1109/CSFW.1997.596782
40. Moody, D., Perlner, R., Regenscheid, A., Robinson, A., Cooper, D.: Transition to Post-Quantum Cryptography Standards. NIST (Nov 2024), https://doi.org/10.6028/NIST.IR.8547.ipd, Initial public draft
41. NIST: Module-Lattice-Based Digital Signature Standard (Aug 2024). https://doi.org/10.6028/NIST.FIPS.204, FIPS 204
42. NIST: Module-Lattice-Based Key-Encapsulation Mechanism Standard (Aug 2024). https://doi.org/10.6028/NIST.FIPS.203, FIPS 203
43. NIST: Stateless Hash-Based Digital Signature Standard (Aug 2024). https://doi.org/10.6028/NIST.FIPS.205, FIPS 205
44. Paul, S., Scheible, P.: Towards post-quantum security for cyber-physical systems: Integrating PQC into industrial M2M communication. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part II. LNCS, vol. 12309, pp. 295–316. Springer, Cham (Sep 2020). https://doi.org/10.1007/978-3-030-59013-0_15
45. Schmidt, B., Meier, S., Cremers, C., Basin, D.A.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Chong, S. (ed.) CSF 2012. pp. 78–94. IEEE Computer Society (2012). https://doi.org/10.1109/CSF.2012.25
46. The Tamarin Team: Tamarin prover manual (May 2024), https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf

A Hybrid Versions for Modes B and C

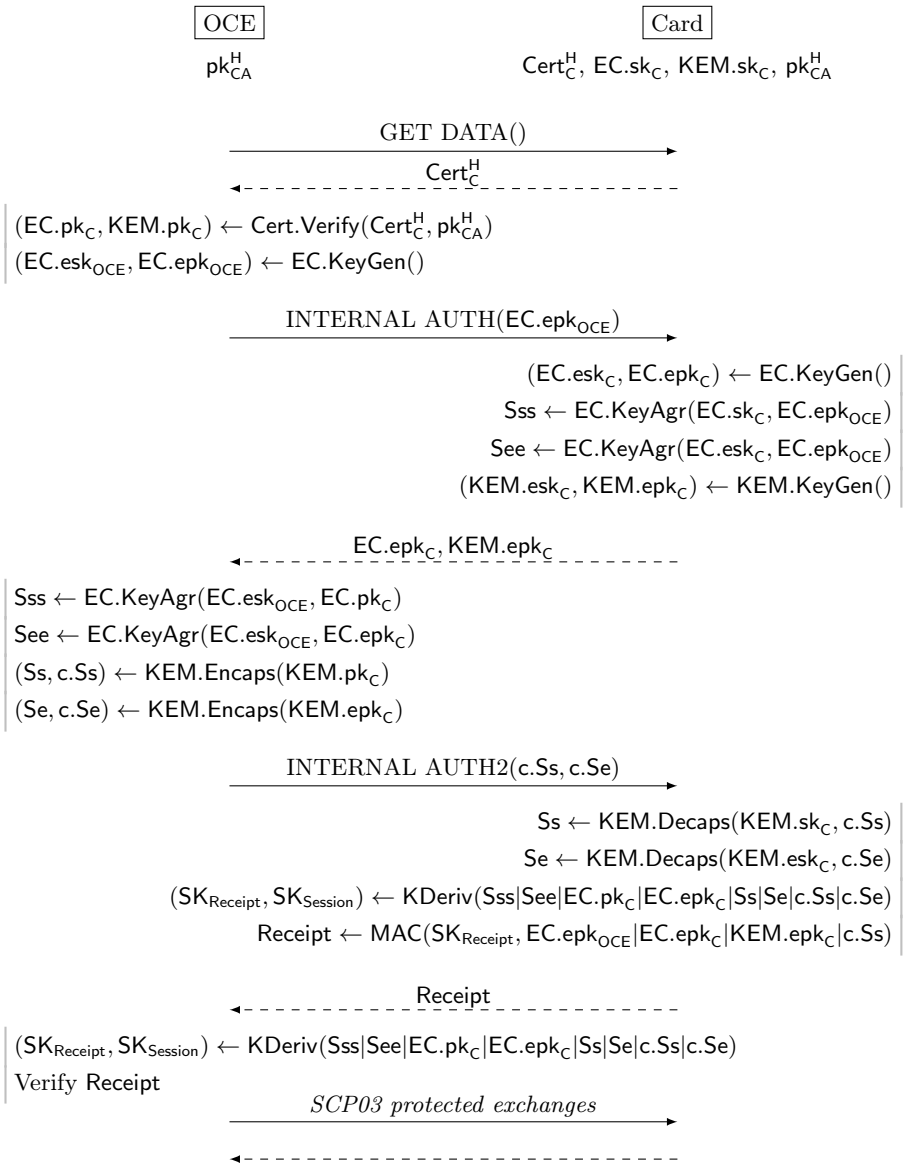

Fig. 8. Hybrid version for Mode B

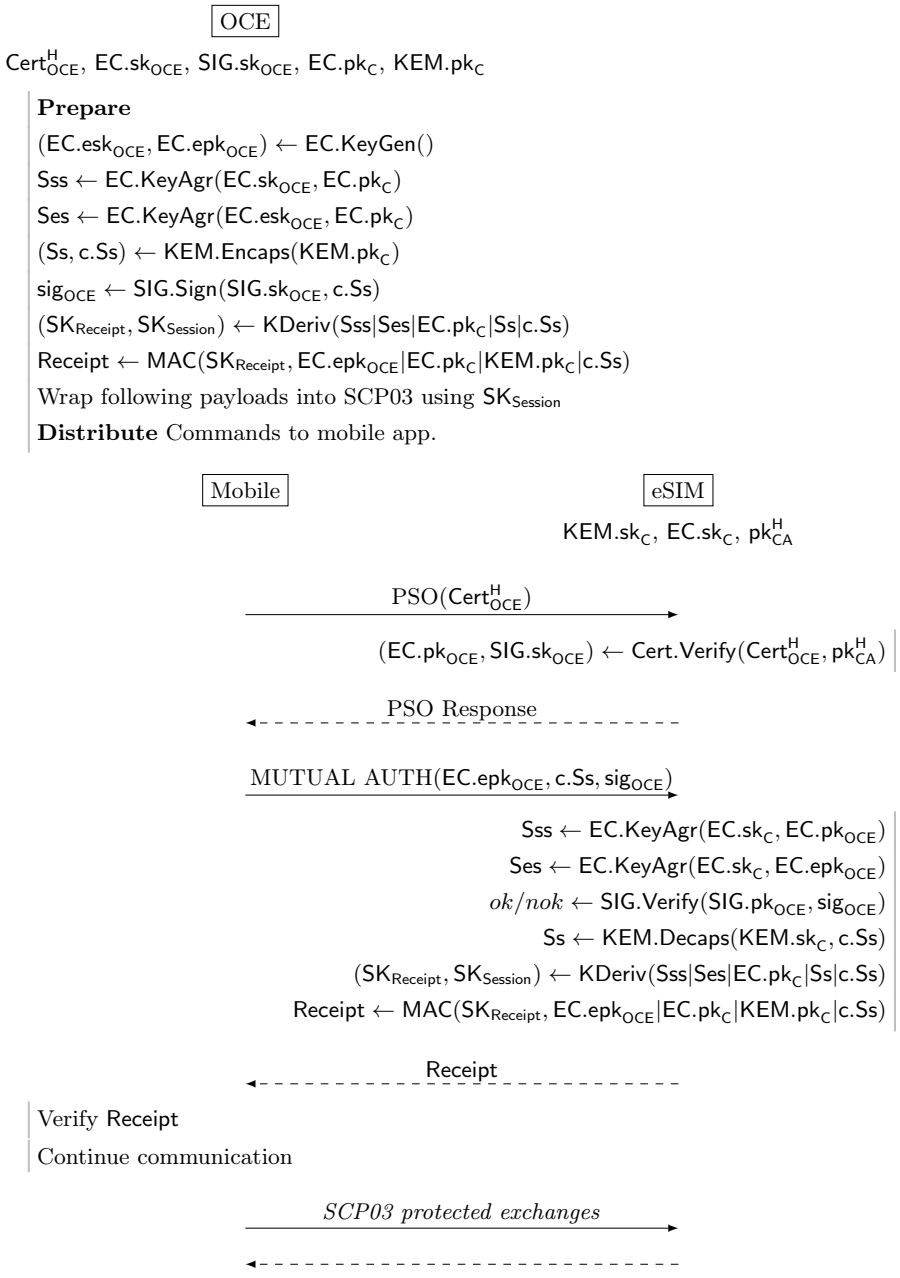

Fig. 9. Hybrid version for Mode C