Post-Quantum Security for the Extended Access Control Protocol

Marc Fischlin[1], Jonas von der Heyden[2] (✉), Marian Margraf[3], Frank Morgner[4], Andreas Wallner[5], and Holger Bock[5]

[1] Technische Universität Darmstadt, `marc.fischlin@tu-darmstadt.de`
[2] Bergische Universität Wuppertal, `jvdh@uni-wuppertal.de`
[3] Fraunhofer AISEC, `marian.margraf@aisec.fraunhofer.de`
[4] Bundesdruckerei GmbH, `frank.morgner@bdr.de`
[5] Infineon Technologies, {`andreas.wallner,holger.bock`}`@infineon.com`

Abstract. The *Extended Access Control* (EAC) protocol for authenticated key agreement is mainly used to secure connections between machine-readable travel documents (MRTDs) and inspection terminals, but it can also be adopted as a universal solution for attribute-based access control with smart cards. The security of EAC is currently based on the Diffie-Hellman problem, which may not be hard when considering quantum computers.

In this work we present PQ-EAC, a quantum-resistant version of the EAC protocol. We show how to achieve post-quantum confidentiality and authentication without sacrificing real-world usability on smart cards. To ease adoption, we present two main versions of PQ-EAC: One that uses signatures for authentication and one where authentication is facilitated using long-term KEM keys. Both versions can be adapted to achieve forward secrecy and to reduce round complexity. To ensure backwards-compatibility, PQ-EAC can be implemented using only Application Protocol Data Units (APDUs) specified for EAC in standard BSI TR-03110. Merely the protocol messages needed to achieve forward secrecy require an additional APDU not specified in TR-03110. We prove security of all versions in the real-or-random model of Bellare and Rogaway.

To show real-world practicality of PQ-EAC we have implemented a version using signatures on an ARM SC300 security controller, which is typically deployed in MRTDs. We also implemented PQ-EAC on a VISOCORE® terminal for border control. We then conducted several experiments to evaluate the performance of PQ-EAC executed between chip and terminal under various real-world conditions. Our results strongly suggest that PQ-EAC is efficient enough for use in border control.

Keywords: Access Control · Machine Readable Travel Documents · Post-Quantum Cryptography · Smart Cards

This is the full version of a paper which appears in the proceedings of Security Standardisation Research, 8th International Conference, SSR 2023, Lyon, France, April 22-23, 2023.

1 Introduction

EAC protocol The Extended Access Control (EAC) protocol for authenticated key agreement was proposed by the *German Federal Office for Information Security* (BSI) for German ePassports in 2005 and is defined in the technical guideline TR-03110 [18]. EAC is meant to provide authenticated key establishment between a terminal and a smart card. ICAO Doc 9303 [33], a standard for electronic machine-readable travel documents (eMRTDs) such as ePassports, mandates authentication via EAC before a terminal may request sensitive data such as fingerprints from the chip in an eMRTD. The security of EAC rests on assumptions about the hardness of computing discrete logarithms. Due to the results of Shor [60], these assumptions are now considered to be false when taking into account quantum computers. In this work we present PQ-EAC, a new version of the EAC protocol that achieves security against attacks from quantum computers by substituting Diffie-Hellman key exchange with post-quantum Key Encapsulation Mechanisms (KEMs).

Post-quantum cryptography In 1994, Peter Shor presented a quantum algorithm that solves integer factorization and discrete logarithms — two problems for which no efficient algorithm on classical computers is known — in polynomial time [60]. Shor's work thereby falsified the foundational assumption of public-key cryptography that factorization and discrete logarithms are hard problems. While only a quantum computer of sufficient size could break public-key cryptography as used in modern internet infrastructure, recent developments show that such a machine is a realistic scenario [4]. Michele Mosca estimates a 1/7 chance of a quantum computer breaking RSA-2048 by 2026 and a 1/2 chance of this break occurring by 2031 [49].

Fortunately, there are alternatives to cryptography based on factorization or discrete logarithms. As proposed by Bernstein [12] we say that a cryptographic construction is *'post-quantum secure'* when it is assumed to be appropriately secure against quantum adversaries. For performance reasons, one of the most promising approaches to post-quantum cryptography is *lattice-based* cryptography. In regards to post-quantum digital signatures, hash-based signatures are an interesting alternative to classical constructions. They have been known since the 1970s [47,42] and provide provable security under the weak assumption that one-way functions exist. They require careful state management and might therefore not be suitable for all use cases. However, due to the confidence in their security, they could serve as a useful building block for *cryptographic agility* in smart cards: In the case that the post-quantum algorithms deployed on a smart card turn out to be insecure, they could be replaced with an update whose integrity is secured by a hash-based signature.

NIST competition To alleviate the threat of quantum computers against cryptography, the US National Institute of Standards and Technology (NIST) initiated a post-quantum cryptography (PQC) competition in 2017. In 2022, NIST announced that they will standardize the lattice-based Key Encapsulation Mechanism (KEM) Kyber [58], two lattice-based signature schemes (Dilithium [44], Falcon [53]) and the hash-based signature scheme SPHINCS+ [30] as quantum-resistant cryptographic mechanisms [2]. In addition, the hash-based signature schemes LMS [46] and XMSS [29] have already been recommended in NIST SP 800-208 [50].

Hybrid Protocols Many of these conjecturally *post-quantum secure* schemes have not yet been adequately studied in terms of their security. This concerns not only cryptanalytic attacks using classical computers, but also side-channel and error attacks and the exploitation of implementation errors. Therefore, it is prudent to use hybrid protocols, i.e., a combination of classical and quantum-resistant primitives; see also, for example, the recommendations of the German Federal Office for Information Security [19].

PoQuID Even though a quantum computer powerful enough to break modern cryptography might be more than a decade away, in the context of identity documents urgent action is required. This is because identity documents typically have a validity period of ten years and technical changes require lengthy regulatory approval. Therefore, in 2019 the German federal ministry for economic affairs commissioned researchers at Fraunhofer AISEC, Infineon and Bundesdruckerei to conduct the research project *Post-Quantum Protocols for Identity Documents* (PoQuID). The goal of PoQuID was the development of a new quantum-resistant version of the EAC protocol, ideally using hybrid schemes that combine post-quantum and classical algorithms. Another goal of PoQuID was a software implementation of this new protocol for contactless smart cards (in eMRTDs) as well as inspection terminals, in order to create a blueprint for the standardization of the new EAC protocol with ICAO.

In this paper, we summarize the project results and showcase several versions of a quantum-resistant EAC protocol with varying security properties and trade-offs.

1.1 Outline

We start out with a discussion of ePassports in Section 2 and exhibit the original EAC protocol in Section 3. In Section 4 we subsequently propose several modifications to EAC, which will replace the Diffie-Hellman-style key exchange with *Key Encapsulation Mechanisms* (KEMs). Afterwards, we discuss our implementation and give performance benchmarks in Section 5.

To distinguish between the EAC protocol as defined by BSI [18] and our proposal for post-quantum EAC, we call the former *classic EAC* or *EAC classic* and the latter *PQ-EAC*. We continue to write EAC when referring to the protocol as such. EAC is usually conducted between two parties: One the one side we have an eMRTD which might be an identity card, a passport or similar, in the following called 'chip'. On the other side we have a chip reader, which could be an inspection terminal at border control, a device to authenticate identity cards for e-government services, or similar, in the following called 'terminal'. The terms MRTD and eMRTD are used in this document as a generic reference to all types of Machine Readable Travel Documents based on optical character reading or electronically enabled means. Examples of MRTDs are ePassports, laissez-passer, identity cards, seafarer cards and refugee travel documents.

We use the notion of *forward secrecy* as given in Boyd and Gellert [16]: An authenticated key exchange (AKE) protocol provides forward secrecy (resp. is *forward-secure*) if compromise of long-term secrets does not lead to compromise of session keys of previously completed sessions.

1.2 Related Work on Post-Quantum Cryptography for MRTDs

Post-quantum Cryptography NIST recently announced [2] that as an outcome of the PQC competition they will recommend and standardize the lattice-based KEM Kyber [58] and the lattice-based signature scheme Dilithium [44] as quantum-resistant schemes. Accordingly, we instantiate the KEMs and signature schemes in our PQ-EAC implementation with Kyber and Dilithium. As we will detail in Section 5, both schemes are efficient enough for use in ressource-constrained environments such as smart cards typically deployed in MRTDs.

Simultaneously, there have been multiple works concerned with making Diffie-Hellman-based protocols quantum-resistant: Schwabe et al. [59] present post-quantum versions of TLS, including a version with mutual authentication. Hülsing et al. [31] show how to achieve post-quantum security for the handshake protocol of the WireGuard VPN. Brendel et al. [17] attempt to give a post-quantum alternative to Signal's X3DH handshake using split KEMs. Finally, Angel et al. [3] introduce a post-quantum variant of the Noise framework. Unlike PQ-EAC, the protocols mentioned above cannot serve as drop-in replacements for EAC as standardized in TR-03110. However, our work and the mentioned protocols share the basic idea of replacing a Diffie-Hellman key exchange with KEMs to achieve post-quantum security.

PAKE In BSI standard TR-03110 [18], the Password Authenticated Connection Establishment (PACE) protocol – a variant of *Password-Authenticated Key Exchange* (PAKE) – is used to set up the initial communication between MRTD chip and terminal reader. Since the security of PACE is based on hardness assumptions about discrete logarithms, it needs to be modified to become quantum-resistant. Fortunately, there have been a number of attempts to devise post-quantum PAKE, for example by Katz and Vaikuntanathan [38] and Katz and Groce [27]. Moreover, there has been a recent proposal for PAKE based on group actions by Abdalla et al. [1]. However, since the only known quantum-resistant instantiation of group actions is based on CSIDH [22], this construction would not be efficient enough for use in smart cards.

PKI Regarding public-key infrastructure (PKI), there has been work by Bindel et al. [14] towards hybrid PKI (meaning PKI that combines classical and post-quantum constructions). Additionally there are proposals by Pradel and Mitchell

[52] as well as Vogt and Funke [61] speficially in regards to post-quantum PKI for MRTDs.

ePassports Recent publications in regards to ePassports have been concerned with the security of Basic Access Control (BAC) [5,11,24], the security of PACE [11,10] and the security of classic EAC [23].

EAC Besides a status report about the PoQuID project [48], and a master thesis by Kussmaul [41] to the best of our knowledge there have not been any publications regarding the post-quantum security of the EAC protocol.

2 ePassports

History Extended Access Control (EAC) was devised as a mechanism to protect biometric data stored in MRTDs. The International Civil Aviation Organization (ICAO), a specialized agency of the United Nations, introduced standards for electronic MRTDs and specifically ePassports in 2006 in Volume 2 of the sixth edition of ICAO Doc 9303 [33]. Since then, more than 140 countries have adopted ePassports, with 90 countries enrolled in the ICAO Public Key Directory (PKD).

ICAO mandates in Doc 9303 [33] the execution of EAC before terminals may read sensitive data from MRTDs. The technical specification of EAC for EU passports has been devised by the Brussels Interoperability Group (BIG) and is published by the German Federal Office for Information Security (BSI) in their technical report TR-03110 [18].

EAC as defined in this specification includes two authentication mechanisms, Chip Authentication (CA) and Terminal Authentication (TA): CA authenticates the chip and TA prevents the reading of biometric data from unauthorized terminals. While EAC originated in the context of travel documents, it is also utilized in different contexts: For example, version 2 of EAC authenticates transactions performed by the German identity card, such as proof of identity or e-government. While we focus on the passport context in this work, most of our results apply more generally.

Security Threats to ePassports Because ePassports use contactless radio-frequency identification (RFID) technology to communicate with terminals, they are subject to *skimming* and *eavesdropping* attacks: In skimming attacks, the attacker retrieves data from the chip by making connection attempts in close physical proximity of the MRTD. Even if the attacker cannot retrieve any information from the passport, he might be able to link it to previous connection attempts, meaning that he is able to *track* the passport. If a MRTD is secure against such attacks, it is said to be *unlinkable* [24].

Eavesdropping refers to the interception of communication between honest parties by an attacker. Authenticated key exchange (AKE) and encryption is used to prevent an eavesdropper from reading or modifying the communication between chip and terminal. ICAO Doc 9303 specifies two layers of authentication:

Access to information that can be read by anyone in physical possession of the ePassport, including primary biometrics (facial image), requires authentication via PACE or BAC (described below) that uses the machine-readable zone (MRZ) of the MRTD as a trust anchor. The underlying assumption here is that when a passport holder allows a terminal to read the MRZ of his passport via optical character recognition (OCR), he is also consenting to the transmission of other data visually recognizable on the document. Also note that the use of OCR instead of RFID to read the MRZ prevents eavesdropping on the MRZ itself by an attacker. The access to secondary biometrics, meaning either fingerprints or iris data, requires the execution of Terminal Authentication.

Other attacks on the ePassport concern its integrity: One way to forge a passport would be to modify the data groups (name, nationality, etc.) on a passport's chip. To prevent this, the passport issuer hashes all data groups, signs them and stores them in the Document Security Object (SO_D). To verify data integrity, a terminal must then use *Passive Authentication* (described four paragraphs below). An adversary could overcome these provisions by copying the SO_D to a new chip (also called *cloning*). Assuming that the protections provided for the chip's private key successfully prohibit any access to it, cloning can be detected during Chip Authentication when the chip fails to generate a shared key.

PACE To establish connection, chip and terminal generate a high-entropy shared key through *Password Authenticated Connection Establishment* (PACE), an asymmetric key exchange procedure based on a shared (possibly low-entropy) password. In the context of ePassports, the password is either the MRZ or the card access number (CAN). Some document types (for example the German identity card) also support a secret user PIN. As such, the terminal authenticates itself to the chip by proving that the holder has entered a pin or put the document on the terminal reader. PACE creates an encrypted communication channel and allows terminals to read less sensitive data groups such as name, nationality and other information.

BAC Alternatively, MRTDs issued before 2018 may use *Basic Access Control* (BAC) instead of PACE to establish a shared key from the MRTD's machine-readable zone (MRZ). Since the MRZ consists of the holder's birth date, the MRTD's expiry date and a 9-digit serial number[1], the maximum entropy of the key is 73 bits, but in practice the entropy of the key can be assumed to be 40-50 bits [11]. The low entropy of the key permits *sniffing* attacks where the communication between terminal and chip is recorded by an eavesdropper and later decrypted by guessing the shared key [43,5]. Moreover, Filimonov et al. showed in 2019 that it is possible to track passports using BAC [24]. Due to these weaknesses, it is recommended (and mandated for ePassports issued after 2017) to use PACE instead of BAC [11].

[1] Depending on the issuing country, the serial number is subject to various constraints: Some digits may be predictable check digits, some blocks may indicate the issuing passport office, etc.

Public-key Infrastructure In order for chip and terminal to be able to mutually authenticate each other, each document issuing country has established a PKI, which consists of the following entities (among others): a Country Signing Certificate Authority (CSCA) and a Country Verifying Certificate Authority (CVCA), Document Signer certificates, Document Verifier certificates and certificate revocation lists (CRLs). Document Signer certificates and key-pairs are issued by the CSCA to the entity that is manufacturing the MRTDs. The private key is then used to sign a list of hashes of the data groups of the MRTD. The signature, hashes and the respective Document Signer certificate are then stored in the SO_D and can be used by the MRTD to authenticate itself to a terminal.

Similarly, the CVCA issues manufacturers of terminals Document Verifier key-pairs and certificates, which can be used by a terminal to authenticate itself towards chips.

To enable international interoperability, all countries share their CSCA and CVCA public keys (via a master list), Document Signer certificates, Document Verifier certificates and CRLs in the ICAO PKD.

Passive Authentication The terminal uses a protocol called *Passive Authentication* to authenticate the data that is stored on the chip. Passive Authentication involves the following steps:

– The terminal reads the *Document Security Object* (SO_D) from the chip and retrieves the corresponding Document Signer certificate, the CSCA certificate and the corresponding certificate revocation list. The SO_D holds hashes of all data groups and a signature over these hashes. Note that the chip's public key is also stored in one of these data groups.
– The terminal verifies the Document Signer certificate.
– The terminal then computes the hash values of all data groups it has access to and compares them to the hash values from the SO_D. Subsequently it verifies the signature over these hash values using the public key from the Document Signer certificate.

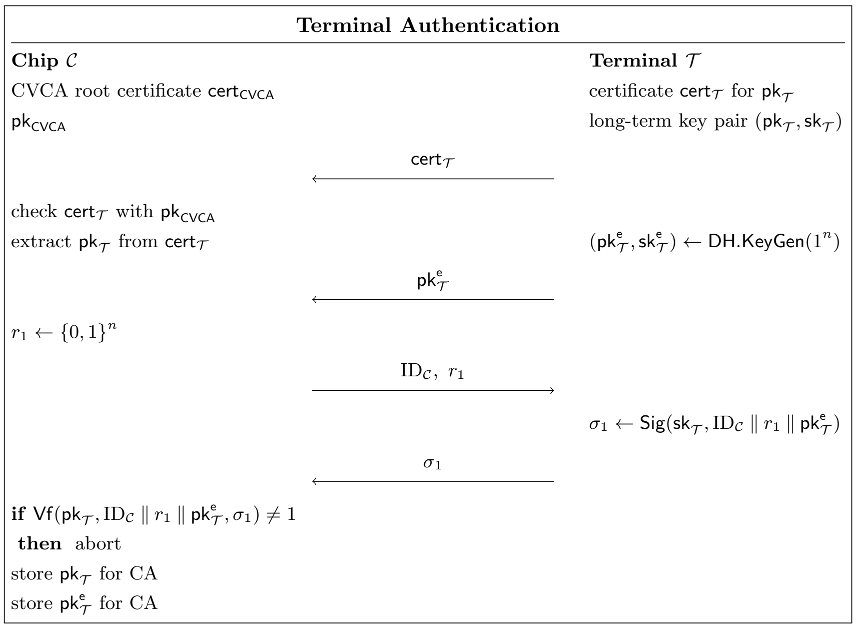
Passive Authentication does not protect against cloning attacks, where the SO_D (but not the secret key which is assumed to be secured against copying) is copied from one MRTD to another, but only verifies the integrity of the data groups on the chip. To prove that the chip is actually in possession of the corresponding secret key, EAC and Chip Authentication come into play.

3 Classic EAC Protocol

Overview EAC works as follows: The terminal uses *Chip Authentication* (CA) to verify the authenticity of the chip. Vice versa, the chip can verify the inspection system's authorization to read sensitive biometric data such as fingerprints or iris data during *Terminal Authentication* (TA) [33]. In EAC version 1, which is included in ICAO's standard for ePassports, chip and terminal first execute CA and then TA. If EAC is not executed, the terminal is not allowed to read secondary biometric data. To assure more privacy for the MRTD holder, in (the otherwise identical) version 2 of EAC chip and terminal first perform TA and then CA, and the terminal is only allowed to read data groups once PACE, TA and CA have completed. EAC version 2 is, for example, used by the German identity card. In this work we aim for the stronger security properties of EAC version 2 but maintain compatibility with EAC version 1.

We say $x \leftarrow \{0,1\}^n$ to denote that a n-bit string x is sampled uniformly at random from $\{0,1\}^n$. With $x \leftarrow F(\dots)$ we refer to a non-deterministic assignment for a function F, or, if F is a distribution, random assignment using this distribution. We denote the Diffie-Hellman (DH) key generation procedure with $\mathsf{DH.KeyGen}$ and the derivation of a shared DH key with $\mathsf{DH}(\cdot,\cdot)$.

Terminal Authentication The goal of TA as shown in Figure 1 is to prove to the chip that the inspection terminal is authorized to view sensitive data of the chip. In the context of passports, sensitive data means secondary biometric data such as fingerprints or iris data of the passport holder. To show that the terminal is in possession of a key-pair $(\mathsf{pk}_\mathcal{T}, \mathsf{sk}_\mathcal{T})$ that is validated by a Document Verifier certificate, the terminal initiates a challenge-response procedure.



Terminal Authentication
Chip $\mathcal{C}$
CVCA root certificate $\mathsf{cert}_\mathsf{CVCA}$
$\mathsf{pk}_\mathsf{CVCA}$
Terminal $\mathcal{T}$
certificate $\mathsf{cert}_\mathcal{T}$ for $\mathsf{pk}_\mathcal{T}$
long-term key pair $(\mathsf{pk}_\mathcal{T}, \mathsf{sk}_\mathcal{T})$
$\mathsf{cert}_\mathcal{T}$
check $\mathsf{cert}_\mathcal{T}$ with $\mathsf{pk}_\mathsf{CVCA}$
extract $\mathsf{pk}_\mathcal{T}$ from $\mathsf{cert}_\mathcal{T}$
$(\mathsf{pk}^\mathsf{e}_\mathcal{T}, \mathsf{sk}^\mathsf{e}_\mathcal{T}) \leftarrow \mathsf{DH.KeyGen}(1^n)$
$\mathsf{pk}^\mathsf{e}_\mathcal{T}$
$r_1 \leftarrow \{0,1\}^n$
$\mathrm{ID}_\mathcal{C},\ r_1$
$\sigma_1 \leftarrow \mathsf{Sig}(\mathsf{sk}_\mathcal{T}, \mathrm{ID}_\mathcal{C} \parallel r_1 \parallel \mathsf{pk}^\mathsf{e}_\mathcal{T})$
σ_1
if $\mathsf{Vf}(\mathsf{pk}_\mathcal{T}, \mathrm{ID}_\mathcal{C} \parallel r_1 \parallel \mathsf{pk}^\mathsf{e}_\mathcal{T}, \sigma_1) \neq 1$
then abort
store $\mathsf{pk}_\mathcal{T}$ for CA
store $\mathsf{pk}^\mathsf{e}_\mathcal{T}$ for CA

Fig. 1: Terminal Authentication in the classic EAC protocol.

Firstly, the terminal generates an ephemeral Diffie-Hellman key pair $(\mathsf{pk}^\mathsf{e}_\mathcal{T}, \mathsf{sk}^\mathsf{e}_\mathcal{T})$ and sends $\mathsf{pk}^\mathsf{e}_\mathcal{T}$ to the chip. It also sends its certificate $\mathsf{cert}_\mathcal{T}$, which consists of the Document Verifier certificate and the validation of its public key $\mathsf{pk}_\mathcal{T}$ by the Document Verifier. Subsequently, the chip verifies the certificate and the terminal's public key $\mathsf{pk}_{\mathcal{T}}$.

To generate the challenge, the chip chooses uniformly at random a nonce r_1 and sends it to the terminal. In addition, it sends a pre-agreed value $\mathrm{ID}_{\mathcal{C}}$. If PACE is used, $\mathrm{ID}_{\mathcal{C}}$ is a suitable hash of the chip's public key from the PACE key agreement; this serves to bind EAC with the previous PACE execution. If BAC is used, it is the document number read from the MRZ.

Finally, the terminal proves possession of the secret key $\mathsf{sk}_{\mathcal{T}}$ by generating a signature over r_1, $\mathrm{ID}_{\mathcal{C}}$ and its ephemeral public key. The chip concludes the TA if it can successfully verify the signature using $\mathsf{pk}_{\mathcal{T}}$.

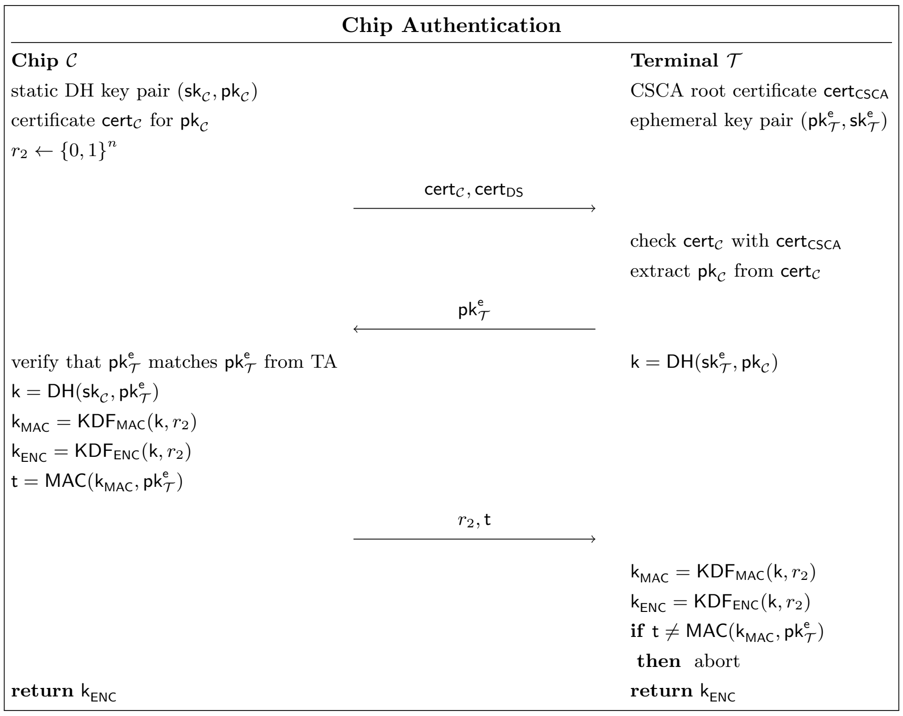
Chip Authentication In CA (shown in Figure 2), the chip $\mathcal{C}$ authenticates itself to the terminal $\mathcal{T}$ with its static DH public key pair $(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}})$ that is validated by the Document Signer. At the outset, the chip sends the terminal its Document Signer certificate, its SO_D and its static public key $\mathsf{pk}_{\mathcal{C}}$, all of which we will call $\mathsf{cert}_{\mathcal{C}}$ in Figure 2.[2] Afterwards, the terminal sends the ephemeral public Diffie-Hellman key $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ generated during the TA. As the chip already received $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ during TA, one might wonder whether this message could be eliminated. Unfortunately, this is not possible due to the need to maintain compatibility with EAC version 1 where the TA is optional.

To avoid non-repudiation — which is undesirable from a privacy standpoint — $\mathcal{C}$ does not produce a signature to authenticate itself. Instead, chip and terminal establish a shared key k via a DH key exchange using the chip's long-term DH key $\mathsf{pk}_{\mathcal{C}}$ and the terminal's ephemeral DH key $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$. Afterwards, the chip generates a nonce r_2 uniformly at random and derives keys $\mathsf{k}_{\mathsf{MAC}}, \mathsf{k}_{\mathsf{ENC}}$ from the shared key k and r_2. Finally, $\mathcal{C}$ sends sends r_2 and a Message Authentication Code (MAC) t over $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ under the key $\mathsf{k}_{\mathsf{MAC}}$ to $\mathcal{T}$. If t matches the MAC computed by $\mathcal{T}$, the Chip Authentication concludes successfully. $\mathsf{k}_{\mathsf{ENC}}$ can be used to encrypt any further communication between terminal and chip after the completion of EAC.

Security properties The main goals of EAC are the authentication of chip and terminal to each other and the establishment of a shared secret key between chip and terminal. Dagdelen and Fischlin [23] have proven authenticated key-exchange (AKE) security for EAC in the random oracle model (ROM). However, there are also other security properties that the EAC protocol is supposed to provide: Firstly, the protocol should provide *forward secrecy*, meaning that past session keys are protected from being recovered, even if the long-term secrets of chip or terminal are compromised. While the classic EAC version only provides this for the terminal's long-term secret [23], we will present versions of PQ-EAC providing forward secrecy for the long-term keys of both chip and terminal in Section 4.

Secondly, since the chip represents a travel document, it is desirable for the chip to have plausible deniability of participation in EAC, meaning we do **not**

[2] This is usually performed during Passive Authentication but to simplify the presentation, we show it as part of the CA.



Chip Authentication
Chip $\mathcal{C}$
static DH key pair $(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}})$
certificate $\mathsf{cert}_{\mathcal{C}}$ for $\mathsf{pk}_{\mathcal{C}}$
$r_2 \leftarrow \{0,1\}^n$
Terminal $\mathcal{T}$
CSCA root certificate $\mathsf{cert}_{\mathsf{CSCA}}$
ephemeral key pair $(\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{e}}_{\mathcal{T}})$
$\xrightarrow{\mathsf{cert}_{\mathcal{C}}, \mathsf{cert}_{\mathsf{DS}}}$
check $\mathsf{cert}_{\mathcal{C}}$ with $\mathsf{cert}_{\mathsf{CSCA}}$
extract $\mathsf{pk}_{\mathcal{C}}$ from $\mathsf{cert}_{\mathcal{C}}$
$\xleftarrow{\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}}$
verify that $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ matches $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ from TA
$\mathsf{k} = \mathsf{DH}(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$
$\mathsf{k}_{\mathsf{MAC}} = \mathsf{KDF}_{\mathsf{MAC}}(\mathsf{k}, r_2)$
$\mathsf{k}_{\mathsf{ENC}} = \mathsf{KDF}_{\mathsf{ENC}}(\mathsf{k}, r_2)$
$\mathsf{t} = \mathsf{MAC}(\mathsf{k}_{\mathsf{MAC}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$
$\mathsf{k} = \mathsf{DH}(\mathsf{sk}^{\mathsf{e}}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$
$\xrightarrow{r_2, \mathsf{t}}$
$\mathsf{k}_{\mathsf{MAC}} = \mathsf{KDF}_{\mathsf{MAC}}(\mathsf{k}, r_2)$
$\mathsf{k}_{\mathsf{ENC}} = \mathsf{KDF}_{\mathsf{ENC}}(\mathsf{k}, r_2)$
if $\mathsf{t} \neq \mathsf{MAC}(\mathsf{k}_{\mathsf{MAC}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$
then abort
return $\mathsf{k}_{\mathsf{ENC}}$
return $\mathsf{k}_{\mathsf{ENC}}$

Fig. 2: Chip Authentication in the original EAC protocol.

want *non-repudiation.* Since signatures usually provide non-repudiation, in EAC the authenticity of the chip is established through a key-agreement.

4 Quantum-resistant EAC Protocol Versions

4.1 Overview

This section is concerned with our efforts towards a quantum-resistant version of EAC (called PQ-EAC). EAC is subject to many, sometimes conflicting constraints. For example, to reduce waiting times for access control, it would be beneficial to use round-reduced EAC. However, for privacy reasons, it may not be desirable that the chip sends information about its public key to an unauthenticated terminal. Similarly, forward secrecy helps to protect earlier messages in the case that the private key of the chip is compromised but increases the number of cryptographic operations. An additional consideration is that some versions of PQ-EAC might be more suitable than others for maintaining backwards-compatibility with EAC classic.

We distinguish between solutions with the terminal signing the data ($\mathsf{SigPQEAC}$, see Section 4.2) and an alternative where the terminal uses a long-term KEM for authentication ($\mathsf{KemPQEAC}$, see Section 4.3). The common approach in both cases is to replace the Diffie-Hellman key agreement step in the CA by having the terminal send a secret key protected under the chip's long-term KEM key $\mathsf{pk}_{\mathcal{C}}$. The KEM itself is assumed to be *post-quantum secure*.

For each of the two types we can consider a forward-secure variant where the chip, in addition to the long-term KEM key $\mathsf{pk}_{\mathcal{C}}$, also generates an ephemeral KEM key $\mathsf{pk}_{\mathcal{C}}^{e}$ during protocol execution. The terminal then encapsulates another key k^{e} under $\mathsf{pk}_{\mathcal{C}}^{e}$. The long-term key is then used for authentication, especially of the ephemeral part. Both parties derive the session key from either the encapsulated key k (in the non-forward-secure version) resp. together with the encapsulated key k^{e} (in the forward-secure version with the ephemeral KEM). Another option we we discuss below is to combine the TA and CA phases and save on the round-trip time by joining some data into single transmissions.

We thus have eight protocol variants via possible combinations: with and without terminal signatures, with and without forward security, and potentially combining messages. The combined versions allow for further simplifications, e.g., by letting the terminal only create a single signature instead of two signatures. We present the two fundamentally different versions, one with signatures for terminal authentication and one with KEMs for authentication, and discuss the other four sub versions for either one within. We discuss further options for the combined versions afterwards. We leave it to the discretion of the implementing bodies to weigh advantages and disadvantages of the various versions.

Assumptions For all protocols we assume that the following holds:

- Passive Authentication binds the chip's data groups with its key-pair $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}})$.
- The communication partners have agreed on $\mathrm{ID}_{\mathcal{C}}$. If PACE is used, it is a suitable hash of the chip's public key from the previous PACE execution; if BAC is used, it is the document number read from the machine-readable zone (MRZ).
- The protocol is instantiated with a post-quantum or hybrid *IND-CCA-secure* key encapsulation mechanism, a post-quantum or hybrid *EUF-CMA-secure* signature scheme, a key derivation function KDF with output length n and a dual pseudorandom key combiner KComb. See Appendix A for the respective definitions and security experiments of the mentioned primitives.
- The security parameter n is of appropriate size and all keys have been generated according to n.

We note that while chip and terminal send the EAC protocol messages in an encrypted channel that was established through BAC or PACE, PQ-EAC would still provide AKE security if all messages were sent in the open.

4.2 Authentication With Signatures

We start with the four variants in which the terminal uses a quantum-resistant signature scheme to authenticate. The protocol framework is given in Figure 3 and describes the plain variant, as well as the forward-secure variant (with the optional ephemeral KEM steps written as [ ]) and the combined version of TA and CA (with the dashed arrows indicating which protocol messages can be combined).

In the protocol we use all relevant exchanged data $(\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$ for authentication purposes, including the certificates and the nonce and ciphertexts sent by the terminal. Since these data coincide with the session identifier sid used in the security proof we conveniently write sid in the protocol, too. We note that the final value $\mathsf{k}_{\mathtt{CNF}}$ transmitted by the chip with the last protocol message also serves as an authentication tag for all values in $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2,$ c, $[\mathsf{pk}^e_{\mathcal{C}}, c^e])$. Instead of using a message authentication code we use a derived key under the KDF directly, inserting sid into the key derivation step instead, and taking advantage of the pseudorandomness of the KDF. This method has been suggested for example in [25] for key confirmation. In principle, one could also use $\mathsf{k}_{\mathtt{CNF}}$ in a message authentication code, applied to some public input.

Terminal Authentication The terminal $\mathcal{T}$ initiates the TA protocol by sending the chip its certificate $\mathsf{cert}_{\mathcal{T}}$, which contains its public key $\mathsf{pk}_{\mathcal{T}}$, a signature over $\mathsf{pk}_{\mathcal{T}}$ and its Document Verifier certificate. The certificates can be validated by the chip with its CVCA certificate. If the verification of certificates was successful, the chip chooses a nonce r_1 uniformly at random and sends it to $\mathcal{T}$, along with $\mathrm{ID}_{\mathcal{C}}$. Subsequently, $\mathcal{T}$ signs r_1 and $\mathrm{ID}_{\mathcal{C}}$, and sends the signature to $\mathcal{C}$. The TA protocol completes successfully if the chip can verify the signature using $\mathsf{pk}_{\mathcal{T}}$. We note that for the authentication of the terminal only, the Terminal Authentication protocol may be run solely, without the subsequent Chip Authentication. The Chip Authentication requires the (valid) public key $\mathsf{pk}_{\mathcal{T}}$ obtained during Terminal Authentication.

Chip Authentication While certificates are usually checked during Passive Authentication before EAC starts, here we move this part of the process into the CA to simplify the presentation. $\mathcal{C}$ sends its Document Signer certificate, its SO_D and its public key (all of which we will call $\mathsf{cert}_{\mathcal{C}}$) to $\mathcal{T}$. The terminal can then verify the certificate chain using the CSCA root certificate. In addition, $\mathcal{C}$ sends a n-bit number r_2, chosen uniformly at random.

In order to check that $\mathcal{C}$ is in possession of the secret key $\mathsf{sk}_{\mathcal{C}}$ pertaining to $\mathsf{pk}_{\mathcal{C}}$, the terminal uses $\mathsf{pk}_{\mathcal{C}}$ to encapsulate a shared key k in a ciphertext c. It then sends c to $\mathcal{C}$, along with a signature to prove integrity of the encapsulation. Next, $\mathcal{C}$ verifies the signature and decapsulates c to obtain k. k is then used along with the session id sid to derive a confirmation key $\mathsf{k}_{\mathrm{CNF}}$ and an encryption key $\mathsf{k}_{\mathtt{KEY}}$. The final value $\mathsf{k}_{\mathrm{CNF}}$ serves as an authentication tag for all values in sid.

Forward secrecy As is, the plain version of $\mathsf{SigPQEAC}$ provides forward secrecy for the case that only the terminal's long term keys are corrupted. However, an adversary would still be able to decrypt past communication after corrupting the chip. To achieve real forward secrecy, we propose several changes to the CA protocol, written as [ ].

In a nutshell, the chip generates an additional, ephemeral key-pair $(\mathsf{pk}_{\mathcal{C}}^{\mathsf{e}}, \mathsf{sk}_{\mathcal{C}}^{\mathsf{e}})$ and sends the public key to $\mathcal{T}$. The terminal then uses $\mathsf{pk}_{\mathcal{C}}^{\mathsf{e}}$ to encapsulate a second key k^{e}, and sends the encapsulation c^{e} back to $\mathcal{C}$, along with the encapsulation c of the first key and a signature over both encapsulations. Both keys, k and k^{e}, serve as input to derive $\mathsf{k}_{\mathrm{CNF}}$ and $\mathsf{k}_{\mathtt{KEY}}$. This way, an attacker that compromises the chip after EAC and subsequent communication has terminated will not be able to recover the ephemeral key k^{e} of that session or other previous sessions, and therefore will not be able to generate the session key $\mathsf{k}_{\mathtt{KEY}}$.

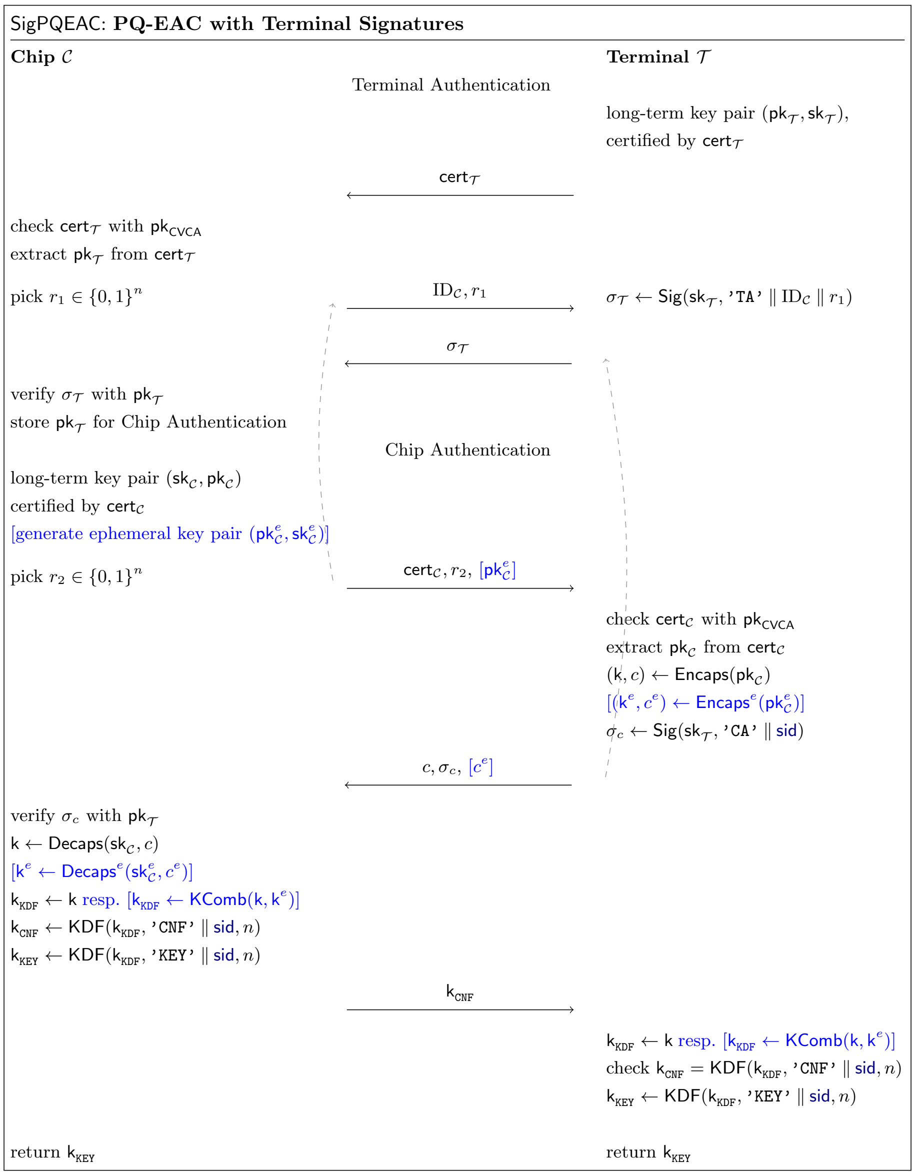
One potential attack that is enabled by sending the ephemeral public key to $\mathcal{T}$, is that an active adversary could exchange $\mathsf{pk}_{\mathcal{C}}^{\mathsf{e}}$ by a different public key and bring himself in possession of k^{e}, which is half of the input required to generate $\mathsf{k}_{\mathtt{KEY}}$. However, since both keys $\mathsf{k}, \mathsf{k}^{\mathsf{e}}$ are combined using a dual pseudorandom key combiner, the adversary still only has negligible chance of guessing the correct session key.

Round-reduced Combined Version At the cost of sacrificing some privacy it is easy to improve efficiency of the protocol described above. In particular, we can combine two pairs of messages into one message each: Instead of exchanging a challenge and a signature during TA as well as during CA, the terminal can encapsulate a key, sign a concatenation of the session id sid and the encapsulation and send the signature and the encapsulation to the chip in one go. The resulting combined variant of PQ-EAC is shown with the dashed arrows indicating which protocol messages can be combined.

As shown in Section 5.1, saving two messages and the signature verification improves performance significantly. On the flip side, the combined version forces the chip to send its public key before the terminal has been authenticated. Even though the public key is usually considered public information, it is desirable to avoid sending it to unauthorized terminals to protect the MRTD holder's privacy as much as possible. This could be important in contexts where an execution of PACE does not already authorize the terminal to read data.

4.3 Authentication via Long-Term KEMs

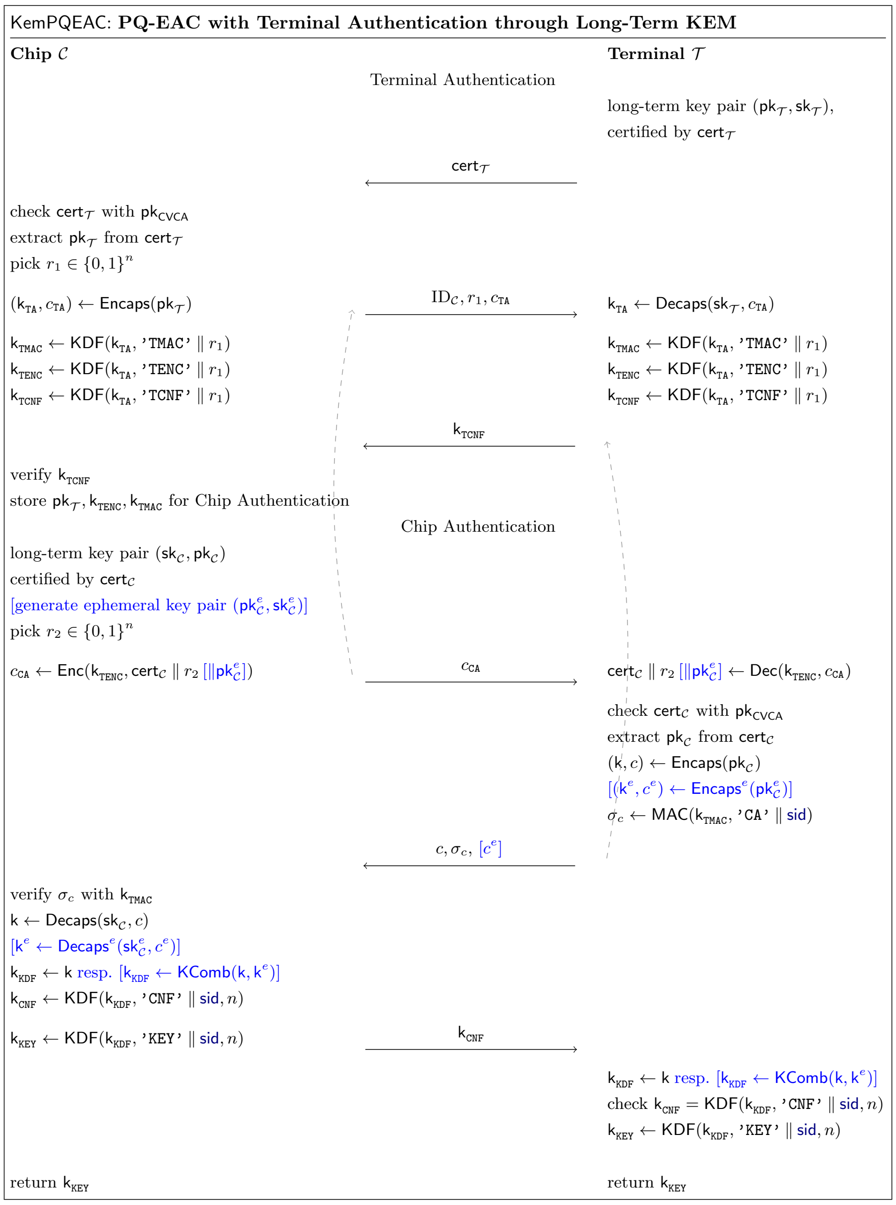
Next we present in Figure 4 the four variants in which the terminal avoids signatures, which are usually expensive, but uses a long-term key of a KEM to authenticate instead. To this end, the chip sends a key $\mathsf{k}_{\mathtt{TA}}$ encapsulated under the terminal's long-term key. While the chip has to perform an additional key encapsulation to do so, this is less expensive than the signature verification it would have to perform normally. The key is used to derive a confirmation value $\mathsf{k}_{\mathtt{TCNF}}$, replacing the signature in terminal authentication, and a MAC key $\mathsf{k}_{\mathtt{TMAC}}$ used instead of the second signature. A noteworthy feature of these versions is that now the certificate $\mathsf{cert}_{\mathcal{C}}$ can be sent encrypted in the CA step to hide the long-term identity of the card. The key $\mathsf{k}_{\mathtt{TENC}}$ for this encryption is also generated in the TA phase with the help of the terminal's long-term KEM.



SigPQEAC: **PQ-EAC with Terminal Signatures**

Chip $\mathcal{C}$		**Terminal $\mathcal{T}$**
	Terminal Authentication	
		long-term key pair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$, certified by $\mathsf{cert}_{\mathcal{T}}$
	$\xleftarrow{\mathsf{cert}_{\mathcal{T}}}$	
check $\mathsf{cert}_{\mathcal{T}}$ with $\mathsf{pk}_{\mathsf{CVCA}}$ extract $\mathsf{pk}_{\mathcal{T}}$ from $\mathsf{cert}_{\mathcal{T}}$		
pick $r_1 \in \{0,1\}^n$	$\xrightarrow{\mathrm{ID}_{\mathcal{C}}, r_1}$	$\sigma_{\mathcal{T}} \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'TA'} \parallel \mathrm{ID}_{\mathcal{C}} \parallel r_1)$
	$\xleftarrow{\sigma_{\mathcal{T}}}$	
verify $\sigma_{\mathcal{T}}$ with $\mathsf{pk}_{\mathcal{T}}$ store $\mathsf{pk}_{\mathcal{T}}$ for Chip Authentication		
	Chip Authentication	
long-term key pair $(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}})$ certified by $\mathsf{cert}_{\mathcal{C}}$ [generate ephemeral key pair $(\mathsf{pk}^e_{\mathcal{C}}, \mathsf{sk}^e_{\mathcal{C}})$]		
pick $r_2 \in \{0,1\}^n$	$\xrightarrow{\mathsf{cert}_{\mathcal{C}}, r_2, [\mathsf{pk}^e_{\mathcal{C}}]}$	
		check $\mathsf{cert}_{\mathcal{C}}$ with $\mathsf{pk}_{\mathsf{CVCA}}$ extract $\mathsf{pk}_{\mathcal{C}}$ from $\mathsf{cert}_{\mathcal{C}}$ $(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk}_{\mathcal{C}})$ $[(\mathsf{k}^e, c^e) \leftarrow \mathsf{Encaps}^e(\mathsf{pk}^e_{\mathcal{C}})]$ $\sigma_c \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'CA'} \parallel \mathsf{sid})$
	$\xleftarrow{c, \sigma_c, [c^e]}$	
verify σ_c with $\mathsf{pk}_{\mathcal{T}}$ $\mathsf{k} \leftarrow \mathsf{Decaps}(\mathsf{sk}_{\mathcal{C}}, c)$ $[\mathsf{k}^e \leftarrow \mathsf{Decaps}^e(\mathsf{sk}^e_{\mathcal{C}}, c^e)]$ $\mathsf{k}_{\mathtt{KDF}} \leftarrow \mathsf{k}$ resp. $[\mathsf{k}_{\mathtt{KDF}} \leftarrow \mathsf{KComb}(\mathsf{k}, \mathsf{k}^e)]$ $\mathsf{k}_{\mathtt{CNF}} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}}, \texttt{'CNF'} \parallel \mathsf{sid}, n)$ $\mathsf{k}_{\mathtt{KEY}} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}}, \texttt{'KEY'} \parallel \mathsf{sid}, n)$		
	$\xrightarrow{\mathsf{k}_{\mathtt{CNF}}}$	
		$\mathsf{k}_{\mathtt{KDF}} \leftarrow \mathsf{k}$ resp. $[\mathsf{k}_{\mathtt{KDF}} \leftarrow \mathsf{KComb}(\mathsf{k}, \mathsf{k}^e)]$ check $\mathsf{k}_{\mathtt{CNF}} = \mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}}, \texttt{'CNF'} \parallel \mathsf{sid}, n)$ $\mathsf{k}_{\mathtt{KEY}} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}}, \texttt{'KEY'} \parallel \mathsf{sid}, n)$
return $\mathsf{k}_{\mathtt{KEY}}$		return $\mathsf{k}_{\mathtt{KEY}}$

Fig. 3: PQ-EAC with terminal signatures, divided into TA and CA phase. Optional steps for achieving forward security are given in blue and brackets [ ]. In the combined version we can save a full round trip by combining protocol messages as indicated by the arrows. We note that each party sets the partner identity pid to be the distinguished name in the received certificate. The session identifier consists of all sent data, except for the authentication parts, $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$.



KemPQEAC: **PQ-EAC with Terminal Authentication through Long-Term KEM**
Chip $\mathcal{C}$
Terminal $\mathcal{T}$
Terminal Authentication
long-term key pair $(\mathsf{pk}_\mathcal{T}, \mathsf{sk}_\mathcal{T})$,
certified by $\mathsf{cert}_\mathcal{T}$
$\mathsf{cert}_\mathcal{T}$
check $\mathsf{cert}_\mathcal{T}$ with $\mathsf{pk}_\mathsf{CVCA}$
extract $\mathsf{pk}_\mathcal{T}$ from $\mathsf{cert}_\mathcal{T}$
pick $r_1 \in \{0,1\}^n$
$(\mathsf{k}_\mathtt{TA}, c_\mathtt{TA}) \leftarrow \mathsf{Encaps}(\mathsf{pk}_\mathcal{T})$
$\mathrm{ID}_\mathcal{C}, r_1, c_\mathtt{TA}$
$\mathsf{k}_\mathtt{TA} \leftarrow \mathsf{Decaps}(\mathsf{sk}_\mathcal{T}, c_\mathtt{TA})$
$\mathsf{k}_\mathtt{TMAC} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TMAC'} \parallel r_1)$
$\mathsf{k}_\mathtt{TENC} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TENC'} \parallel r_1)$
$\mathsf{k}_\mathtt{TCNF} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TCNF'} \parallel r_1)$
$\mathsf{k}_\mathtt{TMAC} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TMAC'} \parallel r_1)$
$\mathsf{k}_\mathtt{TENC} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TENC'} \parallel r_1)$
$\mathsf{k}_\mathtt{TCNF} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{TA}, \mathtt{'TCNF'} \parallel r_1)$
$\mathsf{k}_\mathtt{TCNF}$
verify $\mathsf{k}_\mathtt{TCNF}$
store $\mathsf{pk}_\mathcal{T}, \mathsf{k}_\mathtt{TENC}, \mathsf{k}_\mathtt{TMAC}$ for Chip Authentication
Chip Authentication
long-term key pair $(\mathsf{sk}_\mathcal{C}, \mathsf{pk}_\mathcal{C})$
certified by $\mathsf{cert}_\mathcal{C}$
[generate ephemeral key pair $(\mathsf{pk}^e_\mathcal{C}, \mathsf{sk}^e_\mathcal{C})$]
pick $r_2 \in \{0,1\}^n$
$c_\mathtt{CA} \leftarrow \mathsf{Enc}(\mathsf{k}_\mathtt{TENC}, \mathsf{cert}_\mathcal{C} \parallel r_2\ [\parallel \mathsf{pk}^e_\mathcal{C}])$
$c_\mathtt{CA}$
$\mathsf{cert}_\mathcal{C} \parallel r_2\ [\parallel \mathsf{pk}^e_\mathcal{C}] \leftarrow \mathsf{Dec}(\mathsf{k}_\mathtt{TENC}, c_\mathtt{CA})$
check $\mathsf{cert}_\mathcal{C}$ with $\mathsf{pk}_\mathsf{CVCA}$
extract $\mathsf{pk}_\mathcal{C}$ from $\mathsf{cert}_\mathcal{C}$
$(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk}_\mathcal{C})$
$[(\mathsf{k}^e, c^e) \leftarrow \mathsf{Encaps}^e(\mathsf{pk}^e_\mathcal{C})]$
$\sigma_c \leftarrow \mathsf{MAC}(\mathsf{k}_\mathtt{TMAC}, \mathtt{'CA'} \parallel \mathsf{sid})$
$c, \sigma_c, [c^e]$
verify σ_c with $\mathsf{k}_\mathtt{TMAC}$
$\mathsf{k} \leftarrow \mathsf{Decaps}(\mathsf{sk}_\mathcal{C}, c)$
$[\mathsf{k}^e \leftarrow \mathsf{Decaps}^e(\mathsf{sk}^e_\mathcal{C}, c^e)]$
$\mathsf{k}_\mathtt{KDF} \leftarrow \mathsf{k}$ resp. $[\mathsf{k}_\mathtt{KDF} \leftarrow \mathsf{KComb}(\mathsf{k}, \mathsf{k}^e)]$
$\mathsf{k}_\mathtt{CNF} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{KDF}, \mathtt{'CNF'} \parallel \mathsf{sid}, n)$
$\mathsf{k}_\mathtt{KEY} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{KDF}, \mathtt{'KEY'} \parallel \mathsf{sid}, n)$
$\mathsf{k}_\mathtt{CNF}$
$\mathsf{k}_\mathtt{KDF} \leftarrow \mathsf{k}$ resp. $[\mathsf{k}_\mathtt{KDF} \leftarrow \mathsf{KComb}(\mathsf{k}, \mathsf{k}^e)]$
check $\mathsf{k}_\mathtt{CNF} = \mathsf{KDF}(\mathsf{k}_\mathtt{KDF}, \mathtt{'CNF'} \parallel \mathsf{sid}, n)$
$\mathsf{k}_\mathtt{KEY} \leftarrow \mathsf{KDF}(\mathsf{k}_\mathtt{KDF}, \mathtt{'KEY'} \parallel \mathsf{sid}, n)$
return $\mathsf{k}_\mathtt{KEY}$
return $\mathsf{k}_\mathtt{KEY}$

Fig. 4: PQ-EAC with terminal authentication through long-term KEMs, divided into TA and CA phase. Optional steps for achieving forward security are given in blue and brackets [ ]. In the combined version we can save a full round trip by combining protocol messages as indicated by the arrows.

4.4 Instantiation with Hybrid Schemes

The push for quantum-resistant identity documents is subject to two opposed constraints: On the one hand, the demand to secure transactions from the threat posed by quantum computers and the long validity of MRTDs make it imperative to take care of post-quantum security in the short term. On the other hand, one cannot be truly confident in the concrete security of post-quantum schemes yet: Parameter choices for post-quantum schemes might not yet be reliable [63] and evolving cryptanalysis could show them to be vulnerable even to classical attacks [45,21]. Hybrid schemes[3] offer a way out: they combine two or more algorithms of the same kind such that the combined scheme provides security as long as one of the components provides security.

Of course, if the attacker is in possession of a quantum computer and the post-quantum scheme is flawed, security can only be restored by repair- or replacement of the post-quantum cryptographic scheme. However, in the case that the post-quantum scheme turns out to be insecure against classical attacks, but no quantum computer is available yet, hybrid schemes can thwart attacks by falling back on the security of the classical scheme. In Appendix B we present combiner schemes, which are a drop-in solution to achieve hybrid security. The PQ-EAC protocols defined above achieve hybrid security if they are instantiated with hybrid KEMs and signature schemes. Our implementation as described in Section 5 is using non-hybrid quantum-resistant schemes.

4.5 Security Proofs

A security proof for PQ-EAC in the real-or-random security model of Bellare and Rogaway [9] can be found in Appendix C. Below we provide a short sketch of the proof. The security properties to show are session matching and key secrecy. Session matching covers fundamental properties such as partnered sessions deriving the same key, pairwise uniqueness of partners, and correct identification of roles and partners. For all protocol versions ($\mathsf{SigPQEAC}$ and $\mathsf{KemPQEAC}$, with or without forward secrecy, with or without round reduction) this property follows by protocol construction and the uniqueness of nonces. We note that all proofs take into account potential decryption errors of KEMs. Key secrecy ensures that only the intended partner shares the derived session key and that session keys are indistinguishable from random to the adversary, unless the adversary trivially knows the key. The proof for key secrecy for the variants of $\mathsf{SigPQEAC}$ is via game-hopping. Here, we only outline the main steps. We first exclude attacks in which a party accepts a different public key pk as identified in the issued certificate cert. By the unforgeability of the certificate scheme this cannot occur, except with negligible probability. But it follows that, if the chip accepts only the certified public key of the terminal, then the unforgeability of

[3] The notion of hybrid schemes is also used to refer to schemes that combine asymmetric and symmetric primitives, e.g. the *Integrated Encryption Scheme*. However, in the context of post-quantum cryptography, it denotes the combination of classical and post-quantum algorithms.

the terminal's signature scheme ensures that only the honest terminal is able to create the valid signature σ_{c} over the ciphertext c (and the ephemeral values $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ and c^{e} in the forward-secure version). We can conclude that the encapsulated key k (resp. keys k and k^{e}) must have been created by an honest terminal, such that the IND-CCA security of the chip's long-term KEM guarantees that they are hidden from the adversary. The security of the key derivation function (together with the key combiner in the forward-secure version) ensure that the session key is indistinguishable from random for the adversary.

We note that in the analysis the final confirmation value sent by the chip is only required for the forward-secure case. Here the adversary may impersonate a chip and later learn the long-term secret key of the KEM. But the adversary would have to create a valid confirmation value *before* it later corrupts the chip's long-term secret. This once more is infeasible by the security of the chip's KEM.

The proof of the KEM-based variant KemPQEAC is very similar to the signature-based protocol. Only here we need to argue that only the honest terminal can create the valid MAC σ_{c} over the ciphertext(s). This can be shown via two extra steps in which we argue that only the terminal can decapsulate $\mathsf{k}_{\mathtt{TA}}$ in terminal authentication (by the IND-CCA security of the terminal's long-term KEM) and that the derived keys from $\mathsf{k}_{\mathtt{TA}}$ are random by the security of the key derivation function. The other steps are as in the proof of SigPQEAC.

We finally remark that our proofs show post-quantum security of the protocols. The proof steps consist of (straightline) reductions to the involved standard primitives like signature schemes, KEMs, or key derivation functions, and do not require idealized primitives such as random oracles. Hence, any successful quantum adversary against the key exchange protocol would thus yield a successful quantum adversary against the underlying primitive. Assuming that the primitives are all quantum-resistant, it follows that the overall protocol also withstands such attacks.

5 Implementation

To show the feasibility of PQ-EAC for border control systems and to evaluate the chip-side of the protocol proposals in resource-constrained environments, we implemented the combined version of SigPQEAC (without forward secrecy) on a chip similar to those deployed in ePassports and on a terminal of the type that is used for border control checks. The chip is integrated in a proof of concept post-quantum MRTD and runs an ePassport application compliant with ICAO standards. Our performance tests reveal that the post-quantum version of EAC is practical and only slightly slower than the classic version. We published the benchmarking tool that we used to measure the performance of SigPQEAC on a smart card[4]. Other components utilized for benchmarking such as the VISOCORE® terminal software, the ePassport application and our customized Dilithium and Kyber implementations are part of confidential proprietary software libraries and remain unpublished.

[4] https://github.com/frankmorgner/OpenSC-pqc-SSR2023

We designed the Application Protocol Data Unit (APDU) interface for reading files or initiating cryptographic operations on the chip to be compatible with the existing standards of ICAO, BSI, and ISO [33,18,34,35]. For this reason, our implementation uses more than the minimal number of messages possible. We proceed by stating the results of our runtime experiments with our implementation in Section 5.1. Finally, we analyze the impact of data transmission rates on the performance of the protocol in Section 5.2. Since transmission speeds vary between platforms, and because incorrect placement of the MRTD can slow transmission rates significantly, this analysis will help to gauge the real-world performance of PQ-EAC more precisely.

5.1 Performance Evaluation

In our experimental setup we instantiated Combined PQ-EAC without forward secrecy with the post-quantum signature scheme Dilithium3 [44] and the post-quantum KEM Kyber1024 [58,15]. NIST's security level 3 (for the signature scheme) and 5 (for the KEM), respectively, were chosen to support the MRTD's long lifetime of typically ten years. AES256 and SHA256 were used as supporting cryptographic building blocks for the message authentication codes and key derivation functions. In our implementation we assume that the certificates are based on post-quantum signatures. For the sake of data minimization we stick to a single post-quantum signature and public key per certificate.

Chip Specification A proof of concept post-quantum MRTD was implemented on a a contactless security controller from Infineon Technologies based on an ARM SC300 architecture. The security controller comes with significantly increased RAM-size of 96 kBytes, which is more than double the amount of RAM compared to previous security controller chips in this application domain, to support memory-intensive lattice-based schemes such as Kyber and Dilithium.

Chip Software We implemented Kyber and Dilithium in C according to specifications given in the NIST PQC competition round 3 submissions [58,44], taking into account the requirements posed by the hardware architecture of our security controller. In particular, memory requirements had to be carefully managed: Our implementation utilizes almost the full RAM capacity of 96 kBytes. However, not all memory optimization opportunities have been exploited by us: By assigning overlapping memory segments to public-key and symmetric cryptographic operations, it would be possible to reduce memory usage to ca. 60 kBytes. Other than memory optimizations, our versions stay close to the mostly unoptimized reference implementation. Since the focus of our work has been the creation of a functional proof of concept for PQ-EAC, most cryptographic building blocks have not been optimized for performance or hardened against side channel analysis (SCA) and fault attacks (FA). To improve the security of Kyber and Dilithium against SCA and FA the strategies described in Oder et al. [51], Saarinen [56], Ravi et al. [54], Bache et al. [6] or Heinz et al. [28] could be used.

Besides a hardware-based random number generator, no acceleration has been used in the project. The chip's ePassport application is running as native code on the hardware without an intermediate operating system. All state management (which needs to carefully consider hardware limitations around the volatile and non-volatile memory) is done in that application. For communication with the terminal we utilize a proprietary ISO/IEC 14443 communication library.

Terminal Implementation For demonstration purposes, we modified terminal hard- and software of the type that is used by German border control with post-quantum algorithms. Specifically, we used a Bundesdruckerei VISOTEC® Expert 800[5] running the verification software VISOTEC® Inspect, which was extended by us with the same post-quantum schemes as the MRTD.

Benchmarking We ran 1000 experiment executions with the test chip to benchmark the performance of the combined version of SigPQEAC. We measured the time it took to send and receive the card commands via the PC/SC API on the terminal's operating system. Thus, this duration includes the terminal's overhead of processing the data with the smart card reader's driver, its firmware, transceiving data via ISO 14443, and finally the processing by the test chip. For PQ-EAC, we reliably measured a total runtime of 1.28 seconds. The standard deviation for a single command-response pair was between 0.0001 and 0.0003 milliseconds.

In a second experiment we used the same setup with a smart card emulation device (Hitex Tanto3 FPGA) instead of the MRTD. This device emulates all electrical properties of the MRTD's chip and allows control and inspection of the card's internal workflow. This way we were able to determine or manually control the connection parameters between reader and card. Specifically, we set the data rate to 848 kBit/s and controlled the frame size for sending 4089 bits of payload data. The emulated chip was constantly run with 100MHz and didn't suffer any throttling due to, for example, bad coupling with the smart card reader (a common problem in practice).

The total time for performing all PQ-EAC operations on the emulated chip and the data transfer is 1.28 seconds. In Figure 5 we can see that the execution time on the chip is heavily dominated by the cryptographic operations during signature validation and KEM decapsulation. As such, Kyber decapsulation takes 596 ms, Dilithium verification takes 86 ms, but choosing a nonce and reading $\mathsf{cert}_{\mathcal{C}}$ only take 0.01 ms each.

The rest of the transaction time is caused by transferring the data back and forth between terminal and card. Again, most of this can be attributed to the large ciphertexts, public keys, and signatures of Dilithium and Kyber. For example, $\mathsf{cert}_{\mathcal{T}}$ contains a a Dilithium signature (2701 bytes) and the terminal's Dilithium public key (1472 bytes) which add up to 97.6% of the overall size of the certificate. Transferring this data with a data rate R_b of 848 kBit/s takes roughly 50 ms including the ISO/IEC 14443 overhead.

[5] https://www.bundesdruckerei-gmbh.de/en/solutions/visocore

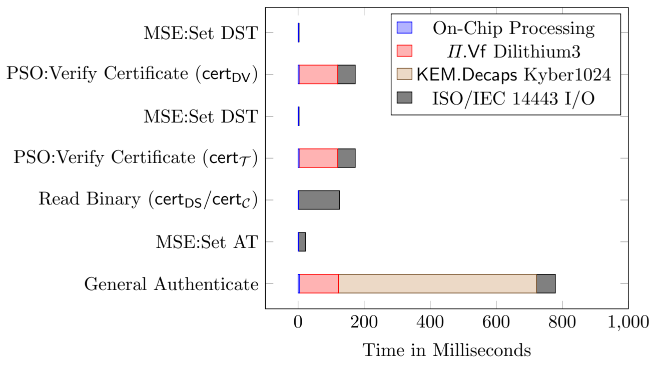


Fig. 5: Measurements of combined SigPQEAC with 848kBit/s via ISO/IEC 14443 using an emulated chip.

Our analysis shows that the processing time on the chip is almost exclusively caused by the the post-quantum algorithms. Further, we observe that in our proof of concept implementation each of encapsulation, decapsulation and key generation take around five times longer than verifying a signature. However, the literature suggests that a better optimized implementation on similar hardware speeds up the Kyber1024 encapsulation by an order of magnitude to half the execution time of the signature verification of Dilithium3 [36]. Such an improvement would most likely allow an overall execution time for PQ-EAC of one second or below.

5.2 Impact of the Data Transfer Rate

We observed that the post-quantum algorithms are causing most of the data that needs to be transceived between MRTD and chip. Using our experimental data we can confirm that the (extended) ISO/IEC 14443 I-Block framing is causing an overhead of roughly 26% of the transmitted protocol data[6]. Also, we observe that the constant overhead for encoding the data into command and response APDUs varies between 9 and 35 bytes. This observation allows to approximate the runtime of other variants of PQ-EAC in relation to data transfer rates.

Using the measurements from our experiments with the test chip and the estimated communication load, we present the estimated runtime for the protocols relative to different transmission data rates in Table 1. Mostly due to the fact that it transfers less data and uses one less signature verification, the combined version of SigPQEAC achieves the shortest runtime. The advantage of KemPQEAC under low data rates is due to the fact that no signatures, which are

[6] The ISO/IEC 14443 overhead is between 23% for very big APDUs and 72% for very small APDUs. Since most of the EAC protocols' runtime is spent on big commands, we stick to an approximation near that of bigger commands.

	Estimated Runtime (in milliseconds)		
Data Rate (R_b)	SigPQEAC	Combined SigPQEAC	KemPQEAC
848 kbit/s	1406	1280	1569
424 kbit/s	1718	1530	1754
106 kbit/s	3586	3204	2865

Table 1: Estimation for the protocol variants for typical transmission rates. For details on PQ-EAC Combined at 848 kbit/s see Figure 5.

typically larger than ciphertexts or keys, need to be sent. The slow key encapsulation of the KEM in our proof-of-concept implementation, however, causes it to be the slowest protocol for higher data transfer rates. Given that benchmarks [36] show that KEM encapsulation is usually much faster than signature verification, runtime of PQ-EAC and especially of KemPQEAC should significantly benefit from further optimizations of the implementation.

Comparison with Classical EAC An execution of classical EAC on MRTDs and terminals commonly in use today typically takes around 1.5 to 2 seconds. Noting that most deployed electronic passports and verification terminals are already supporting 848 kbit/s data rate, our proof-of-concept implementation for PQ-EAC with well below 2 seconds execution time supports the conclusion that eMRTDs can be migrated to quantum-resistant algorithms. Considering performance gains from hardware acceleration and general implementation improvements we assume that instantiations with hybrid schemes are also feasible.

6 Conclusions

In the preceding sections we have presented and implemented quantum-resistant versions of the EAC protocol. Our results show that post-quantum travel documents are practical. Moreover, our implementation of PQ-EAC can still be optimized. It can also easily be instantiated with alternative KEMs and signature schemes. Below, we document a few opportunities for future research.

Post-quantum PKI and PACE In our protocols we assumed that the authenticity of the chip and terminal are secured by quantum-resistant certificates: A quantum-resistant PKI that will provide such certificates is a challenge to be addressed by future research. The transition towards a post-quantum PKI could potentially be conducted by means of intelligent composed algorithms as described in Byszio et al. [20]. Similarly, we assumed that the initial connection between chip and terminal is secured via PACE. As current versions of PACE are based on classical assumptions, there is need for a quantum-resistant construction that can secure the transmission of less sensitive data groups. Another option would be to simply do without PACE and instead mandate an execution of EAC for all requests of chip data, not only for sensitive data groups. This solution would provide the same security guarantees as a sequential execution of PACE and EAC, and additionally rectify a current privacy issue that arises when only PACE is performed: Since Passive Authentication requires the chip to send a list of hashes of all data groups, it allows anyone who knows the chip's MRZ to match the hash of the fingerprint with a collection of precollected fingerprints. When EAC is mandatory, we can eliminate this privacy threat by forcing the terminal to authenticate before performing Passive Authentication.

Cryptographic Agility Another problem we leave for future research is cryptographic agility in MRTDs. In particular it is desirable to have an update mechanism that provides an authenticated update for MRTDs in the case that the deployed quantum-resistant algorithms need to be replaced. This would be the case when flaws in the used algorithms have been discovered. The challenge here is that we have a chicken-egg situation: Ultimately the authentication mechanism of the update would have to inspire more confidence than the post-quantum schemes we have instantiated EAC with. A potential candidate for such an authentication mechanism are hash-based signatures [29,46]: While they are less practical than lattice-based constructions, their security rests on weaker assumptions, namely on the one-wayness of certain hash functions.

Future Directions for MRTDs Recently ICAO has established a working group on the so called Digital Travel Credential (DTC) and published version 4.4 of 'Guiding Core Principles for the Development of Digital Travel Credential DTC' [32]. In this document the DTC is described as consisting of two parts: the DTC Physical Component (DTC-PC) and the DTC Virtual Component (DTC-VC). While a DTC-PC represents a physical MRTD, the DTC-VC could potentially be stored on a smartphone and then be used in lieu of a physical MRTD. However, at the moment it is planned to store only primary biometrics (facial image) in the DTC-VC, while secondary biometrics (fingerprint) will be exclusively stored on the DTC-PC. Future research will need to address security of DTCs, especially when coupled with smartphones.

Acknowledgements We thank the anonymous reviewers for valuable comments. This work was funded through the project PoQuID (WIPANO project 03TNK011A-C). WIPANO projects are financed by the German Federal Ministry for Economic Affairs and Energy and managed by Forschungszentrum Jülich. The authors thank all PoQuID project partners for the discussions and support.

References

1. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. Cryptology ePrint Archive (2022), https://eprint.iacr.org/2022/770

2. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D.: Status report on the third round of the nist post-quantum cryptography standardization process, nist ir 8413. Tech. rep., National Institute for Standards and Technology (NIST) (2022). https://doi.org/10.6028/NIST.IR.8413-upd1
3. Angel, Y., Dowling, B., Hülsing, A., Schwabe, P., Weber, F.: Post quantum noise. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 97–109. ACM (2022). https://doi.org/10.1145/3548606.3560577
4. Arute, F., et al.: Quantum supremacy using a programmable superconducting processor **574**(7779), 505–510. https://doi.org/10.1038/s41586-019-1666-5
5. Avoine, G., Kalach, K., Quisquater, J.J.: ePassport: Securing International Contacts with Contactless Chips. In: Tsudik, G. (ed.) Financial Cryptography and Data Security. pp. 141–155. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_11
6. Bache, F., Paglialonga, C., Oder, T., Schneider, T., Güneysu, T.: High-speed masking for polynomial comparison in lattice-based kems. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 483–507 (2020)
7. Bellare, M., Lysyanskaya, A.: Symmetric and dual prfs from standard assumptions: A generic validation of an HMAC assumption. IACR Cryptol. ePrint Arch. p. 1198 (2015), `http://eprint.iacr.org/2015/1198`
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer (1993). https://doi.org/10.1007/3-540-48329-2_21
9. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security. pp. 62–73. CCS '93, Association for Computing Machinery, Fairfax, Virginia, USA (Dec 1993). https://doi.org/10.1145/168588.168596
10. Bender, J., Dagdelen, Ö., Fischlin, M., Kügler, D.: The pace|aa protocol for machine readable travel documents, and its security. In: Keromytis, A.D. (ed.) Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7397, pp. 344–358. Springer (2012). https://doi.org/10.1007/978-3-642-32946-3_25
11. Bender, J., Fischlin, M., Kügler, D.: Security Analysis of the PACE Key-Agreement Protocol. In: Proceedings of the 12th International Conference on Information Security. pp. 33–48. ISC '09, Springer-Verlag, Berlin, Heidelberg (Sep 2009). https://doi.org/10.1007/978-3-642-04474-8_3
12. Bernstein, D.J.: Introduction to post-quantum cryptography. In: Post-quantum cryptography, pp. 1–14. Springer (2009)
13. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 206–226. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_12
14. Bindel, N., Herath, U., McKague, M., Stebila, D.: Transitioning to a quantum-resistant public key infrastructure. In: Lange, T., Takagi, T. (eds.) Post-Quantum

Cryptography. Lecture Notes in Computer Science, vol. 10346, p. 384–405. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_22

15. Botros, L., Kannwischer, M.J., Schwabe, P.: Memory-efficient high-speed implementation of kyber on cortex-m4. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11627, pp. 209–228. Springer (2019). https://doi.org/10.1007/978-3-030-23696-0_11
16. Boyd, C., Gellert, K.: A modern view on forward security. The Computer Journal **64**(1), 639–652 (2019). https://doi.org/10.1093/comjnl/bxaa104
17. Brendel, J., Fischlin, M., Günther, F., Janson, C., Stebila, D.: Towards post-quantum security for signal's X3DH handshake. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12804, pp. 404–430. Springer (2020). https://doi.org/10.1007/978-3-030-81652-0_16
18. Bundesamt für Sicherheit in der Informationstechnik: Bsi tr-03110. Standard (2016)
19. Bundesamt für Sicherheit in der Informationstechnik: Migration to Post Quantum Cryptography: Recommendations for action by the BSI (2020), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Migration_to_Post_Quantum_Cryptography.pdf?__blob=publicationFile&v=2
20. Byszio, F., Wirth, K.D., Nguyen, K.: Intelligent composed algorithms. Cryptology ePrint Archive, Paper 2021/813 (2021), https://eprint.iacr.org/2021/813
21. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive (2022)
22. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An Efficient Post-Quantum Commutative Group Action. In: Advances in Cryptology – ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III. pp. 395–427. Springer-Verlag, Berlin, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
23. Dagdelen, Ö., Fischlin, M.: Security analysis of the extended access control protocol for machine readable travel documents. In: Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.) Information Security - 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6531, pp. 54–68. Springer (2010). https://doi.org/10.1007/978-3-642-18178-8_6
24. Filimonov, I., Horne, R., Mauw, S., Smith, Z.: Breaking unlinkability of the icao 9303 standard for e-passports using bisimilarity. In: Computer Security – ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I. p. 577–594. Springer-Verlag, Berlin, Heidelberg (Sep 2019). https://doi.org/10.1007/978-3-030-29959-0_28
25. Fischlin, M., Günther, F., Schmidt, B., Warinschi, B.: Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 452–469. IEEE Computer Society (2016). https://doi.org/10.1109/SP.2016.34
26. Giacon, F., Heuer, F., Poettering, B.: KEM Combiners. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 190–218. Lecture

Notes in Computer Science, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_7
27. Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 516–525. CCS '10, Association for Computing Machinery, New York, NY, USA (Oct 2010). https://doi.org/10.1145/1866307.1866365
28. Heinz, D., Pöppelmann, T.: Combined fault and DPA protection for lattice-based cryptography. IACR Cryptol. ePrint Arch. p. 101 (2021), https://eprint.iacr.org/2021/101
29. Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. https://doi.org/10.17487/RFC8391
30. Hulsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS+. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
31. Hülsing, A., Ning, K., Schwabe, P., Weber, F., Zimmermann, P.R.: Post-quantum wireguard. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 304–321. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00030
32. International Civil Aviation Organization: Guiding Core Principles for the Development of Digital Travel Credential (DTC) (10 2020), https://www.icao.int/Security/FAL/TRIP/PublishingImages/Pages/Publications/Guiding%20core%20principles%20for%20the%20development%20of%20a%20Digital%20Travel%20Credential%20%20%28DTC%29.PDF
33. International Civil Aviation Organization: Icao doc 9303. Standard (2021), https://www.icao.int/publications/pages/publication.aspx?docnum=9303, 8th Edition
34. International Organization for Standardization / International Electrotechnical Commission: Iso/iec 14443-4: Identification cards – contactless integrated circuit cards – proximity cards. Standard (6 2018)
35. International Organization for Standardization / International Electrotechnical Commission: Iso/iec 7816-4: Identification cards – integrated circuit cards. Tech. rep. (5 2020)
36. Kannwischer, M.J., others: Pqm4 (2022), https://github.com/mupq/pqm4/blob/master/benchmarks.md
37. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014), https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269
38. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. p. 636–652. Lecture Notes in Computer Science, Springer (2009). https://doi.org/10.1007/978-3-642-10366-7_37
39. Krawczyk, H.: SIGMA: the 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 400–425. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_24

40. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 631–648. Springer (2010). https://doi.org/10.1007/978-3-642-14623-7_34
41. Kußmaul, F.: Armed for the quantum revolution: Implementing post-quantum-cryptography on the goid card. In: für Sicherheit in der Informationstechnik, B. (ed.) IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung. pp. 275–284. SecuMedia Verlags-GmbH
42. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (Oct 1979)
43. Liu, Y., Kasper, T., Lemke-Rust, K., Paar, C.: E-Passport: Cracking Basic Access Control Keys. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. pp. 1531–1547. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76843-2_30
44. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`
45. MATZOV: Report on the Security of LWE: Improved Dual Lattice Attack (Apr 2022). https://doi.org/10.5281/zenodo.6493704
46. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali Hash-Based Signatures. https://doi.org/10.17487/RFC8554
47. Merkle, R.C.: Secrecy, Authentication and Public Key Systems. Ph.D. thesis (1979), `https://www.merkle.com/papers/Thesis1979.pdf`
48. Morgner, F., von der Heyden, J.: Analyzing requirements for post quantum secure machine readable travel documents. In: Roßnagel, H., Schunck, C.H., Mödersheim, S. (eds.) Open Identity Summit 2021. pp. 205–210. Gesellschaft für Informatik e.V., Bonn (2021)
49. Mosca, M.: Cybersecurity in an era with quantum computers: will we be ready? Cryptology ePrint Archive, Paper 2015/1075 (2015), `https://eprint.iacr.org/2015/1075`
50. National Institute of Standards and Technology (NIST): Recommendation for stateful hash-based signature schemes, sp 800-208. Standard (2020). https://doi.org/10.6028/NIST.SP.800-208
51. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. IACR TCHES **2018**(1), 142–174 (2018)
52. Pradel, G., Mitchell, C.: Post-quantum certificates for electronic travel documents. In: Proceedings of DETIPS 2020 (Interdisciplinary Workshop on Trust, Identity, Privacy, and Security in the Digital Economy), September 18 2020. pp. 56–73. Lecture Notes in Computer Science, Springer (Dec 2020). https://doi.org/10.1007/978-3-030-66504-3_4
53. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`
54. Ravi, P., Poussier, R., Bhasin, S., Chattopadhyay, A.: On configurable SCA countermeasures against single trace attacks for the NTT - A performance evaluation study over kyber and dilithium on the ARM cortex-m4. In: SPACE. Lecture Notes in Computer Science, vol. 12586, pp. 123–146. Springer (2020)

55. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446, https://www.rfc-editor.org/info/rfc8446
56. Saarinen, M.O.: Arithmetic coding and blinding countermeasures for lattice signatures - engineering a side-channel resistant post-quantum signature scheme with compact signatures. J. Cryptogr. Eng. **8**(1), 71–84 (2018)
57. Schanck, J.M., Stebila, D.: A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret. Internet-Draft draft-schanck-tls-additional-keyshare-00, Internet Engineering Task Force (Apr 2017), https://datatracker.ietf.org/doc/draft-schanck-tls-additional-keyshare/00/, work in Progress
58. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
59. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1461–1480. ACM (2020). https://doi.org/10.1145/3372297.3423350
60. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press (11 1994). https://doi.org/10.1109/SFCS.1994.365700
61. Vogt, S., Funke, H.: How quantum computers threat security of pkis and thus eids. In: Roßnagel, H., Schunck, C.H., Mödersheim, S. (eds.) Open Identity Summit 2021. pp. 83–94. Gesellschaft für Informatik e.V., Bonn (2021)
62. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences **22**(3), 265–279 (Jun 1981). https://doi.org/10.1016/0022-0000(81)90033-7
63. Wenger, E., Chen, M., Charton, F., Lauter, K.: Salsa: Attacking lattice cryptography with transformers. Cryptology ePrint Archive, Paper 2022/935 (2022), https://eprint.iacr.org/2022/935

A Security Definitions

In this section we introduce the underlying primitives and their security notions for building the key exchange protocols as defined in, for example, [37].

A.1 Key Encapsulation

Definition 1 (Key Encapsulation Mechanism). *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *consists of three efficient algorithms where:*

Key Generation: *Algorithm* KeyGen *on input the security parameter* 1^n *(in unary) outputs a key pair,* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^n)$*. We assume that* 1^n *is recoverable from either key.*

Encapsulation: *The encapsulation algorithm takes as input a public key* pk *and returns a ciphertext and a key,* $(c, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$*. We assume usually that the key is of bit length* n*.*

Decapsulation: *The decapsulation algorithm takes as input a secret key* sk *and a ciphertext* c*, and returns a key or an error symbol,* $\mathsf{k} \leftarrow \mathsf{Decaps}(\mathsf{sk}, c)$*, where* k *is either of size* n *or equals* $\perp$*. Usually decapsulation is deterministic.*

We require that decapsulation merely has a negligible error. That is, we denote by $\Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(n) = 1\right]$ *the probability of an encryption error for* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$*, where* $\mathsf{Decaps}(\mathsf{sk}, c) \neq \mathsf{k}$ *for* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(n)$ *and* $(c, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$*.*

We next define CPA- and CCA-security for key encapsulation mechanism in one go:

Definition 2 (IND-CPA and IND-CCA security of KEM). *For a key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *and adversary* $\mathcal{A}$ *define experiment* $\boldsymbol{Exp}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(n)$ *as in Figure 6. We say that* KEM *is IND-att secure (for att=CPA or CCA) if for any efficient adversary* $\mathcal{A}$ *we have that*

$$\boldsymbol{Adv}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(n) := \Pr\left[\boldsymbol{Exp}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(n) = 1\right] - \frac{1}{2}$$

is negligible.

```
Experiment $\mathbf{Exp}^{\text{IND-att}}_{\mathsf{KEM},\mathcal{A}}(n)$
1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^n)$
2 : $b \leftarrow \{0,1\}$
3 : $(\mathsf{k}_0^*, c^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
4 : $\mathsf{k}_1^* \leftarrow \{0,1\}^{|\mathsf{k}_0^*|}$
5 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Decaps}}(\cdot)}(\mathsf{pk}, \mathsf{k}_b^*, c^*)$
6 : return $[b = b^*]$
```

```
Oracle $\mathcal{O}_{\mathsf{Decaps}}(c)$
1 : if $c = c^*$ or att=CPA then
2 :   return $\perp$
3 : else
4 :   return $\mathsf{Decaps}(\mathsf{sk}, c)$
```

Fig. 6: IND-CPA/IND-CCA security experiment for key encapsulation mechanism

For our key exchange protocol $\mathsf{KemPQEAC}$ we also use a symmetric encryption scheme (for keys k from $\{0,1\}^n$) where $c \leftarrow \mathsf{Enc}(\mathsf{k}, m)$ creates a ciphertext, and $m \leftarrow \mathsf{Dec}(\mathsf{k}, c)$ always recovers the encrypted message m. In the protocol this encryption step provides extra privacy for the chip by encrypting its identity. We do not discuss this privacy property formally here and thus neither the security notions for the encryption scheme.

A.2 Message Authentication, Signature Schemes, and Certificate Schemes

We define message authentication schemes, signature schemes, and certificate schemes with a single definition. All schemes serve the purpose of authenticating data. The only difference between the private-key message authentication codes (MACs) and the public-key signature and certificate schemes lies in the verification key vk: MACs use the key $\mathsf{vk} = \mathsf{sk}$ to verify authenticity, whereas signatures and certificates are verified against the public key $\mathsf{vk} = \mathsf{pk}$. In the descriptions and security games we thus set vk accordingly, and the public key for MACs to be empty and for signatures and certificates equal to vk. We call the primitive abstractly an authentication scheme:

Definition 3 (Authenticaton Scheme). *An authentication scheme* $\mathcal{AS} = (\mathsf{AKGen}, \mathsf{AAuth}, \mathsf{AVf})$ *consists of three efficient algorithms such that:*

Key Generation: *Algorithm* AKGen *on input* 1^n *returns a key triple* $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SKGen}(1^n)$*. We assume that* n *is recoverable from either key.*

Authentication: *On input the secret key* sk *and a message* $m \in \{0,1\}^*$*, the authentication algorithm outputs an authenticator,* $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$*.*

Verification: *On input a verification key* vk*, a message* m*, an authenticator* σ*, the verification algorithm outputs a decision bit,* $d \leftarrow \mathsf{AVf}(\mathsf{vk}, m, \sigma)$*. Usually,* AVf *is deterministic.*

We require that verification always succeeds. That is, it never happens that $\mathsf{AVf}(\mathsf{sk}, m, \sigma) = 0$ *for any* $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SKGen}(1^n)$*, any* $m \in \{0,1\}^*$*, and any* $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$*.*

Unlike key encapsulation we define authentication schemes with perfect correctness since all known schemes in practice achieve this.

Definition 4 (EUF-CMA of Authentication Schemes). *For an authentication scheme* $\mathcal{AS} = (\mathsf{AKGen}, \mathsf{AAuth}, \mathsf{AVf})$ *and adversary* $\mathcal{A}$ *define experiment* $\textbf{\textit{Exp}}^{\textit{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(n)$ *as in Figure 7. We say that* $\mathcal{AS}$ *is EUF-CMA if for any efficient adversary* $\mathcal{A}$ *we have that* $\Pr\left[\textbf{\textit{Exp}}^{\textit{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(n) = 1\right]$ *is negligible.*

```
Experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(n)$
1 : $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^n)$
2 : $\mathcal{Q} \leftarrow \emptyset$
3 : $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AAuth}}(\mathsf{sk},\cdot)}(\mathsf{pk})$
4 : return $[\mathsf{SVf}(\mathsf{vk}, m^*, \sigma^*)$ and $m^* \notin \mathcal{Q}]$

Oracle $\mathcal{O}_{\mathsf{AAuth}}(\mathsf{sk}, m)$
1 : $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
2 : $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$
3 : return $\sigma$
```

Fig. 7: EUF-CMA security experiment for authentication schemes

A signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{SVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^n)$ for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SKGen}(1^n)$ and $\mathsf{vk} \leftarrow \mathsf{pk}$. A certificate scheme $\mathcal{C} = (\mathsf{CKGen}, \mathsf{CSign}, \mathsf{CVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^n)$ for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{CKGen}(1^n)$ and $\mathsf{vk} \leftarrow \mathsf{pk}$. A message authentication code $\mathcal{M} = (\mathsf{MKGen}, \mathsf{MAC}, \mathsf{MVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^n)$ for $\mathsf{sk} \leftarrow \mathsf{MKGen}(1^n)$ and $\mathsf{vk} \leftarrow \mathsf{sk}$ and $\mathsf{pk} \leftarrow \bot$. EUF-CMA security now follows from the general definition. We usually assume that the key generation algorithm simply generates a uniformly distributed key of n bits.

A.3 Key Derivation Functions

We assume that key derivation functions act as pseudorandom functions, as long as the keying material contains enough (min-)entropy. The latter is captured by considering arbitrary distributions $\mathcal{D}$ which take the seucurity parameter 1^n as input and output IKM with min-entropy $H_\infty(\mathsf{IKM}) \geq n$. We call such distributions *non-trivial*. We follow here the presentation of Krawczyk [40].

Definition 5 (Key Derivation Function). *A key derivation function* KDF *takes as input keying material* IKM*, context information* ctxt*, and an integer* ℓ*, and outputs a string of length* ℓ*. We assume that the length of* IKM *determines the security parameter* n*.*

Security now requires that the key derivation function's output for IKM looks random, even if the adversary sees actual derived keys at other inputs (ctxt, ℓ):

Definition 6 (Pseudorandomness of Key Derivation Function). *For a key derivation function* KDF*, adversary* $\mathcal{A}$*, and distribution* $\mathcal{D}$ *define experiment* $\boldsymbol{Exp}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(n)$ *as in Figure 8. We say that* KDF *is pseudorandom if for any efficient adversary* $\mathcal{A}$ *and any non-trivial distribution (with min-entropy* n*), we have that*

$$\boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(n) := \Pr\left[\boldsymbol{Exp}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(n) = 1\right] - \frac{1}{2}$$

is negligible.

A.4 Key Combiners

For the forward-secure version of the PQEAC protocols we use a static KEM and an ephemeral KEM to share keys k and k^e. In this case the parties need to derive a single key from the two keys. This has been discussed more broadly in the context of KEM combiners in [26] and for quantum adversaries in [13], but since we have already embedded the KEM mechanism in the key exchange protocol, we focus here on the pure key combining part. We note that we cannot immediately rely on the KEM combiner in [13], since it assumes faithfully created but potentially weak encapsulations, whereas in our setting the adversary may choose the encapsulations maliciously. While Bindel et al. [13] argue that such genuine encapsulations are sufficient to build hybrid authenticated key exchange protocols via Krawczyk's SIGMA compiler [39], the EAC protocol does not perfectly comply to the SIGMA standard.

```
Experiment $\mathbf{Exp}^{\mathrm{PRF}}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(n)$
1 : $\mathsf{IKM} \leftarrow \mathcal{D}(1^n)$
2 : $\mathcal{Q} \leftarrow \emptyset$
3 : $b \leftarrow \{0,1\}$
4 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KDF}}(\mathsf{IKM},\cdot,\cdot),\mathcal{O}_b(\mathsf{IKM},\cdot,\cdot)}(1^n)$
5 : return $[b = b^*]$
```

```
Oracle $\mathcal{O}_{\mathsf{KDF}}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
1 : if $(\mathsf{ctxt}, \ell) \in \mathcal{Q}$ then
2 :     return $\perp$
3 : $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{ctxt}, \ell)\}$
4 : $\mathsf{k} \leftarrow \mathsf{KDF}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
5 : return $\mathsf{k}$
```

```
Oracle $\mathcal{O}_b(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
1 : if $(\mathsf{ctxt}, \ell) \in \mathcal{Q}$ then
2 :     return $\perp$
3 : $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{ctxt}, \ell)\}$
4 : $\mathsf{k}_0 \leftarrow \mathsf{KDF}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
5 : $\mathsf{k}_1 \leftarrow \{0,1\}^\ell$
6 : return $\mathsf{k}_b$
```

Fig. 8: Pseudorandomness experiment for Key Derivation Functions

Definition 7 (Key Combiner). *A key combiner* KComb *takes as input keying material* $\mathsf{IKM}_0, \mathsf{IKM}_1$*, both of length* n*, and outputs a string of length* n*.*

Security demands that KComb is a dual pseudorandom function, meaning that both $\mathsf{KComb}(\mathsf{IKM}_0, \cdot)$ and $\mathsf{KComb}(\cdot, \mathsf{IKM}_1)$ are pseudorandom functions. It follows that we can reasonably assume that HKDF resp. HMAC [40,7] or the TLS-based nested key derivation function in [57] is an appropriate instantiation for a key combiner.

Definition 8 (Dual Pseudorandomness of Key Combiner). *For a key combiner* KComb *and adversary* $\mathcal{A}$ *define experiment* $\boldsymbol{Exp}^{dPRF-\beta}_{\mathsf{KComb},\mathcal{A}}(n)$ *as in Figure 9. We say that* KComb *is (dual) pseudorandom if for any efficient adversary* $\mathcal{A}$ *we have that*

$$\boldsymbol{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{A}}(n) := \max_{\beta\in\{0,1\}} \left\{ \Pr\left[\boldsymbol{Exp}^{dPRF-\beta}_{\mathsf{KComb},\mathcal{A}}(n) = 1\right] - \frac{1}{2} \right\}$$

is negligible.

B Hybrid Schemes

KEM Combiner To achieve hybrid security, we make use of combiner schemes as proposed by Bindel et al. [13]. In the following, let $\mathsf{KEM}_1 = (\mathsf{KeyGen}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1)$ and $\mathsf{KEM}_2 = (\mathsf{KeyGen}_2, \mathsf{Encaps}_2, \mathsf{Decaps}_2)$ be two KEMs and let $\mathcal{C}[\mathsf{KEM}_1, \mathsf{KEM}_2] = (\mathsf{KeyGen}_{\mathcal{C}}, \mathsf{Encaps}_{\mathcal{C}}, \mathsf{Decaps}_{\mathcal{C}})$ be the hybrid KEM constructed by combiner mechanism $\mathcal{C}$ from KEM_1 and KEM_2. For all combiners, the key generation of the

```
Experiment $\mathbf{Exp}^{\text{dPRF}-\beta}_{\mathsf{KComb},\mathcal{A}}(n)$
1 : $\mathsf{IKM}_0, \mathsf{IKM}_1 \leftarrow \{0,1\}^n$
2 : $b \leftarrow \{0,1\}$
3 : $\mathcal{Q} \leftarrow \emptyset$
4 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{b,\beta}(\mathsf{IKM}_0,\mathsf{IKM}_1,\cdot)}(1^n)$
5 : return $[b = b^*]$
```

```
Oracle $\mathcal{O}_{b,\beta}(\mathsf{IKM}_0, \mathsf{IKM}_1, x)$
1 : if $x \in \mathcal{Q}$ then return $\perp$
2 : $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$
3 : if $\beta = 0$ then
4 :     $\mathsf{k}_0 \leftarrow \mathsf{KComb}(\mathsf{IKM}_0, x)$
5 : else
6 :     $\mathsf{k}_0 \leftarrow \mathsf{KComb}(x, \mathsf{IKM}_1)$
7 : $\mathsf{k}_1 \leftarrow \{0,1\}^{|\mathsf{k}_0|}$
8 : return $\mathsf{k}_b$
```

Fig. 9: Dual pseudorandomness experiment for Key Combiners

```
$\mathcal{C}[\Sigma_1, \Sigma_2].\mathsf{KeyGen}(1^n)$
1 : $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \Sigma_1.\mathsf{KeyGen}(1^m)$
2 : $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \Sigma_2.\mathsf{KeyGen}(1^m)$
3 : $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$
4 : $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$
5 : return $(\mathsf{pk}, \mathsf{sk})$
```

Fig. 10: Key generation function $\mathcal{C}[\Sigma_1, \Sigma_2].\mathsf{KeyGen}(1^n)$. The security parameter m needs to be derived from n depending on the requirements of the combiner.

combined scheme will simply be the concatenation of the two scheme's keys as shown in Figure 10.

A combiner is called **robust** if it combines two or more algorithms of the same kind such that the combined scheme provides security as long as one of the components provides security.

The XOR-Combiner A naive method to combine two KEMs would be to take the XOR of their keys $k = k_1 \oplus k_2$ as shown in Figure 11. As noticed by Giacon et al. [26] this results in a KEM that is IND-CPA, but not IND-CCA secure. Assuming that the challenger combines two IND-CCA secure KEMs by taking the XOR of their keys as described, then an adversary in the IND-CCA experiment can proceed as follows: Given a challenge (c_1^*, c_2^*), the adversary makes two decapsulation requests for (c_1^*, c_2) and (c_1, c_2^*) with randomly chosen ciphertexts $c_1 = c_2$. This information then allows the adversary to compute the decapsulation of the challenge ciphertext by taking the xor: $k = k_1^* \oplus k_2 \oplus k_1 \oplus k_2^* = k_1^* \oplus k_2^*$.

The XOR-then-MAC Combiner A better way to combine two KEMs is the XOR-then-MAC combiner as specified in Figure 12. This approach prevents the mix-and-match attack. The construction requires a strongly robust MAC combiner $\mathcal{C}[\mathsf{MAC}_1, \mathsf{MAC}_2]$ that provides one-time unforgeability even if one of the keys are

```
$\mathsf{Encaps}_{\mathrm{xor}}(\mathsf{pk}_1, \mathsf{pk}_2)$
1 : $(\mathsf{c}_1, \mathsf{k}_1) \leftarrow \mathsf{KEM}_1.\mathsf{Encaps}(\mathsf{pk}_1)$
2 : $(\mathsf{c}_2, \mathsf{k}_2) \leftarrow \mathsf{KEM}_2.\mathsf{Encaps}(\mathsf{pk}_2)$
3 : $\mathsf{k} \leftarrow \mathsf{k}_1 \oplus \mathsf{k}_2$
4 : $\mathsf{c} \leftarrow (\mathsf{c}_1, \mathsf{c}_2)$
5 : **return** $(\mathsf{c}, \mathsf{k})$
```

```
$\mathsf{Decaps}_{\mathrm{xor}}((\mathsf{sk}_1, \mathsf{sk}_2), (\mathsf{c}_1, \mathsf{c}_2))$
1 : $\mathsf{k}_1 \leftarrow \mathsf{KEM}_1.\mathsf{Decaps}(\mathsf{sk}_1, \mathsf{c}_1)$
2 : $\mathsf{k}_2 \leftarrow \mathsf{KEM}_2.\mathsf{Decaps}(\mathsf{sk}_2, \mathsf{c}_2)$
3 : $\mathsf{k} \leftarrow \mathsf{k}_1 \oplus \mathsf{k}_2$
4 : **return** $\mathsf{k}$
```

Fig. 11: KEM XOR[$\mathsf{KEM}_1, \mathsf{KEM}_2$] constructed by the naive XOR combiner (not IND-CCA secure).

chosen adversarially. Such a MAC combiner can be instantiated based on the Carter-Wegman paradigm [62] using universal hash functions. The XOR-then-MAC combiner is shown by Bindel et al. [13] to be **robust**. One drawback of this construction is that the resulting key-length is only half of that of the underlying KEMs.

```
$\mathsf{Encaps}_{\mathrm{XtM}}(\mathsf{pk}_1, \mathsf{pk}_2)$
1 : $(\mathsf{c}_1, \mathsf{k}_{\mathsf{KEM}_1} || \mathsf{k}_{\mathsf{MAC}_1}) \leftarrow \mathsf{KEM}_1.\mathsf{Encaps}(\mathsf{pk}_1)$
2 : $(\mathsf{c}_2, \mathsf{k}_{\mathsf{KEM}_2} || \mathsf{k}_{\mathsf{MAC}_2}) \leftarrow \mathsf{KEM}_2.\mathsf{Encaps}(\mathsf{pk}_2)$
3 : $\mathsf{k}_{\mathsf{KEM}} \leftarrow \mathsf{k}_{\mathsf{KEM}_1} \oplus \mathsf{k}_{\mathsf{KEM}_2}$
4 : $\mathsf{k}_{\mathsf{MAC}} \leftarrow \mathcal{C}[\mathsf{MAC}_1, \mathsf{MAC}_2](\mathsf{k}_{\mathsf{MAC}_1}, \mathsf{k}_{\mathsf{MAC}_2})$
5 : $\mathsf{c} \leftarrow (\mathsf{c}_1, \mathsf{c}_2)$
6 : $\tau \leftarrow \mathsf{MAC}(\mathsf{k}_{MAC}, c)$
7 : **return** $((\mathsf{c}, \tau), \mathsf{k}_{\mathsf{KEM}})$
```

```
$\mathsf{Decaps}_{\mathrm{XtM}}((\mathsf{sk}_1, \mathsf{sk}_2), (\mathsf{c}_1, \mathsf{c}_2), \tau)$
1 : $\mathsf{k}'_{\mathsf{KEM}_1} || \mathsf{k}'_{\mathsf{MAC}_1} \leftarrow \mathsf{KEM}_1.\mathsf{Decaps}(\mathsf{sk}_1, \mathsf{c}_1)$
2 : $\mathsf{k}'_{\mathsf{KEM}_2} || \mathsf{k}'_{\mathsf{MAC}_2} \leftarrow \mathsf{KEM}_2.\mathsf{Decaps}(\mathsf{sk}_2, \mathsf{c}_2)$
3 : $\mathsf{k}'_{\mathsf{KEM}} \leftarrow \mathsf{k}'_{\mathsf{KEM}_1} \oplus \mathsf{k}'_{\mathsf{KEM}_2}$
4 : $\mathsf{k}'_{\mathsf{MAC}} \leftarrow \mathcal{C}[\mathsf{MAC}_1, \mathsf{MAC}_2](\mathsf{k}'_{\mathsf{MAC}_1}, \mathsf{k}'_{\mathsf{MAC}_2})$
5 : **if** $\mathsf{Vf}(\mathsf{k}'_{\mathsf{MAC}}, (\mathsf{c}_1, \mathsf{c}_2), \tau) = 0$ : **return** $\perp$
6 : **return** $\mathsf{k}'_{\mathsf{KEM}}$
```

Fig. 12: KEM XtM[$\mathsf{KEM}_1, \mathsf{KEM}_2$] constructed by the XOR-then-MAC combiner.

Signature combiner As with KEMs there are also combiners that provide hybrid security for signature schemes. One might be tempted to avoid the use of signature combiners and instead deploy hash-based signature schemes, which are well-known to provide post-quantum security based on very weak assumptions. Such kinds of schemes have been around since the 1980s, which means that the usual concerns over the maturity of post-quantum cryptography do not apply here. However, even when using hash-based signatures it might be wise to combine them with a classical scheme as a fallback. This is because hash-based signatures require careful state management that is often difficult to assure.

Let $\Pi_1 = (\mathsf{KeyGen}_1, \mathsf{Sig}_1, \mathsf{Vf}_1)$ and $\Pi_2 = (\mathsf{KeyGen}_2, \mathsf{Sig}_2, \mathsf{Vf}_2)$ be two signature schemes. Then denote as $\mathcal{C}[\Pi_1, \Pi_2] = (\mathsf{KeyGen}_{\mathcal{C}}, \mathsf{Sig}_{\mathcal{C}}, \mathsf{Vf}_{\mathcal{C}})$ the hybrid signature scheme constructed from Π_1 and Π_2 using combiner mechanism $\mathcal{C}$. For all combiners, the key generation of the combined scheme will simply be the concatenation of the two scheme's keys as shown in Figure 10. A signature combiner is called **robust** if it combines two or more algorithms of the same kind such that the combined scheme provides security as long as one of its components provides security.

$\mathcal{C}_{||}$ *Combiner* This trivial combiner concatenates independent signatures from the two schemes side-by-side, as defined in Figure 13. Even though very simple, the construction is shown to be robust by Bindel et al. [14].

```
Sig_{C_||}(sk, m)
1 : $\sigma_1 \leftarrow \Pi_1.\mathsf{Sig}(\mathsf{sk}_1, m)$
2 : $\sigma_2 \leftarrow \Pi_2.\mathsf{Sig}(\mathsf{sk}_2, m)$
3 : return $\sigma \leftarrow (\sigma_1, \sigma_2)$

Vf_{C_||}(pk, σ)
1 : return $\Pi_1.\mathsf{Vf}(\mathsf{pk}_1, m, \sigma_1) \wedge \Pi_2.\mathsf{Vf}(\mathsf{pk}_2, m, \sigma_2)$
```

Fig. 13: Hybrid signature scheme $\mathcal{C}_{||}[\Pi_1, \Pi_2]$ constructed by concatenation.

$\mathcal{C}_{\text{str-nest}}$-*Combiner* One problem with the $\mathcal{C}_{||}$-Combiner is that due to downgrade attacks, separability of signatures is usually considered a liability in signature combiners. In downgrade attacks an adversary queries a signing oracle for a combined signature and later pretends to know only one of the schemes – this makes it possible for the adversary to *separate* a signature from a combined signature and pass it as a forgery. If downgrade attacks are to be expected – as it might be the case with an international protocol like EAC with multiple versions in concurrent use – it is recommended to use a $\mathcal{C}_{\text{str-nest}}$-Combiner. Here, the second signature scheme signs both the message and the signature from the first signature scheme, as defined in Figure 14. Bindel et al. [14] show that the $\mathcal{C}_{\text{str-nest}}$-Combiner is robust and inseparable.

```
Sig_{C_str-nest}(sk, m)
1 : $\sigma_1 \leftarrow \Pi_1.\mathsf{Sig}(\mathsf{sk}_1, m)$
2 : $\sigma_2 \leftarrow \Pi_2.\mathsf{Sig}(\mathsf{sk}_2, (m, \sigma_1))$
3 : return $\sigma \leftarrow (\sigma_1, \sigma_2)$

Vf_{C_str-nest}(pk, σ)
1 : return $\Pi_1.\mathsf{Vf}(\mathsf{pk}_1, m, \sigma_1)$
        $\wedge \Pi_2.\mathsf{Vf}(\mathsf{pk}_2, (m, \sigma_1), \sigma_2)$
```

Fig. 14: Hybrid signature scheme $\mathcal{C}_{\text{str-nest}}[\Pi_1, \Pi_2]$ constructed by nesting.

C Security Proof

C.1 Security Model

Setup We assume that there is a set $\mathcal{P}$ of parties. At the beginning, each party $P \in \mathcal{P}$ is assigned a long-term key pair $(\mathsf{sk}_P, \mathsf{pk}_P)$, certified via cert_P under a public key $\mathsf{pk}_{\mathsf{CVCA}}$ of a country verifying certificate authority. We assume that all parties know this public key of the certificate authority. For the attack we assume, to the advantage of the adversary, that all public keys pk_P and certificates cert_P are already known by the adversary. We also presume that each certificate carries the identifier from $\mathcal{P}$ or, to be precise, since we do not make any stipulation on the form of the identifiers in $\mathcal{P}$, we rather set the identifiers according to the distinguished name in the certificate.

It is sometimes convenient to further divide the set of parties $\mathcal{P}$ disjointly into chips and terminals, $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$. We say that a party P is a chip if $P \in \mathcal{C}$, and a terminal of $P \in \mathcal{T}$. Note that the two protocol versions, with signatures and using KEMs instead, differentiate between the type of the public keys for terminals: In the former case it is a certified signing key, in the latter case it is a certified KEM key. Chips use KEM keys in both versions. For the abstract security model, however, we do not distinguish the type of public key.

Sessions The adversary can now interact with the protocol in different sessions, representing a local protocol execution of a party. This is done via oracles to initiate new sessions (Init), to send protocol messages to a session (Send), to reveal a session key (Reveal), to test a session (Test), and to corrupt the long-term key of a party (Corrupt). Besides the protocol state the session can be described by the following entries:

- lbl is a unique administrative identifier lbl, chosen by the game during the execution.
- owner denotes the identity (from $\mathcal{P}$) of the owner of the session.
- $\mathsf{role} \in \{\text{chip}, \text{terminal}\}$ describes the role of the party owner.
- pid specifies the identity of the intended communication partner from $\mathcal{P}$.
- $\mathsf{st} \in \{\text{running}, \text{accepted}, \text{rejected}\}$ denotes the current state of the session.
- sid denotes the protocol's session identifier, initialized to $\perp$ and set only once upon acceptance.
- $\mathsf{revealed} \in \{\mathsf{true}, \mathsf{false}\}$ indicates if the adversary has revealed the session key.
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}$ indicates if the adversary has tested the session.
- key denotes the session key (or $\perp$).

It is convenient to use the unique identifier lbl to refer to individual entries, i.e., by writing $\mathsf{lbl.owner}$, $\mathsf{lbl.role}$, $\mathsf{lbl.pid}$ etc.

Attacks As mentioned earlier we prove security of the protocols in the real-or-random model of Bellare Rogaway [8]. This means that a secret challenge bit $b \leftarrow \{0, 1\}$ is chosen randomly at the outset, and the adversary either receives the actual session key of a tested session, or an independently sampled random string of the same length, the choice depending on the bit b. The task of the adversary is to predict b, ruling out trivial attacks like cases where the adversary knows the session key of a communication partner. To capture forward security we also assume a set $\mathcal{F} \subseteq \mathcal{P}$ describing the parties resp. roles which provide forward security. In our setting, if the chip picks an ephemeral KEM key, then

$\mathcal{F} = \mathcal{C} \cup \mathcal{T}$ covers all parties, else we have forward secrecy only for the terminals, $\mathcal{F} = \mathcal{T}$.

The attacker initially receives all certificates cert_P including the public keys pk_P of all parties $P \in \mathcal{P}$. We will use the initially empty set $\mathfrak{C}$ to store the identities of all corrupt parties. During the attack, the adversary may issue the following oracle queries (where we cover both forward and non-forward security in a single definition):

Initialization: The $\textsc{Init}(P, r)$ command first checks that party $P \in \mathcal{P} \backslash \mathfrak{C}$ is not corrupt, and that $P \in \mathcal{C}$ belongs to the chips for role $r = \text{chip}$ resp. $P \in \mathcal{T}$ for $r = \text{terminal}$. If all properties hold then it picks a new label lbl and initializes a new session for this label for owner $\mathsf{owner} \leftarrow P$ with role $\mathsf{role} \leftarrow r$. It sets $\mathsf{st} \leftarrow \text{running}$, $\mathsf{revealed} \leftarrow \mathsf{false}$, $\mathsf{tested} \leftarrow \mathsf{true}$, $\mathsf{sid} \leftarrow \perp$, and $\mathsf{key} \leftarrow \perp$.

Sending protocol messages: Upon calling $\textsc{Send}(\mathsf{lbl}, m)$ for an initialized session with identifier lbl, check that the session owner $\mathsf{lbl.owner} \notin \mathfrak{C}$ has not been corrupted yet. If so, then deliver the protocol message m to the party and hand the possible response back to the adversary. This may affect the status of the session and switch it to rejected or accepted (which we assume is known by the adversary). If it turns to accepted then the key key and the session identifier sid are set.

Upon acceptance, if there exists a partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid}$ as well as $\mathsf{lbl}'\mathsf{.revealed} = \mathsf{true}$, or if $\mathsf{lbl.pid} \in \mathfrak{C}$, then set $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$; else set $\mathsf{lbl.revealed} \leftarrow \mathsf{false}$. If there exists a partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid}$ and $\mathsf{lbl}'\mathsf{.tested} = \mathsf{true}$, then set $\mathsf{lbl.tested} \leftarrow \mathsf{true}$; else set $\mathsf{lbl.tested} \leftarrow \mathsf{false}$. This means that the session here inherits the $\mathsf{revealed}$ and tested flag from a partnered session resp. is considered to be revealed if the intended partner is already corrupt upon completion of the session here.

Reveal session key: Upon $\textsc{Reveal}(\mathsf{lbl})$ check that $\mathsf{lbl.st} = \text{accepted}$. If not, return $\perp$. Else, return $\mathsf{lbl.key}$ and set $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$ and also set $\mathsf{lbl}'\mathsf{.revealed} \leftarrow \mathsf{true}$ for any partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl}'\mathsf{.sid} = \mathsf{lbl.sid}$.

Test session: For $\textsc{Test}(\mathsf{lbl})$ check that $\mathsf{lbl.st} = \text{accepted}$ and that $\mathsf{lbl.tested} = \mathsf{false}$. If not, return $\perp$. Otherwise, if $b = 0$ then return $\mathsf{lbl.key}$, and if $b = 1$ then return a random value $\mathsf{k} \leftarrow \{0,1\}^{|\mathsf{lbl.key}|}$ of the same length. Set $\mathsf{lbl.tested} \leftarrow \mathsf{true}$ and also $\mathsf{lbl}'\mathsf{.tested} \leftarrow \mathsf{true}$ for any other partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl}'\mathsf{.sid} = \mathsf{lbl.sid}$. The latter spares us from keeping track of consistency, when the adversary tests both partners.

Corruption of long-term secrets: When calling $\textsc{Corrupt}(P)$ for $P \in \mathcal{P} \backslash \mathfrak{C}$, return sk_P to the adversary and add P to $\mathfrak{C} \leftarrow \mathfrak{C} \cup \{P\}$. Note that any session for this party cannot be stepped anymore via $\textsc{Send}$ queries.

In the non-forward-secure setting, when $P \notin \mathcal{F}$, mark all sessions lbl with $\mathsf{lbl.owner} = P$ or $\mathsf{lbl.pid} = P$ as revealed: $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$. This means that subsequent corruption of the long-term secret endangers the secrecy of the session key (on either side).

We assume that the adversary eventually outputs a guess b^* for b. We write $\mathbf{Exp}^{\mathrm{ke}}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n) = 1$ if $b^* = b$ in the above experiment for adversary $\mathcal{A}$.

C.2 Security Requirements

With the above attack model we can now state the security requirements. We note that we also require functional correctness in the sense that if two parties engage in an interaction and the adversary does not interfere with the executions, then the two parties obtain the same session identifier sid. This can be easily seen to hold for the protocols. Session matching now enforces that, even in presence of the adversary, two partnered sessions also derive the same key and correctly identify the intended partner and its role:

Definition 9 (Session Matching). *A key exchange protocol Π (for party sets $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$ and $\mathcal{F}$) provides session matching if for any efficient adversary $\mathcal{A}$ the following holds:*

Matching Keys: *For any distinct completed partnered sessions $\mathsf{lbl}, \mathsf{lbl}'$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} \neq \bot$ we have $\mathsf{lbl.key} = \mathsf{lbl}'.\mathsf{key}$.*

Exclusive Session Identifiers: *There do not exist three distinct sessions lbl, lbl', lbl'' with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl}''.\mathsf{sid} \neq \bot$.*

Matching Roles: *For any distinct partnered sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} \neq \bot$ we have $\{\mathsf{lbl.role}, \mathsf{lbl}'.\mathsf{role}\} = \{chip, terminal\}$.*

Authentication: *For any distinct partnered sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} \neq \bot$ we have $\mathsf{lbl.pid} = \mathsf{lbl}'.\mathsf{owner}$.*

We write $\boldsymbol{Exp}^{match}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n) = 1$ if the adversary in the attack described above violates any of the four properties.

Key secrecy now demands that the adversary cannot predict the secret challenge bit b better than by guessing, guaranteeing that session keys look random. This requires to exclude some trivial attacks, e.g., where a party interacts with an already corrupt partner or where the session key (of the party or of the partner) has also been revealed:

Definition 10 (Key Secrecy). *A key exchange protocol Π (for party sets $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$ and $\mathcal{F}$) provides key secrecy if for any efficient adversary $\mathcal{A}$ mounting the attack described above, the probability that*

Correct Prediction: *$b^* = b$, and*

Non-trivial attack: *There do not exist (not necessarily distinct) completed sessions $\mathsf{lbl}, \mathsf{lbl}'$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} \neq \bot$ with $\mathsf{lbl.tested} = \mathsf{lbl}'.\mathsf{revealed} = \mathsf{true}$.*

is negligibly close to $\frac{1}{2}$. More concretely we write $\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n)$ for the probability minus $\frac{1}{2}$ that the two properties hold.

We next prove the security of the two variants $\mathsf{SigPQEAC}$ (with signatures) and $\mathsf{KemPQEAC}$ (with KEM-based authentication). We start with the signature-based variant, and simultaneously consider the forward-secure and non-forward-secure version in the proof. We discuss afterwards that the proof also holds for the combined version. The proof of the KEM-based version then follows straightforwardly from the signature case, as we discuss afterwards.

C.3 Security Proofs for SigPQEAC

We first show the session matching property for SigPQEAC. We note once more that we give the proof of the forward-secure and non-forward-secure version within a single proof:

Proposition 1 (Session Matching of SigPQEAC). *The protocol SigPQEAC provides session matching. Specifically, for any adversary $\mathcal{A}$ initiating at most S sessions we have*

$$\begin{aligned}&\Pr\left[\boldsymbol{Exp}^{match}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n)=1\right]\\&\le 2S^2\cdot\Big(2^{-|r_2|}+2^{-|\mathsf{k}|}+\Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(n)=1\right])+\Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}^e}(n)=1\right]\Big).\end{aligned}$$

We note that the decryption error term $\Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}^e}(n)=1\right]$ is only required in the forward-secure version where the parties use an ephemeral KEM. In the non-forward-secure version this term disappears.

Proof. Recall that we have to show four properties, namely, that (1) identical session identifiers imply identical keys, (2) session identifiers are unique between two (partnered) session, (3) the roles match, and (4) for any distinct partnered sessions they point to the identity of the other party. Session identifiers are given as $\mathsf{sid}=(\mathsf{cert}_\mathcal{T},\mathsf{cert}_\mathcal{C},c,r_2,\ [\mathsf{pk}^e_\mathcal{C},c^e])$, and the partner identities are the distinguished names appearing in the certificates.

As for (1), the final session key is derived as $\mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}},\texttt{'KEY'}\|\mathsf{sid})$, where $\mathsf{k}_{\mathtt{KDF}}$ is the encapsulated key in ciphertext c under long-term key $\mathsf{pk}_\mathcal{C}$ or, if present, mixed with the encapsulated key k^e in ciphertext c^e under the ephemeral key $\mathsf{pk}^e_\mathcal{C}$. Since all values $c,\mathsf{pk}_\mathcal{C},c^e,\mathsf{pk}^e_\mathcal{C}$ and r_2 appear in the session identifier, a key mismatch for identical session identifiers can only happen in case of a decryption error for the encapsulations chosen by an honest terminal. The probability for this to happen for either of the two keys in any session is at most $S\cdot(\Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(n)=1\right]+\Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}^e}(n)=1\right])$. In the proposition's claim we subsume this under the factor S^2 for property (2).

For (2) note that the client contributes random string r_2 to the session identifiers, and the terminal sends a ciphertext c containing a random key k. Hence, the probability of having a session identifier for a chip-chip combination is at most $2^{-|r_2|}$, and for a terminal-terminal combination at most $2^{-|\mathsf{k}|}+\Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(n)=1\right]$, taking into account the possibility of creating the same ciphertext for different keys, resulting in a decryption error. Multiplying this by the at most S^2 number of session pairs, still yields a negligible bound. For (3) we assume that a chip or terminal only starts a communication in its indented role, such that the first certificate $\mathsf{cert}_\mathcal{T}$ specifies the terminal, and the second one $\mathsf{cert}_\mathcal{C}$ determines the chip. Hence, (4) also follows, because a match on session identifiers means that distinct partnered sessions use the same certificates, pointing to each other mutually. □

Theorem 1 (Key Secrecy of SigPQEAC). *Protocol SigPQEAC provides key secrecy. Specifically, for any adversary $\mathcal{A}$ initiating at most S sessions with U users there exist adversaries $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KDF}}, \mathcal{C}_{\mathsf{KComb}}$ such that*

$$\begin{aligned}
&\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n)\\
&\quad\le 3S^3 \cdot \left(2^{-|r_2|} + 2^{-|\mathsf{k}|} + \Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(n) = 1\right]\right)\\
&\qquad + U \cdot \left(\Pr\left[\boldsymbol{Exp}^{cert\text{-}unf}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}(n) = 1\right] + \Pr\left[\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{S},\mathcal{B}_{\mathsf{Sig}}}(n) = 1\right]\right)\\
&\qquad + S^2 \cdot \Big(4U \cdot \boldsymbol{Adv}^{IND\text{-}CCA}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(n) + 2 \cdot \boldsymbol{Adv}^{IND\text{-}CPA}_{\mathsf{KEM}^e,\mathcal{C}_{\mathsf{KEM}^e}}(n)\\
&\qquad\qquad + 4 \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{C}_{\mathsf{KDF}},\mathcal{D}_{KDF}}(n) + 4 \cdot \boldsymbol{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(n) + 2^{-|\mathsf{k}_{CNF}|}\Big)
\end{aligned}$$

Here, the sets for forward security are given as $\mathcal{F} = \mathcal{P} = \mathcal{C} \cup \mathcal{T}$ if the ephemeral KEM^e is used, and $\mathcal{F} = \mathcal{T}$ otherwise. The algorithms $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}$, $\mathcal{C}_{\mathsf{KComb}}$, $\mathcal{C}_{\mathsf{KDF}}$ run in approximately equal time as $\mathcal{A}$, and $\mathcal{D}_{KDF}$ is the uniform distribution on bit strings of length equal to $|\mathsf{k}_{KDF}|$.

Proof. Since the proofs for both versions are almost identical we give them simultaneously, explaining the slight adaptions when using the ephemeral data. We also note that the proof for the combined version where we slot together the chip's and terminal's messages is also identical, such that we do not mention this explicitly here. The proof itself follows by game hopping, starting with Game_0, the original attack of adversary $\mathcal{A}$. In each game hop we make slight changes to the games, gradually taking away attack possibilities. Formally, we will declare the adversary to lose and we stop the game immediately if one of these attack possibilities is triggered. We compensate for this by adding the (usually small) probability of such an attack to happen to the overall advantage of adversary $\mathcal{A}$. In the final game, $\mathcal{A}$'s advantage in predicting the secret bit b is exactly $\frac{1}{2}$.

We denote by $\Pr[\mathsf{Game}_i]$ the probability that $\mathcal{A}$ wins, i.e., correctly predicts the bit b in the i-th game and satisfies the freshness condition, minus the pure guessing probability $\frac{1}{2}$. In particular, $\Pr[\mathsf{Game}_0] = \Pr\left[\mathbf{Exp}^{\mathrm{ke}}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n) = 1\right] - \frac{1}{2}$. We also say that a party (resp. a key) is honest (at a certain point in time during the attack) if this party (resp. the party holding the key) has not been added to $\mathfrak{C}$ through a CORRUPT request at this point; else it is called malicious or corrupt.

Game Game_0. The original attack of $\mathcal{A}$.

Game Game_1. In this game we declare the adversary to lose if there are two (or more) sessions of honest chips resulting in the same value r_2.

Since the values r_2 are chosen uniformly, the probability of a collision in two of the at most S sessions, is given by $S^2 \cdot 2^{-|r_2|}$ by the birthday bound. Hence,

$$\Pr[\mathsf{Game}_0] \le \Pr[\mathsf{Game}_1] + S^2 \cdot 2^{-|r_2|}.$$

Game Game_2. In this game we now declare the adversary to lose if there are two (or more) sessions of the honest terminals sending the same ciphertext values c for the same chip identity $\mathsf{cert}_{\mathcal{C}}$.

Note that any such collision would mean that, either the encapsulated keys k must be equal, which happens with probability at most $S^2 \cdot 2^{-|\mathsf{k}|}$ across all sessions, or the keys are distinct but map to the same ciphertext c. The latter, however, would mean that decryption cannot be correct, such that we have a bound of $S^2 \cdot \Pr\left[\mathbf{Exp}_{\mathsf{KEM}}^{\mathrm{decErr}}(n) = 1\right]$ for this case. Therefore,

$$\Pr[\mathsf{Game}_1] \leq \Pr[\mathsf{Game}_2] + S^2 \cdot (2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}_{\mathsf{KEM}}^{\mathrm{decErr}}(n) = 1\right]).$$

Game Game_3. Compared to Game_2 declare the adversary to lose if an honest party accepts a certificate cert_P of some honest party (with identity P) but for a public key different than the generated one pk_P.

Note that this can happen on the chip side, during Terminal Authentication, as well as on the terminal's side during Chip Authentication. Either way it gives in a straightforward way a reduction $\mathcal{B}_{\mathsf{cert}}$ to the (un)forgeability of certificates: Initially guess (with probability $1/|\mathcal{P}|$) for which party P this will happen. Create all public keys of users, and certify pk_P for the certificate scheme for verification key $\mathsf{pk}_{\mathsf{CVCA}}$. Run $\mathcal{A}$'s attack. If $\mathcal{A}$ eventually sends a valid certificate cert_P^* for $\mathsf{pk}_P^* \neq \mathsf{pk}_P$, then output this certificate with pk_P^* as the certificate forgery.

Since we guess the right party with probability $1/U$ for $U = |\mathcal{P}|$, and the simulation of $\mathcal{A}$'s attack is perfect, we can bound the advantage in the previous game by the term for certificate unforgeability:

$$\Pr[\mathsf{Game}_2] \leq \Pr[\mathsf{Game}_3] + U \cdot \Pr\left[\mathbf{Exp}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}^{\mathrm{cert\text{-}unf}}(n) = 1\right].$$

At this point any (completed) session of an honest party contains a unique identifying value (r_2 on the chip's side and c on the terminal's side). Furthermore, looking at the opposite received value in that session (c for a chip resp. r_2 for a terminal), it follows that there can exist at most one honest partner session which has sent that value. We can thus, via a hybrid argument, assume that the adversary $\mathcal{A}$ only makes a single TEST oracle call, to a completed honest session, and that we can predict the number i of the session (according to initialization calls) upfront. This comes at a factor S in the security loss.

Game Game_4. We change Game_4 by letting the adversary lose if an honest chip obtains a valid signature $\sigma_{\mathcal{C}}$ of an honest terminal $\mathcal{T}$ for a message sid which has not been signed by the terminal during chip authentication at this point yet.

Note that, by the previous game hops, the signature can only be for the actual certified public key $\mathsf{pk}_{\mathcal{T}}$ of the terminal. Furthermore, since terminals use different prefixes `'TA'` resp. `'CA'` when signing in the two phases, the terminal cannot have signed these data in terminal authentication either.[7] It follows that

[7] This would in particular be also true if we combine the messages and let the terminal only sign once over all values. See Remark 3 after the proof.

such a valid signature constitutes a forgery for the signature scheme $\mathcal{S}$. The formal reduction $\mathcal{B}_{\mathsf{Sig}}$ receives a public key pk of the signature scheme, to guess the terminal's identity upfront with probability $1/|\mathcal{P}|$, and to run the attack in Game_3, with injecting the given signature key pk as $\mathsf{pk}_{\mathcal{T}}$ for the corresponding terminal. All other steps of the attacks are carried out locally, except for signature requests for our challenge terminal (where our reduction calls the signing oracle). If the reduction eventually encounters a valid signature for previously unsigned data, it outputs this as a valid forgery under $\mathsf{pk} = \mathsf{pk}_{\mathcal{T}}$.

Since the reduction provides a perfect view of $\mathcal{A}$'s attack in Game_3 up to the point of forgery for a correct guess of the party, we thus have:

$$\Pr[\mathsf{Game}_3] \leq \Pr[\mathsf{Game}_4] + U \cdot \Pr\Big[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{S},\mathcal{B}_{\mathsf{Sig}}}(n) = 1\Big].$$

Note that, since we have at most one honest matching partner according to games Game_1 and Game_2, then receiving a valid signature on the chip's side implies that it must come for the partnered terminal session—or the terminal is already corrupt at this point, in which case the chip's session cannot be successfully tested anymore.

For the next game hop we remark that with another factor S we can account for the prediction of the (at most) single honest partner session j, where we assume $i = j$ if there does not exist such a session. If any of the predictions turns out to be false, then we immediately output a random guess for the secret challenge bit b and stop. We can view this as a game modification, resulting in Game_5:

Game Game_5. Let the adversary only make a single TEST query and output the session number i and its potential partner session number j at the outset.

With the above discussion it follows that

$$\Pr[\mathsf{Game}_4] \leq S^2 \cdot \Pr[\mathsf{Game}_5].$$

We next branch according to a non-forward secure or forward-secure execution for the (only) test session.

Non-Forward Secure Case Assume that the i-th session, the one which eventually gets tested, is run in non-forward secure mode. In particular, the session does not involve the ephemeral keys.

Game Game_{6a}. Consider the ciphertext c for key k sent or received in the i-th session, encrypted under the long-term key $\mathsf{pk}_{\mathcal{C}}$ of a chip with certificate $\mathsf{cert}_{\mathcal{C}}$. Let $\bar{\mathsf{k}} \leftarrow \{0,1\}^{|\mathsf{k}|}$ be a random and independent key of the same length. In all subsequently processed honest sessions of the same chip with $\mathsf{cert}_{\mathcal{C}}$ in which this chip receives c, as well as in the i-th session and in the j-th session, use $\bar{\mathsf{k}}$ instead of k for the internal computations. Note that we do not need to take care of other honest terminal sessions using the same pair $(c, \mathsf{cert}_{\mathcal{C}})$, since there exists none according to Game_2.

Note that if the test session i is a terminal session, then the partner chip session j cannot be for a corrupt chip identity $\mathcal{C}$; else $\mathcal{A}$'s attack would be declared unsuccessful. This, together with the fact that only certified public keys are accepted by the terminal according to Game_3, implies that the public key $\mathsf{pk}_\mathcal{C}$ used to generate the ciphertext c must not be compromised. Hence, because the ciphertext c was created by the honest terminal itself, we have a pair $(\mathsf{pk}_\mathcal{C}, c)$ generated resp. held by honest parties only.

Analogously, if the test session i is a chip session, then expected partner terminal with certificate $\mathsf{cert}_\mathcal{T}$ must be honest at this point, or else the attack is deemed as trivial. But according to Game_5 it must then be the case that the incoming message with the ciphertext c for $\mathsf{pk}_\mathcal{C}$ has been signed (and thus created) exactly by the honest partner terminal with $\mathsf{cert}_\mathcal{T}$. Once more, it follows that the pair $(\mathsf{pk}_\mathcal{C}, c)$ is faithfully generated.

We first note that the probability that the chip with $\mathsf{cert}_\mathcal{C}$ "accidentally" receives the ciphertext c before the i-th resp. j-th session is bounded by $S \cdot (2^{-|\mathsf{k}|} + \Pr\Big[\mathbf{Exp}^{\text{decErr}}_{keyEM}(n) = 1\Big])$. Observe that Game_2 only covers the case of ciphertexts generated by honest terminals, but now c may have also been created by a malicious terminal. But it also holds here that either the freshly picked key in our ciphertext c equals the previously decapsulated key, or that a decryption error under the honest public key $\mathsf{pk}_\mathcal{C}$ occurs. Since there are at most S sessions before, the bound follows. From now on we exclude this case.

Now we can argue that replacing k by $\bar{\mathsf{k}}$ is valid, assuming security of the KEM. The reduction $\mathcal{B}_{\mathsf{KEM}}$ to the IND-CCA security of the long-term KEM now works as follows. The reduction guesses the chip's identity for $\mathsf{pk}_\mathcal{C}$ in question at the outset (with probability $1/|\mathcal{P}|$). Knowing $\mathcal{C}, i, j$, it receives $(\mathsf{pk}^*, \mathsf{k}^*, c^*)$ and injects pk^* as $\mathsf{pk}_\mathcal{C}$ in the game. This uses the fact that the public key is genuine in the i-th and j-th session. The reduction next executes the entire attack game, but uses k^* whenever it is supposed to use $\bar{\mathsf{k}}$. It also injects c^* in the corresponding terminal session j as c (which is again possible due to the previous considerations). For any other decapsulation request for a ciphertext different from c^* sent to chip $\mathcal{C}$ for $\mathsf{pk}_\mathcal{C} = \mathsf{pk}^*$, it calls the decapsulation oracle; all other decapsulation requests for other parties can be done locally. Here we use the fact that no previous session of the chip accidentally receives the same ciphertext c earlier, such that we can indeed process all possible incoming ciphertexts accordingly.

Since the adversary cannot corrupt the chip's long-term KEM key in the non-forward-secure setting, $\mathcal{F} = \mathcal{T}$, the simulation above is sound: We never have to reveal the decapsulation key of the chip later via a Corrupt request. We also observe that later corruption of the terminal's long-term signing key can be simulated; we only required confidentiality of the key in Game_5 up to the step where the honest terminal signs. In Game_{6a} the reduction may know the singing key and can reveal it in a subsequent Corrupt call. It follows that

$$\begin{aligned}\Pr[\mathsf{Game}_5] \leq \Pr[\mathsf{Game}_{6a}]& \\ &+S \cdot \big(2^{-|\mathsf{k}|} + \Pr\Big[\mathbf{Exp}^{\text{decErr}}_{\mathsf{KEM}}(n) = 1\Big]\big) + 2U \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(n),\end{aligned}$$

where the factor 2 in term $2U$ compensates for moving from a random challenge bit in the IND-CCA game to one with a fixed challenge bit.

Game Game_{7a}. Replace in Game_{6a} the result of applying KDF to $\bar{\mathsf{k}}$ for label `'KEY'` in sessions i and j by an independent and random value of the corresponding length.

To capture the loss for performing this game hop we give a reduction to the pseudorandomness of the KDF for the uniform distribution $\mathcal{D}_{\mathtt{KDF}}(1^n)$ on $\{0,1\}^{|\mathsf{k}_{\mathtt{KDF}}|}$. Our reduction $\mathcal{B}_{\mathsf{KDF}}$ essentially runs $\mathcal{A}$'s attack with help of the oracle $\mathcal{O}_{\mathsf{KDF}}$ (returning true KDF values) and the challenge oracle $\mathcal{O}_b$ (returning actual or random values). The only stipulation is that the two oracles are never called about the same input (ctxt, ℓ). For the simulation note that we have already replaced the actual key by a random value $\bar{\mathsf{k}}$ which now acts as the keying material IKM in the pseudorandomness game. However, this is done consistently for all chip sessions which receive the same ciphertext c as in the i-th and j-th session. Necessarily, according to the previous game, this can only happen for ciphertexts received after the i-th or j-th session.

In the simulation we thus call oracle $\mathcal{O}_{\mathsf{KDF}}$ for all sessions receiving the same value c about $\mathsf{ctxt} \leftarrow$ `'CNF'` $\|$ sid resp. $\mathsf{ctxt} \leftarrow$ `'KEY'` $\|$ sid and $\ell = n$. For the i-th and j-th session, however, for prefix `'KEY'` we call oracle $\mathcal{O}_b$ instead, for the value `'KEY'` $\|$ ctxt. This is admissible since any session of honest parties uses a unique value r_2 resp. c according to Game_1 and Game_2, such that the context information ctxt is two calls to $\mathcal{O}_{\mathsf{KDF}}$ and $\mathcal{O}_b$ are never equal in other sessions (and in the i-th and j-th session the call for label `CNF` is different). Our reduction eventually outputs whatever $\mathcal{A}$ outputs.

It is now easy to conclude that

$$\Pr[\mathsf{Game}_{6a}] \leq \Pr[\mathsf{Game}_{7a}] + 2 \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathsf{KDF},\mathcal{B}_{\mathsf{KDF}},\mathcal{D}_{\mathtt{KDF}}}(n),$$

for the uniform input distribution $\mathcal{D}_{\mathtt{KDF}}$ on bit strings of length $|\mathsf{k}_{\mathtt{KDF}}|$.

Remarkably, this already concludes the analysis in the non-forward secure case. We have now replaced $\mathsf{k}_{\mathtt{KEY}}$ by an independent and random value in the test session, such that $\mathcal{A}$ cannot predict b better than by guessing. This step, in contrast to the forward-secure case considered next, does not depend on the security of the confirmation value $\mathsf{k}_{\mathtt{CNF}}$. The reason is that the security of the chip's long-term KEM key, together with the fact that it cannot be legitimately corrupted later, already provides the necessary level of (implicit) authentication.

Forward-Secure Case The forward-secure case is slightly more involved but follows a similar line of reasoning. We branch off from Game_5 again. We denote by the terminal session of interest the session i if this is a terminal session resp. the session j if this is the partnered terminal session (and if it exists, i.e., if $i \neq j$). Analogously, we let the chip session of interest be the corresponding other session (if it exists).

A potential attack vector is now that the adversary may impersonate an honest chip $\mathcal{C}$ in a session with the honest terminal $\mathcal{T}$. Then the adversary could send its own (unauthenticated) ephemeral public key $\tilde{\mathsf{pk}}^e$ in the session with $\mathcal{T}$, allowing to decapsulate the ephemeral key $\tilde{\mathsf{k}}^e$ sent by the terminal. Only the other key part k, protected by the chip's long-term key $\mathsf{pk}_{\mathcal{C}}$, would not be immediately available. But in the forward-secure scenario the adversary could later corrupt the chip's long-term key, revealing also the part k and thus obtain both secrets. As we discuss below, however, the protocol prevents such attacks, since the adversary would have to send the confirmation value $\mathsf{k}_{\mathtt{CNF}}$ in the session with the terminal *before* being able to corrupt the chip's long-term secret and getting hold of both keys.

Game Game_{6b}*.* Change Game_5 by declaring the adversary to lose if the terminal session of interest (for certificate $\mathsf{cert}_{\mathcal{T}}$) exists and receives a valid final value $\mathsf{k}_{\mathtt{CNF}}$, and where the intended chip partner $\mathsf{cert}_{\mathcal{C}}$ is still honest at this point, but there is no partnered chip session of interest. In particular, the latter means that the test session is the terminal session and we must have $i = j$.

We argue through a sequence of several sub steps that this cannot happen too often. The reason is that, for this to happen, the adversary would still need to learn the key k protected under the chip's (then) honest long-term key $\mathsf{pk}_{\mathcal{C}}$. Otherwise the steps to combine and derive the keys still make $\mathsf{k}_{\mathtt{CNF}}$ random. The first step is thus similar to the IND-CCA case in the non-forward case. Since we only consider the case that the chip is honest at this point, and only need to measure if $\mathsf{k}_{\mathtt{CNF}}$ is valid before corruption takes place, we can in a first step replace k for any c sent to the intended chip partner by an independent random value $\bar{\mathsf{k}}$. It follows as in Game_{6a} that this increases the adversary's success probability (of generating such a value $\mathsf{k}_{\mathtt{CNF}}$) by at most

$$S \cdot \left(2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(n) = 1\right]\right) + 2U \cdot \mathbf{Adv}^{\mathrm{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(n)$$

for reduction $\mathcal{C}_{\mathsf{KEM}}$.

Next, replace $\mathsf{k}_{\mathtt{KDF}} \leftarrow \mathsf{KComb}(\bar{\mathsf{k}}, \mathsf{k}^e)$ in any honest session receiving or sending the ciphertext c for chip $\mathsf{cert}_{\mathcal{C}}$ by a random and independent value $\bar{\mathsf{k}}_{\mathtt{KDF}}$. Note that this is possible by the (dual) pseudorandomness of KComb. The reduction $\mathcal{C}_{\mathsf{KComb}}$ controls all steps of Game_{6b} and uses the real-or-random oracle to simulate all queries about $\mathsf{KComb}(\bar{\mathsf{k}}, \cdot)$ for sessions with the ciphertext c. Since we have already excluded for the CCA-case the possibility to receive c in some session before it appears in the i-th or j-th session, this simulation is sound and corresponds to have real values $\mathsf{k}_{\mathtt{KDF}}$ or random values $\bar{\mathsf{k}}_{\mathtt{KDF}}$. Let us stress once more that we let reduction determine its output depending on the adversary sending a valid confirmation value $\mathsf{k}_{\mathtt{CNF}}$ in the terminal session of interest. Altogether, we lose another term

$$2 \cdot \mathbf{Adv}^{\mathrm{dPRF}}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(n).$$

The next step is to replace the key $\mathsf{k}_{\mathtt{CNF}}$ in the i-th session (and thus j-th session because $i = j$) by an independent and random value $\bar{\mathsf{k}}_{\mathtt{CNF}}$. It follows from the pseudorandomness of KDF that this is a valid strategy. More formally, we build a reduction $\mathcal{C}_{\mathsf{KDF}}$ which calls the real-or-random oracle about $\texttt{'CNF'} \| \mathsf{sid}$ in the i-th session, and the real key derivation oracle for any other honest session (necessarily with a different session identifier). The reduction checks once more if the adversary succeeds in sending a valid confirmation value in the terminal session of interest. This adds another term of

$$2 \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathsf{KDF},\mathcal{C}_{\mathsf{KDF}},\mathcal{D}_{\mathsf{KDF}}}(n)$$

for the uniform distribution $\mathcal{D}_{\mathsf{KDF}}$ on strings equal to the length of derivation keys.

The final step is now to notice that the adversary needs to predict a random and unknown value $\bar{\mathsf{k}}_{\mathsf{CNF}}$. The reason is that we have already replaced the actual value $\mathsf{k}_{\mathsf{CNF}}$ by this random value in the i-th session. Since there is no chip session of interest, it follows that the adversary, when supposed to send the confirmation value, has absolutely no information about $\bar{\mathsf{k}}_{\mathsf{CNF}}$ and can thus only succeed with probability $2^{-|\mathsf{k}_{\mathsf{CNF}}|}$.

We emphasize that we are still not done yet. The previous game hop only covers cases where the chip is honest in the moment when it is supposed to send the confirmation value. But we can conclude that, if the test session is a chip session, then an honest party picks the ephemeral public key $\mathsf{pk}^e_{\mathcal{C}}$ freshly in that execution, and since the terminal cannot be corrupt for a successful attack when sending the signature, it must be the partnered honest terminal session creating the ciphertext c^e. Moreover, by the previous game hop, if the test session is a terminal session and created the ephemeral ciphertext c^e, then there must be a (unique) matching chip session, where the chip was honest when creating $\mathsf{pk}^e_{\mathcal{C}}$ in that session. In both cases we thus have an ephemeral public key $\mathsf{pk}^e_{\mathcal{C}}$ and a ciphertext c^e generated honestly by the partnered sessions.

Game Game_{7b}. Replace the key k^e in the i-th and j-th session by an independent random value $\bar{\mathsf{k}}^e$.

We can now run a reduction to the IND-CPA security of the ephemeral KEM^e. The reduction $\mathcal{C}_{\mathsf{KEM}^e}$ essentially simulates the attack according to Game_{6b}. We inject a given key pk^* as the ephemeral public key $\mathsf{pk}^e_{\mathcal{C}}$ of the chip $\mathcal{C}$ in the session of interest. If the terminal is supposed to create a ciphertext c^e in its session of interest then we inject the given ciphertext c^* as c^e. We use the corresponding given key value k^*, either random or encapsulated in c^*, as a replacement for k^e in both sessions. Note that this reduction and especially the injection of the given values are possible because, according to the previous game hop, the ephemeral public key and ciphertext are generated by honest parties. Any subsequent corruption of the chip's long-term key can be easily simulated by the reduction, since the ephemeral secret key to $\mathsf{pk}^e_{\mathcal{C}}$ is not revealed for this. We derive:

$$\Pr[\mathsf{Game}_{6b}] \leq \Pr[\mathsf{Game}_{7b}] + 2 \cdot \mathbf{Adv}^{\mathrm{IND\text{-}CPA}}_{\mathsf{KEM}^e,\mathcal{C}_{\mathsf{KEM}^e}}(n).$$

Game Game_{8b}. In the i-th and j-th session replace the computed value $\mathsf{k}_{\mathsf{KDF}}$ immediately by an independent and random value $\bar{\mathsf{k}}_{\mathsf{KDF}}$.

By the previous game hop we now use an independent random value $\bar{\mathsf{k}}^e$ in the i-th and j-th session. By the (dual) pseudorandomness of KComb, the output of $\mathsf{KComb}(\mathsf{k}, \bar{\mathsf{k}}^e)$ is thus indistinguishable from random. We can easily wrap this into a corresponding reduction $\mathcal{C}'_{\mathsf{KComb}}$, and obtain

$$\Pr[\mathsf{Game}_{7b}] \leq \Pr[\mathsf{Game}_{8b}] + 2 \cdot \mathbf{Adv}^{\mathrm{dPRF}}_{\mathsf{KComb}, \mathcal{C}'_{\mathsf{KComb}}}(n).$$

Game Game_{9b}*.* In the i-th and j-th session replace $\mathsf{k}_{\mathtt{KEY}}$ by an independent and random value $\bar{\mathsf{k}}_{\mathtt{KEY}}$ of the same length.

This can be done by the pseudorandomness of KDF. Formally, we get a reduction $\mathcal{C}'_{\mathsf{KDF}}$ which simulates all other steps of Game_{8b} locally, but uses the real-or-random oracle for KDF when computing $\mathsf{k}_{\mathtt{KEY}}$ in the sessions of interest. For the confirmation key $\mathsf{k}_{\mathtt{CNF}}$ it uses the actual KDF oracle in the pseudorandomness game. Since the computation of the confirmation key uses a different label `'CNF'` than for the session key, this is admissible according to the pseudorandomness game.

We finally have that the test session (and its partner session) both output an independent and random session key. It follows that the adversary cannot predict the secret challenge bit b in the key exchange protocol better than by guessing.

Conclusion To bound the original success probability $\Pr[\mathsf{Game}_0]$ we can consider the maximum over the non-forward secure and the forward secure variant:

$$\Pr[\mathsf{Game}_0] \leq \max\left\{\Pr[\mathsf{Game}_{7a}], \Pr[\mathsf{Game}_{9b}]\right\}.$$

We simply sum the other advantages in both cases to get the claimed bound. The stated bound in the theorem simplifies the bound further, e.g., by bunching together adversaries attacking the same primitive and prepending a corresponding factor. □

C.4 Variations and Remarks

We discuss some properties and options of the protocol and its security proof.

Remark 1 (Post-Quantum Security). We stress that our derived adversaries $\mathcal{B}_{\mathsf{cert}}$, $\mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KComb}}, \mathcal{C}_{\mathsf{KDF}}$ in the reductions of the game hops make straight-line (black-box) use of adversary $\mathcal{A}$. It follows that they are quantum if $\mathcal{A}$ is, but the reductions work nonetheless. In particular, if we assume that the underlying primitivies are *post-quantum secure*, then so is the key exchange protocol.

Remark 2 (Terminal Authentication). In the proof we do not consider the terminal's first signature $\sigma_{\mathcal{T}}$ in the Terminal Authentication. The reason is that for the secrecy of the derived key only the second signature σ_c is crucial to prevent the adversary from sending data in the name of the terminal. Still, terminal authentication through $\sigma_{\mathcal{T}}$ provides an extra layer of *entity* authentication via a challenge-response sub protocol, even if Chip Authentication is not executed.

Remark 3 (Combining signatures). In the round-reduced version we can simplify the protocol further. Instead of computing two signatures in the same round on the terminal's side, $\sigma_{\mathcal{T}} \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'TA'} \,\|\, \mathrm{ID}_{\mathcal{C}} \,\|\, r_1)$ and $\sigma_c \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'CA'} \,\|\, \mathsf{sid})$, one can compute a single signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'TC'} \,\|\, \mathrm{ID}_{\mathcal{C}} \,\|\, r_1 \,\|\, \mathsf{sid})$ instead. The proof remains valid for this case.

Remark 4 (Re-using random challenges). For the combined version, since the random values r_1, r_2 are sent within the same message flow now, one can omit r_1 and use a single random value r_2 for both signatures. Or, if combining with the above signature simplification, omit r_1 and create the (single) signature over the data with r_2 only.

Remark 5 (Key Confimation). We have used a simplification according to [25] for the confirmation values. Usually, one uses the derived key $\mathsf{k}_{\texttt{CNF}}$ to compute a message authentication code (on quasi unique parts of the session transcripts, such as the r_2-value). This is possible and the proof would indeed hold if the MAC is unforgeable. However, as pointed out in [25], one can also use some keying material (different from the session key) directly to accomplish key confirmation.

Remark 6 (Transcript Hashing). The signature and later the key derivation steps use the session identifiers $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$. To save on storage, one can compute a hash value of sid under a collision-resistant hash function. This will add a term for finding collisions under the hash function to the security bound in the theorem. If the hash function is an iterated hash function, then the party only needs to mix in the current data and merely needs to keep the intermediate hash value. This has been done for example in TLS 1.3 [55] for transcript hashes.

C.5 Security Proofs for KemPQEAC

We next discuss the security of the protocol version KemPQEAC where the terminal uses a long-term KEM key to decapsulate a key $\mathsf{k}_{\texttt{TA}}$ from which it derives an encryption key $\mathsf{k}_{\texttt{TENC}}$, a confirmation key $\mathsf{k}_{\texttt{TCNF}}$ for Terminal Authentication, and a MAC key $\mathsf{k}_{\texttt{TMAC}}$. The encryption key is used by the chip to encrypt its certificate information in Chip Authentication, providing privacy. The confirmation key is used as a replacement for the terminal's signature in Terminal Authentication, and the MAC key is used as a replacement for the signature in Chip Authentication.

We note that the proofs for KemPQEAC is very similar to the one for Sig-PQEAC, noting that the KEM-based transfer of $\mathsf{k}_{\texttt{TCNF}}$ implicitly ensures that only the honest terminal can recover the key and use it in the message authentication step. We thus only discuss the necessary adaptations. We also note that session matching holds as before.

Theorem 2 (Key Secrecy of KemPQEAC). *Protocol KemPQEAC provides key secrecy. Specifically, for any adversary $\mathcal{A}$ initiating at most S sessions with*

U users there exist adversaries $\mathcal{B}_{\mathsf{cert}}, \mathcal{D}_{\mathsf{MAC}}, \mathcal{D}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{D}_{\mathsf{KDF}}, \mathcal{C}_{\mathsf{KComb}}$ such that

$$\begin{aligned}
&\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(n) \\
&\quad\le 3S^3 \cdot \left(2^{-|r_1|} + 2^{-|r_2|} + 2^{-|\mathsf{k}|} + \Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(n) = 1\right]\right) \\
&\qquad + U \cdot \Pr\left[\boldsymbol{Exp}^{cert\text{-}unf}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}(n) = 1\right] + SU \cdot \Pr\left[\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{M},\mathcal{D}_{\mathsf{Sig}}}(n) = 1\right] \\
&\qquad + S^2 \cdot \Big(6U \cdot \boldsymbol{Adv}^{IND\text{-}CCA}_{\mathsf{KEM},\mathcal{D}_{\mathsf{KEM}}}(n) + 2 \cdot \boldsymbol{Adv}^{IND\text{-}CPA}_{\mathsf{KEM}^e,\mathcal{C}_{\mathsf{KEM}^e}}(n) \\
&\qquad\qquad + 6U \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{D}_{\mathsf{KDF}},\mathcal{D}_{KDF}}(n) + 4 \cdot \boldsymbol{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(n) + 2^{-|\mathsf{k}_{CNF}|}\Big)
\end{aligned}$$

Here, the sets for forward security are given as $\mathcal{F} = \mathcal{P} = \mathcal{C} \cup \mathcal{T}$ if the ephemeral KEM^e is used, and $\mathcal{F} = \mathcal{T}$ otherwise. The algorithms $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}$, $\mathcal{C}_{\mathsf{KComb}}$, $\mathcal{C}_{\mathsf{KDF}}$ run in approximately equal time as $\mathcal{A}$, and $\mathcal{D}_{KDF}$ is the uniform distribution on bit strings of length equal to $|\mathsf{k}_{KDF}| = |\mathsf{k}_{TA}|$.

Proof. The proof follows the one for the signature-based variant SigPQEAC almost identically. The only difference lies in Game_4 where the proof for SigPQEAC we relied on the unforgeability of the terminal's signature scheme. Subsequently, however, we merely used the fact that this signature could have only come from the terminal session of interest. We can prove the same property here for the KEM-based solution, but need several game hops to reach that conclusion. We start again in the proof for SigPQEAC after Game_4, and replace the hop to Game_5 by the analogue for MACs:

Game $\mathsf{Game}_{4'}$. Modify Game_3 by letting the adversary lose if an honest chip (having sent $c_{\mathtt{TA}}$ and r_1) obtains a valid MAC σ_c of an honest terminal $\mathcal{T}$ for a message sid which has not been authenticated by the terminal during chip authentication at this point yet.

The reduction considers several sub steps: First we need to take into account potential collisions for the r_1 value for honest chips. It follows by the birthday bound that honest chip sessions have colliding values with probability at most $S^2 \cdot 2^{-|r_1|}$. From now on we exclude such cases, meaning that there is at most one honest chip session which uses the same r_1-value as in the forgery.

Next we need to guess the right terminal (with probability $1/U$) as well as the chip session in which this happens for the first time (with probability $1/S$). Then we replace the the key $\mathsf{k}_{\mathtt{TA}}$ encapsulated by the honest chip in ciphertext $c_{\mathtt{TA}}$ by a random key $\bar{\mathsf{k}}_{\mathtt{TA}}$. This is done consistently for any other session of that terminal receiving $c_{\mathtt{TA}}$ (and any other honest chip session creating the same $c_{\mathtt{TA}}$ for $\mathcal{T}$). It follows by the IND-CCA security of the KEM that this is indistinguishable for the adversary. The formal reduction $\mathcal{D}_{\mathsf{KEM}}$ receives $(\mathsf{pk}^*, c^*, \mathsf{k}^*)$ as input and injects pk^* as $\mathsf{pk}_{\mathcal{T}}$ for the predicted terminal, as well as c^* as $c_{\mathtt{TA}}$ in the chip session. It uses k^* as the decapsulated key in all sessions for this terminal in which it receives $c_{\mathtt{TA}}$ (and, analogously in any honest chip session creating $c_{\mathtt{TA}}$ for $\mathcal{T}$). Note that this is possible since c^* is given to the reduction in advance. Any other ciphertext sent to $\mathcal{T}$ can be decapsulated via the decryption oracle (and honest chips know the encapsulated values anyway). We also remark that we are only interested in the point of time in which $\mathcal{T}$ is still honest, such that Corrupt queries cannot occur at this point.

Next, replace $\mathsf{k}_{\mathtt{TMAC}}$ in the predicted chip session and in any terminal session receiving $c_{\mathtt{TA}}$ and r_1 by a random value $\bar{\mathsf{k}}_{\mathtt{TMAC}}$. This is admissible since $\mathsf{k}_{\mathtt{TMAC}}$ is derived by the pseudorandom key derivation function KDF, applied to the now random key $\bar{\mathsf{k}}_{\mathtt{TA}}$. By the consideration above, it also holds that there is at most one chip session in which r_1 is used. The reduction $\mathcal{D}_{\mathsf{KDF}}$ is straightforward, and uses the fact that we can compute the other derived keys under $\bar{\mathsf{k}}_{\mathtt{TA}}$ via the KDF oracle, either because they use a different r_1-value or a different prefix `'TENC'` or `'TCNF'`.

Lastly, use the unforgeability of the MAC for the final step. That is, our reduction $\mathcal{D}_{\mathsf{MAC}}$ uses an external MAC oracle for computing the MAC σ_c in chip authentication for any terminal session receiving $c_{\mathtt{TA}}$ and r_1. We remark once more that no other chip (or terminal session) than the predicted chip session needs to call verification (or MAC) for that key. Hence, if the predicted chip session receives a valid σ_c for a previously unauthenticated message $\mathtt{'CA'} \,\|\, \mathsf{sid}$, then the reduction has found a forgery for the MAC scheme.

Altogether it follows:

$$\begin{aligned}\Pr[\mathsf{Game}_3] \leq \Pr\left[\mathsf{Game}_4'\right] &+ S^2 \cdot 2^{-r_1} \\ &+ SU \cdot \big(\Pr\Big[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{M},\mathcal{D}_{\mathsf{MAC}}}(n) = 1\Big] + 2 \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{D}_{\mathsf{KEM}}}(n) \\ &+ 2 \cdot \mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{D}_{\mathsf{KDF}},\mathcal{D}_{\mathtt{TA}}}(n)\big)\end{aligned}$$

for the uniform distribution $\mathcal{D}_{\mathtt{TA}}$ on $\{0,1\}^{|\mathsf{k}_{\mathtt{TA}}|}$. The rest of the proof is as in the one for Theorem 1. Subsuming the terms above under the theorem's statement yields the claimed bound. $\square$

Remark 7 (Privacy). Recall that we encrypt the chip's identity $\mathsf{cert}_{\mathcal{C}}$ in the first message of Chip Authentication under the key $\mathsf{k}_{\mathtt{TENC}}$ derived in Terminal Authentication for privacy reasons. In the argument above the security of the encryption schemes does not enter the security bound, such that it follows immediately that the proof also holds if encryption is not used. We note that the chip's identity also enters the value sid which is used at several places in the protocol, but usually in MACs or key derivation steps which are often regarded as pseudorandom function.