Wrocław University Of Science and Technology

DOCTORAL DISSERTATION

PACE and PACE CAM: Security Issues and Protocol Extensions

Author
Patryk Kozieł

Supervisor
prof. dr hab.
Mirosław Kutyłowski

Wrocław 2023

Abstract

The first part of the thesis is a security analysis of PACE and PACE CAM protocols (Password Authenticated Connection Establishment and Password Authenticated Connection Establishment with Chip Authentication Mapping). These protocols are included in a standard adopted by International Civil Aviation Organization (ICAO). The purpose of these protocols is to aid automation of border control with biometric passports. The security analysis presented here is a follow-up work on 2019 paper "Privacy and security analysis of PACE GM protocol" by Mirosław Kutyłowski and Przemysław Kubiak with draft proofs on security of PACE. In the thesis, full versions are provided as well as they are coupled with security analysis of PACE CAM.

The security analysis is prefaced with a literature review and legal and technical context. An important part of the introduction is a European Union regulation making deployment of PACE obligatory for official personal ID cards issued in the EU. The same regulation also allows for the introduction of extensions of PACE. Two such extensions are presented in the second part of the thesis. This work was published at two major conferences: IFIP Networking 2021 as "Poster: e-ID in Europe - Password Authentication Revisited" and at ESORICS 2021 as "PACE with Mutual Authentication – Towards an Upgraded eID in Europe". Both papers are coauthored by the author of this thesis. These extensions introduce two new functionalities to PACE: Proof of Presence - undeniable cryptographic proof that a successful protocol session has occurred with a given card and Mutual Authentication – strong authentication of both parties participating in the protocol. We discuss security features and design choices for these extensions.

Streszczenie rozprawy doktorskiej

Pierwszą częścią rozprawy doktorskiej jest analiza bezpieczeństwa protokołów PACE (Password Authenticated Connection Establishment) oraz PACE CAM (PACE with Chip Authentication Mapping). Protokoły te zawarte są w standardzie przyjętym przez International Civil Aviation Organization (ICAO). Ich celem jest automatyzacja procesu kontroli granicznej z użyciem paszportów biometrycznych. Analiza bezpieczeństwa przedstawiona w rozprawie to rozwinięcie pracy z 2019 roku "Privacy and Security Analysis of PACE GM Protocol" autorstwa prof. Mirosława Kutyłowskiego i dra Przemysława Kubiaka z zarysami dowodów bezpieczeństwa dla podstawowych własności protokołu PACE. W niniejszej rozprawie, przedstawiamy pełne wersje dowodów wraz z argumentacją dla protokołu PACE CAM.

Część analizy bezpieczeństwa jest poprzedzona wstępem zarysowującym kontekst prawny, technologiczny oraz zawierającym przegląd literaturowy. Istotną częścią wprowadzenia są rozporządzenia Unii Europejskiej, które obligują państwa członkowskie do implementowania protokołu PACE w warstwie elektronicznej oficjalnych dokumentów tożsamości oraz pozwalają na rozszerzenia funkcjonalności tych protokołów. Dwa takie rozszerzenia są przedstawione w drugiej części rozprawy. Są to wyniki opublikowane na dwóch czołowych konferencjach, a autor rozprawy jest jednym ze współautorów. Pierwsza praca ukazała się na konferencji IFIP Networking 2021: "Poster: eID in Europe - Password Authentication Revisited", zaś druga "PACE with Mutual Authentication – Towards an Upgraded eID in Europe" na konferencji ESORICS 2021. Rozszerzenia to, odpowiednio, PACE PoP (Proof of Presence) - rozszerzający podstawowy protokół o funkcjonalność niezaprzeczalnego dowodu że udana sesja protokołu miała miejsce z udziałem okreslonej karty oraz PACE MA (Mutual Authentication) - rozszerzająca protokół o możliwość silnego kryptograficznego uwierzytelnienia obu stron uczestniczących w protokole. W rozprawie omawiamy własności bezpieczeństwa i decyzje projektowe dla tych protokołów.

Contents

Chapter 1

Introduction

1.1 eIDs in Europe

Electronic identity cards – eIDs – equipped with a chip are becoming the standard identity documents issued by European governments in recent years. Among other functionalities, they support electronic authentication. It is needed, because apart of personal data printed on the document, they store sensitive personal data in the memory, including biometric data authenticated cryptographically. Moreover, an eID document serves as a cryptographic token held by its owner. Electronic authentication could be used in many scenarios, with many automated services that require the authentication of the person holding the eID. Services can range from airport e-Booths, e-gov web services, health services (including access to patient's medical information), financial services, or even vending machines with age or identity verification.

For the sake of the durability of the document and a possible wide range of usage, wireless interface communication between a reader (entry point to the service) and the eID is a preferred choice over using other interfaces. This creates possibilities for attacks for adversaries who could try to exploit this channel, from merely intercepting to even hijacking the session between the eID and the reader. That could be done by installing malicious devices around the reader or even trying to make a connection with the chip of an eID being in its vicinity. As the chip does not have its own power supply and is activated by the reader, in practice there is no option to control when the connection is made. Therefore, there is a need for strong cryptographic protection, taking into account different attack scenarios.

As an eID is compulsory for all adult citizens, it provides an opportunity to be used as a strong cryptographic token ready to use in all the scenarios mentioned. It has the advantage of being widely recognized and universally understood, with users knowing how to use it, as opposed to maybe trying to introduce a new solution for such scenarios. Countries such as Germany and Estonia have successfully introduced eIDs for their citizens before the EU

regulation and integrated them with e-goverment services. However, there are issues that need to be examined from the point of view of the European common market. eIDs issued by different countries can be very valuable cryptographic tokens; however, they are usually not implemented with the interoperability and cooperation between countries in mind. Therefore, until recently, they could not be used as universal ID tokens. This has created barriers to effective cross-border e-government services and free movement of the citizens.

1.2 Legal context and introduction of PACE

To ensure interoperability on eIDs in member states of the European Union, a Regulation 2019/1157 [1] was introduced with the aim of implementing the common mechanisms deployed on official eIDs in countries belonging to the EU. This regulation requires that on all eIDs issued after August 2, 2021, PACE protocol will be deployed. The Regulation refers to the standard [2] adopted by the International Civil Aviation Organization (ICAO) and intended for Machine Readable Travel Documents (MRTDs). It is an example of a PAKE protocol - Password Authenticated Key Exchange, where the authentication is based on the knowledge of a password. PACE was developed by the German Federal Office for Information Security (BSI) [3]. It was intended for travel documents such as e-passports, with the aim of facilitating automatic border control while securing against potential privacy threats. This protocol is widely used in biometric passports - e-passports. It automates the process of border control at special e-Booths, where the front page of the passport with MRZ (machine readable zone) with individual data (e.g., so-called CAN number) is scanned. Thanks to PACE, there is a secure connection established between the chip in the passport and the device at the airport and secured with the password derived from the data scanned. This is a password-authenticated key exchange which should fail if a wrong password is provided. The established connection is used to send data for further processing using symmetric encryption and MAC keys, effectively creating a secure channel from the point of view of privacy and data protection.

The aforementioned regulations, in addition to enforcing PACE on eIDs, also provide an opportunity to introduce extensions of PACE on eIDs next to the original protocol, as long as they do not interfere with PACE. This prompted the development of an approach in which it is proposed to create a multitude of protocols based on PACE, fulfilling different purposes and providing many functionalities. Such an approach makes sense from the point of view of efficiency, as chips on eIDs are limited in computational power and memory. Deploying extensions, with a high level of reuse of the existing mechanisms, is more pragmatic than implementing new protocols from scratch. Using what is already there can promote backward compatibility and greatly simplify security analysis. Good practices of this kind are presented in Section 4.1. In this thesis, two extensions following presented principles are proposed: PACE MA (Mutual Authentication) and PACE PoP (Proof of Presence). Those extensions come

with interesting and useful functionalities that can be used in many scenarios.

The proposed approach may ease the solution to the problem of reaching international consensus when it comes to agreeing on universal and internationally recognized tokens. This can be done by favoring options for particular schemes based on already deployed PACE with the benefit of greater interoperability, rather than choosing one of existing, probably incompatible solutions already deployed on national eIDs in Europe that would surely advantage particular manufacturers (which can create unfairness).

Although intended for different purposes, protocols from the "PACE family" may be used outside of the eID realm. They are well-researched protocols, used for many years now and there is a lot of development experience in deploying them on biometric passports and eIDs. There is also a sense of trust among the general public that comes from that. As there are many more scenarios of usage of PACE and its extensions, throughout the thesis the terms eID and the reader are replaced by ***chip*** and ***terminal***, as these terms are more general and fitting, especially for remote communication.

1.3 Original security analysis of PACE and contribution described in this thesis

The first security analysis of PACE was published in [4] showing that the PACE protocol is secure in the real-or-random sense of Abdalla, Fouque and Pointcheval [5], under a number-theoretic assumption related to the Diffie-Hellman problem and assuming random oracles and ideal ciphers. Another article on the subject appeared by Kutyłowski and Kubiak [6] in the year 2019 with concerns about the original security analysis regarding the validity of the proof and the limited scope of the proven results when it comes to attack scenarios. The main contribution was a different approach to the analysis with the aim of providing a thorough analysis, regarding both the confidentiality and the privacy features. One of the main parts of the contribution of this thesis is a full and refined version of the proofs from that paper with some ideas re-thought and gaps fixed. A strong result on reducing all active adversaries to a passive one is one of the most important contributions. Except that, crucial results about key confidentiality and password security are presented. Additionally, within that topic, particular design choices in PACE are discussed together with their impact on the practical security of the scheme. At the time of writing, this work is under preparation for a publication.

1.4 Introduction of PACE CAM

Although PACE was originally intended to be used for a verification at a local terminal, it can also be used for online authentication on a remote terminal (cf. [3]). In this case, the local terminal is merely an untrustworthy man-in-the-middle, and an end-to-end connection is created between the chip and the

remote terminal. PACE ensures that no connection will be established unless the chip holder gives explicit consent by providing the password. However, the chip is authenticated only by the knowledge of the password. As the chip holder explicitly enters the password in the terminal, chip authentication is very weak. For that reason, efforts were made to couple PACE with strong chip authentication. Bender et al. [7] proposed a PACE|AA protocol, where the chip proves that it has a private key assigned to it. The idea is to create a digital signature of the chip using the messages already exchanged in the protocol. Subsequently, a simplified version of the above protocol was presented independently in [8] and [9] with some overlap. It was patented by the German government and adopted by ICAO under the name PACE CAM (PACE with Chip Authentication Mapping) [2]. This thesis couples the security analysis of plain PACE with the security analysis of PACE CAM as to the author's knowledge there is no such comprehensive analysis at the time of writing, and this is a new contribution. Since PACE CAM differs only by a single message added at the end of the communication (disregarding technicalities concerning PKI), many security features of PACE transform smoothly and can be reused for the analysis of PACE CAM.

1.5 PACE modifications and extensions in literature

PACE and its extensions are of interest to many researchers in the field. There is a lot of work published on this subject, in addition to those described above. To better understand the landscape of research, some of them will be cited here. When it comes to the topic of authentication, there have been efforts to link PACE with biometric authentication, where the password is derived from biometrics of the eID holder [10]. Also, to close a technical gap, PACE CAM has been extended to the option of Integrated Mapping for establishing the common generator $\hat{g}$ in [11] and also a couple of years earlier the security analysis of the IM mode was published [12]. There are also different motivations in mind when modifying PACE, for example, motivated by the concern of the security and cost of implementing PRNGs on low-power devices, a paper by Mirosław Kutyłowski and Adam Bobowski was presented in which they propose to substitute PRNG with a secure deterministic solution ([13]). An interesting modification of the PACE protocol has been presented in [14] – it uses a one-time pad as encryption method during the first phase of PACE. It seems to be quite controversial, but it turns out, for instance, that it doubles the expected number of trials in the brute-force attack on the password.

1.5.1 Proposed extensions PACE MA and PACE PoP

As already mentioned, apart from the security analysis of PACE and PACE CAM there is also a contribution of proposing two extensions to PACE: PACE with Mutual Authentication [15] presented at ESORICS conference in 2021 and

PACE with Proof of Presence [16] presented at IFIP Netowrking conference also in 2021. These extensions were designed with a strong re-usability mindset in terms of executable code and the security analysis of the original protocol. Some of the security analysis from Chapter 3 will be reused to provide arguments about the security features of the extensions. PACE MA is an extended version of PACE with mutual authentication, both of the chip and the terminal. PACE PoP is short for Proof of Presence, where in this modification at the end of the protocol a strong cryptographic signature of the session is created. The security discussion and design comments are presented in Chapter 4.

1.6 Requirement-driven security analysis

Security analysis of PACE and PACE CAM referenced in this thesis is carried out with security requirements in mind. Instead of using theoretical models, pragmatic requirements are presented, and the security proofs are centered around them. Such an approach benefits from being less error-prone than wide security/privacy models with multiple properties for the analyzed protocols. This probably provides more trust to the analysis, as often the soundness of a complicated model is not clear. Using requirement-driven approach for particular security issues allows also to focus more on practical security (Section 2.2.1).

A stepping stone for the analysis is to greatly reduce the capabilities of the active adversary to a passive one, meaning an adversary not able to interfere with the messages exchanged by the protocol parties such that the session is not aborted. This result is shown early in the analysis and then drives many results.

When it comes to password-authenticated protocols, it is a known fact that such authentication is weaker than authentication with strong cryptography based on the hardness of particular security assumptions because of the low-entropy of the password options. The best thing that can be done is to make sure that the adversary has to carry out an attack with all possible passwords one by one (brute force attack). That stands in opposition to the adversary having some kind of advantage, e.g. being able to filter out many passwords at once. That is an example of a pragmatic requirement. Some other concerns around password security could be that the adversary is able to recognize that two sessions were executed using the same (correct) password. This is valuable information because it can be used to track an eID holder. Such an approach, which aims to define practical problems, leads to practical corollaries that provide a deep understanding of the security features of the discussed protocols. The capabilities of the adversary are also modeled with a practical mindset in mind. That is because, depending on the context, the adversary can have different goals, such as getting to know the password, impersonating a party in the protocol, or hijacking the session to steal biometric data.

Chapter 2

Preliminaries

Definitions, assumptions, and cryptographic building blocks that are used throughout the thesis are presented in this chapter. Some assumptions, however, will be presented during the security analysis to put them in the context and make them easier to follow.

2.1 Cyclic groups

The calculations in the discussed protocols take place in some chosen cyclic algebraic subgroup of order q of a multiplicative group modulo p with a generator g, i.e. an element of the group such that $g, g^2, g^3, \ldots, g^q = 1$ are all the elements of the group. It is assumed that $q|p-1$ and both p and q are large prime numbers, so the assumptions in the following sections hold. Recall that in cyclic groups of prime order, every element except "1" (the identity element) is a generator. Sometimes the subgroup generated by g will be denoted by $\langle g \rangle$. Throughout this thesis, it is implicitly assumed that when operating on elements of this group, the operations are performed within that group and therefore mod p will be skipped for brevity. Additionally, this makes it more universal, as the specification of PACE [2] allows the usage of a group based on elliptic curves in the protocol and in those cases such a notation would not make sense. The exponents for the elements of $\mathcal{G}$ come from a finite field $\mathbb{Z}_q$ (sometimes denoted by $\mathbf{Z}/q\mathbf{Z}$), with modular addition, subtraction, multiplication and division, with elements in a set $\{0, 1, 2, \ldots, q-1\}$ and the operations on them will use mod q throughout the thesis.

2.2 Adversary

In order to conduct the security analysis, the notion of an adversary $\mathcal{A}$ is introduced. This notion represents an entity whose objective is to somehow break the security of a protocol or some cryptographic primitive, depending on the context. As the protocol itself is clearly an algorithm run by some parties, it

makes (formal) sense to introduce $\mathcal{A}$ as an algorithm as well, which is a standard practice in cryptography. To reflect real world adversaries, commonly called *attackers*, this algorithm must be computationally bounded[1] to exclude brute-force attacks. The brute-force attack way of working is to search through all possible algorithm runs, usually of the order 2^n, where n is the bit length of the security parameter (usually the key length) in a given security experiment. Such computational effort is in practice impossible, and cryptographic schemes are constructed in a way that it is the only way to break their security. Hence, the following definition:

Definition 2.1 (Adversary)**.** *An adversary $\mathcal{A}$ is a polynomial time in the security parameter randomized algorithm.*

Randomization can mean different things depending on the context. In this thesis, it is an access to some source of randomness that can be used in different strategies of the adversary. The adversary in the security experiment/proof needs some way of operating (a *model*), and his capabilities need to be clearly defined, reflecting real-world scenarios close enough to put trust in the security proof. Usually the capabilities are very wide, to the point that seems unobtainable in practice, which only puts more trust in the security of a particular scheme when it is shown that the scheme is secure against the adversary. Here, the examples of the adversaries are skipped, to be described in the actual security analysis, depending on the security feature in question.

2.2.1 Negligible advantage and practically negligible advantage

Informally speaking, the probability of $\mathcal{A}$ to achieve his goals is called his *advantage*. In the classical sense, the advantage of $\mathcal{A}$ is negligible when it is a function of the security parameter n of order $\mathcal{O}(2^{-n})$. This represents the notion that the adversary has no better option than a brute-force attack that is outside of his computational power or, in other words, that there is no polynomial-time randomized algorithm that could conduct a successful attack. In this thesis, the most common way to discuss the advantage of the adversary is closer to what is called *concrete security*, which is a more practical approach, focusing on estimating the actual computational complexity of the problem. Also, as the subjects of the work presented are interactive protocols, when the key is used only once per session and time for an action is limited, a practical advantage can be modeled differently than, for example, trying to model an attack on an encrypting oracle. This means that probabilities that in classical cryptography would be deemed too high, here are practically more than sufficient, as usually $\mathcal{A}$ has only one chance to attack having a particular input. These issues will be discussed in the security analysis to follow.

[1] Sometimes in the security analysis in the following Chapters, the adversary is given a "superpower", e.g. he is able to generate a very large pool of keys, say $\sim 2^{90}$ keys. In those cases it is needed to show that even a very powerful adversary is not able to break the security of a particular scheme.

2.3 Group assumptions

In this section, problems considered computationally infeasible in the group $\mathcal{G}$ are presented and formulated as assumptions used throughout the thesis.

Recall that a *discrete logarithm to the base b* of an element E from the group $\mathcal{G}$ is $x \in \mathbb{Z}_q$ such that $b^x = E$. The term *discrete logarithm* will be used without the base specified when the base is obvious from context (usually the generator). Discrete logarithm problem is a standard problem in cryptography, meaning that it is believed that for particular groups finding discrete logarithm of its elements is computationally infeasible, meaning in practice computing a solution of a random instance takes so much time that it is considered impossible to compute. It is formalized by the following assumption:

Assumption 2.1 (Discrete Logarithm Problem Assumption)**.** *In the considered group $\mathcal{G}$, given g and g^x where x is a random variable with uniform distribution, the adversary $\mathcal{A}$ has a negligible advantage in computing x.*

Other known related problem from that field is Computational Diffie-Hellman Problem, which can be formulated as an assumption:

Assumption 2.2 (Computational Diffie-Hellman Problem)**.** *In the considered group $\mathcal{G}$, given g, g^x, g^y, where x and y are random variables with uniform distribution, the adversary $\mathcal{A}$ has a negligible advantage in computing g^{xy}.*

There is also a decisional version of that assumption:

Assumption 2.3 (Decisional Diffie-Hellman Assumption)**.** *In the considered group $\mathcal{G}$, given a Diffie-Hellman tuple (g, g^x, g^y, R), where x, y are uniformly distributed and $R = g^{xy}$ with probability $\frac{1}{2}$ and a random group element otherwise, adversary $\mathcal{A}$ has a negligibly greater advantage than $\frac{1}{2}$ in deciding whether $R = g^{xy}$.*

There are others related computational assumptions used in this thesis. One of them is:

Assumption 2.4 (Inverse Computational Diffie-Hellman Assumption)**.** *In the considered group $\mathcal{G}$, given g and g^x, where x is a random variable with uniform distribution, the adversary $\mathcal{A}$ has a negligible probability of computing $g^{\frac{1}{x}}$.*

Assumption 2.2 and Assumption 2.4 are equivalent to each other [17].

2.3.1 KEA 1

Besides standard cryptographic assumptions, the following one is used, as it turns out is the assumption that is fundamental for the security of PACE [18] (the formulation from [19]):

Assumption 2.5 (KEA1 - Knowledge of Exponent Assumption 1)**.** *For any adversary $\mathbf{A}$ that takes input q, g, g^a, where q is a prime such that $2q+1$ is also*

a prime and g is a generator of order q of a subgroup of a group of order $2q+1$ and returns (C, Y) with $Y = C^a$, there is an extractor $\overline{\mathbf{A}}$ that given the same inputs as $\mathbf{A}$ returns c such that $g^c = C$.

This assumption captures an interesting point of the discrete logarithm problem: (roughly) if $\mathcal{A}$ is able to create a valid Diffie-Hellman tuple only by raising to the same known power the first two elements from the tuple. This is an assumption that will prove useful many times in this thesis.

Note 2.5.1. *The number "1" next to the assumption is added to avoid confusion with other KEA assumptions which have been proven to be false in the past [19].*

Note 2.5.2. *Implementation of the protocol in real-world applications should be performed in a way that carefully takes into account security assumptions. Sometimes implementation choices are driven by performance or easiness of incorporating some algebraic structure in the project. While those premises are very important, the crucial deciding factor should be if the implemented protocol will be secure and adhering to the assumptions ensures that (or at least prevents many security threats).*

For the security analysis of password security, another KEA assumption will be introduced in Section 3.4.2.

2.4 Cryptographic primitives

2.4.1 Hash functions

Hash function H is a one-way function that takes a bit string input of an arbitrary length and returns a bit string of fixed length. *"One-wayness"* of the hash function means that it is computationally infeasible to find input for given output. Good hash function should be collision resistant, meaning that it is computationally infeasible to find x and y such that $H(x) = H(y)$. In this thesis, every hash function is modeled as a *random oracle*, which means that they are thought of as black-box that on a new input returns a truly uniform random output and for an input already seen returns what was returned in the past. Other interpretation is also sometimes useful – that a random oracle is an instance of a truly random function from family of random functions mapping from a particular space to another where every element is mapped uniformly. Hash functions in the thesis can have different domains and image spaces, depending on the context, not only mapping bit string to bit strings, but for example mapping elements of $\mathcal{G}$ to $\mathbb{Z}_q$.

For the security analysis, also a following assumption will be needed at some point:

Assumption 2.6. *In the following left-or-right game there are given keys K_1, K_2 and parameters $p_1 \neq p_2$. Distinguisher needs to decide whether there is a K that K_1 and K_2 were produced as $K_1 = H(K||p_1)$, $K_2 = H(K||p_2)$ or if they are random strings.*

The assumption says that the advantage of the distinguisher is negligible over a random guess if the length of K is sufficiently large.

Random Oracle Model is a very strong assumption about hash functions, sometimes a little bit weaker assumptions suffice, hence the introduction of Assumption 2.6 which is used when when proving the main result about key confidentiality.

Throughout the thesis hash functions will be simply denoted by H, as all of them are modeled as random oracles and differentiating them would not provide a useful contribution, but rather could cause confusion with the notation. The image of the particular functions is always taken into account in the discussion.

2.4.2 Encryption

A symmetric encryption algorithm is a function that takes a key K from key space $\mathcal{K}$ and a plaintext P from a message space $\mathcal{P}$ and returns a ciphertext C from the space of ciphertexts $\mathcal{C}$:

$$\mathbf{Enc} : (\mathcal{K}, \mathcal{P}) \longrightarrow \mathcal{C}$$

It is coupled with a decrypton function **Dec** that on the same key K and ciphertext C returns P. In this thesis encryption is modeled as a family of random keyed permutations, meaning that for a key K it is a random bijection between the space of plaintext and ciphertexts, what of course means that they need to be the same.

Encryption is usually denoted in the thesis as $\mathbf{Enc}(key, message)$, there are however a couple of places when it is denoted as $\mathbf{Enc}_{key}(message)$ when there is a focus placed on the message itself in the discussion.

2.4.3 Schnorr Signature

For the PACE PoP modification (Section 4.3) we reuse Schnorr Signature, a widely used and recognized scheme, which appeared first in [20].

The setup is the following: a user who wants to sign a message M needs to have a public key $pk = g^{-sk}$, where $sk \in \mathbb{Z}_q$ is his private key. g is a generator of a subgroup of prime order q of a group $\mathcal{G}$. There is also a hash function $H : (\mathcal{G}, \{0,1\}^*) \longrightarrow \mathbb{Z}_q \backslash \{0\}$ in the system. Here, the details of the registration of the public key in a Certificate Authority are skipped, as there are outside of the security analysis. The following procedures are of interest: how to generate a signature on a message M and how to verify it.

Signature generation with the secret key sk:

1. Pick a random number $r \in \mathbb{Z}_q \backslash \{0\}$ and compute $X := g^r$.
2. Compute $e := H(X, M)$
3. Compute $y := r + sk \cdot e \ (mod\ q)$ and output (e, y) as the signature.

Signature (e, y) verification with public key pk: Verifier computes:

$$\bar{X} = g^y \cdot pk^e$$

and checks if $H(\bar{X}, M) = e$. If not, the signature is rejected. The security of the scheme is based on the hardness of the Discrete Logarithm Problem in $\mathcal{G}$ and Random Oracle Model for hash function .

2.4.4 Schnorr Identification Scheme

As a part of PACE PoP also a slightly modified version od Schnorr Identification Scheme is used (formulation based on [20]). The setting for the user wanting to identify himself is the same as in the Schnorr Signature - he possesses a pair $(sk, pk = g^{-sk})$ of private and public keys registered in a trusted third party. Now the steps for the identification in front of a verifier are the following:

1. User chooses $r \in \mathbb{Z}_q \backslash \{0\}$, computes $X := g^r$ and sends it to the verifier.
2. Verifier chooses $e \in \mathbb{Z}_q \backslash \{0\}$ and sends it to the user.
3. User computes $y := r + sk \cdot e \ (mod\ q)$ and sends it to the verifier.
4. Verifier accepts if $X = g^y \cdot pk^e$, because for the correct data, $g^y \cdot pk^e = g^r \cdot g^{sk \cdot e} \cdot g^{-sk \cdot e} = g^r = X$.

Security of the scheme is based on the hardness of the Discrete Logarithm Problem in $\mathcal{G}$.

2.4.5 AES-CMAC

Message Authentication Code is a tag for a message M that is created with a secret key. Its task is to ensure the integrity of the message.

For Message Authentication Codes (MACs), PACE specification [2] allows a couple of algorithm options, but recommends only using the Advanced Encryption Standard in CMAC mode with different key lengths and 8 byte octets of tag length. Here the specification of this MAC is presented, following NIST Special Publication 800-38B [21] which consist of two procedures: sub-keys generation from key K and tag generation.

Sub-keys generation

1. Let $L = \mathbf{Enc}(K, 0^b)$, where b is a block length.
2. If $MSB(L) = 0$, then $K_1 = L \ll 1$, else $K_1 = (L \ll 1) \oplus R_b$, where MSB stands for the most significant bit in the bit string and R_b is a bit string specified in [21] depending on the block length.
3. If $MSB(K_1) = 0$, then $K_2 = K_1 \ll 1$, else $K_2 = (K_1 \ll 1) \oplus R_b$.
4. Return K_1, K_2

Tag generation for a message M of length M_{len} in bits

1. Apply the sub-key generation process to produce K_1 and K_2.
2. If $M_{len} = 0$, let $n = 1$, else let $n = \lceil M_{len}/b \rceil$
3. Let $M_1, M_2, \ldots, M_{n-1}, M_n^*$ denote the unique sequence of bit strings such that $M = M_1||M_2||\ldots||M_{n-1}||M_n^*$, where $M_1, M_2, \ldots, M_{n-1}$ are complete blocks.
4. If M_n^* is a complete block, let $M_n = K_1 \oplus M_n^*$; else, let $M_n = K_2 \oplus (M_n^*||10^j)$, where $j = nb - M_{len} - 1$.
5. Let $C_0 = 0^b$.
6. For $i = 1$ to n, let $C_i = \mathbf{Enc}(K, C_{i-1} \oplus M_i)$.
7. Let $T = MSB_{T_{len}}(C_n)$, where T_{len} denotes the required length of the tag in bits.
8. Return T.

For the analysis, we treat an encryption algorithm as a family of random keyed permutations; observe, however, that for the presented scheme the output before truncation is (for messages of bit length greater than n) a lot shorter, which makes it not a suitable choice. In addition, decryption is not possible, as the output is truncated to 64 bits (and, as a matter of fact, is not needed). In PACE, according to the specification [2] tags are 64-bit long. Aside from those technical reasons, MAC generating algorithm resembles more a hash function than an encryption algorithm and also (intuitively) should behave like one to ensure an acceptable level of security. For these reasons, the tag-generating algorithm is modeled as a random oracle that takes as an input a pair (key, message) a returns a bit string of length 64.

MACs are donoted as $\mathbf{MAC}(key, message)$.

Chapter 3

PACE and PACE CAM security analysis

In this chapter, crucial security features for the PACE and PACE CAM protocols will be presented together and analyzed. This analysis is at the time in the peer review process and has not yet been published, however, the preliminary version of the research with draft proofs appeared as a conference paper [6] and includes security analysis of PACE GM. As the difference between PACE and PACE CAM is only an additional round of exchanged messages needed for chip authentication, many of the security features of PACE transfer very easily to PACE CAM while others require a closer look.

3.1 Description of PACE protocol

The acronym PACE stands for Password Authenticated Connection Establishment protocol. It is a protocol recommended in the ICAO specification [2] for eMRTDs - electronic machine readable travel documents. Its main purpose (in this context) is to establish secure keys between a chip in a travel document (e-passport, eID) and a terminal (reader) in order to exchange securely biometric data intended for border control. The keys are intended to be used in symmetric key encryption algorithms and for calculating message authentication code tags. The technical details of how the data are exchanged after successful session of PACE (e.g. encryption algorithms, communication specification) are out of the scope of PACE (and this thesis) because it is done **after** establishing a secure channel for data exchange with encryption and MAC keys.
There are two parties participating in the protocol. The objective is to execute the protocol authenticated with password π delivered by the chip owner and after a successful session to have the same keys established on both sides for the subsequent communication to send the biometric data.
Before going into details, it is worth mentioning that during a protocol run, the participants agree on a new generator in the group $\mathcal{G}$ as defined in Section 2.1.

Although there are two proposed options for establishing the new generator in the ICAO specification ([2]), in this thesis the focus is placed on the variant PACE GM - Generic Mapping as it is a base for PACE CAM and the proposed modifications. Also the security analysis is based on this choice. Other option is Integrated Mapping (outside the scope of this thesis).

PACE is build of the following steps (see Figure 3.1):

1. The password π is delivered to terminal by the chip user - it is done typically by scanning the machine readable zone (MRZ) by an optical reader or could be done by typing in the password.

2. The chip and the terminal create a key $K_\pi := H(\pi||0)$ based on the password π, where $||$ is the concatenation.

3. The chip selects uniformly at random a nonce $s \leftarrow \mathbb{Z}_q \setminus \{0\}$, encrypts it to $z := \mathbf{Enc}_{K_\pi}(s)$ and sends it to the terminal together with the algebraic group description (see Section 3.2.1).

4. The terminal performs decryption $s := \mathbf{Dec}_{K_\pi}(z)$.

5. (DH2Point) The parties then establish together a new common generator $\hat{g}$ using a Diffie-Hellman key exchange and the nonce s. The chip randomly chooses x_A, sends $X_A = g^{x_A}$, the terminal chooses x_B and sends $X_B = g^{x_B}$. Then they create on both sides the same $h = g^{x_A x_B}$, where the chip computes $X_B^{x_A}$ and the terminal calculates symmetrically $X_A^{x_B}$. After that, they compute $\hat{g} = h \cdot g^s$ (independently on each side). This procedure is called General Mapping.

6. Participants perform another Diffie-Hellman key exchange using the new common generator $\hat{g}$ to obtain a key K by randomly choosing exponents y_A and y_B and exchanging $Y_A = \hat{g}^{y_A}$ and $Y_B = \hat{g}^{y_B}$. The resulting key is $K = \hat{g}^{y_A \cdot y_B}$.

7. Participants derive new keys: $K_{\mathbf{Enc}}, K_{\mathbf{MAC}}, K'_{\mathbf{MAC}}$ from K using the hash algorithm on concatenation of K and subsequent natural numbers starting from one, e.g. $K_{\mathbf{Enc}} = H(K||1)$.

8. The chip and the terminal exchange tags (Message Authentication Codes) computed on the elements of the second Diffie-Hellman key exchange using the key $K'_{\mathbf{MAC}}$, namely tags $T_B = \mathbf{MAC}(K'_{\mathbf{MAC}}, (Y_A, \mathcal{G}))$ and $T_A = \mathbf{MAC}(K'_{\mathbf{MAC}}, (Y_B, \mathcal{G}))$. After checking the validity of the tag provided by the other party, it is confirmed that the Diffie-Hellman key exchange group element was not modified during the exchange (integrity of the message). Observe that the both parties are able to recompute the tag sent by the other party, as it done with the same session key $K'_{\mathbf{MAC}}$ and data sent over the open channel. When communication is not aborted by any of the parties of the protocol, it means that the main session key K is delivered correctly on both sides.

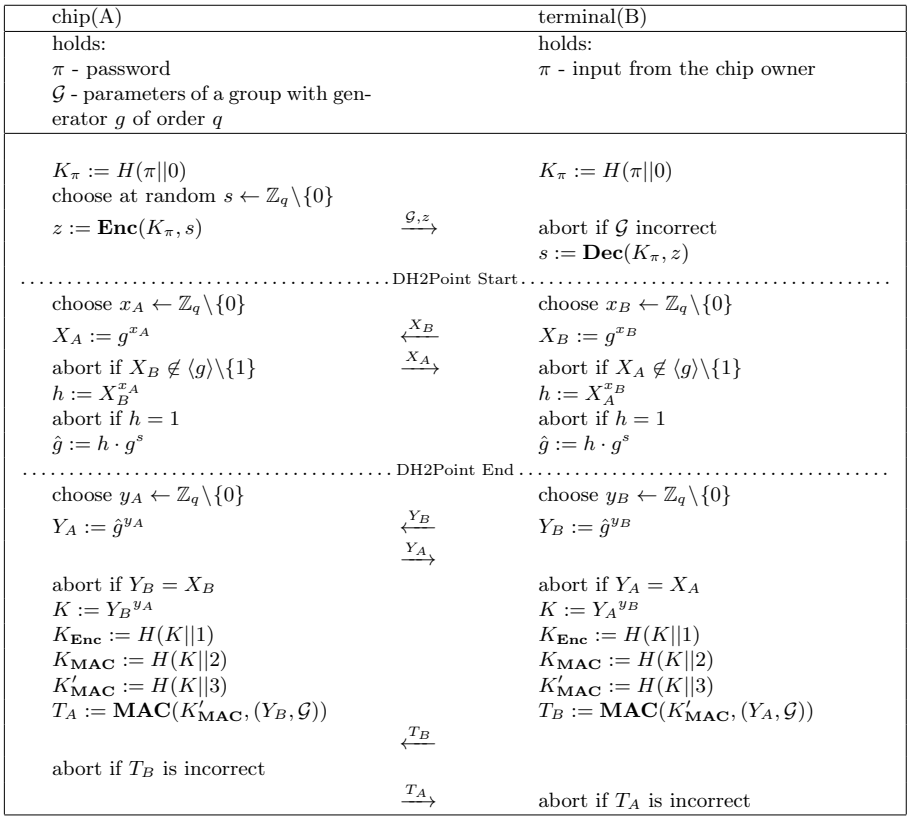

Figure 3.1: PACE protocol. The output key is $(K_{\mathbf{Enc}}, K_{\mathbf{MAC}})$. $\mathbf{Enc}(k, m)$ denotes a ciphertext of m obtained with key k, $\mathbf{MAC}(k, m)$ represents a MAC for m obtained with key k, H represents a hash function with image depending on the context.

9. If there was no rejection of the session from any side, the protocol session is complete and both sides derived keys $K_{\mathbf{Enc}}$ and $K_{\mathbf{MAC}}$ for subsequent communication.

The above list is a high-level description. In addition to that, there are many technical details behind the scenes, like the technical specification of the communication itself and also strictly defined messages exchanged with the description of the algorithms that will be used. It is always the chip that establishes the technical context for the session, as it is usually equipped with little computational power and which comes with a predefined set of operations and algorithms deployed on it, as opposed to the terminal that can be a more powerful device, connected to a network of computers (e.g. in an airport). In this thesis, the focus is on the security features of the protocol independent of the implementation and technical details, and more emphasis is placed on the security features behind the design.

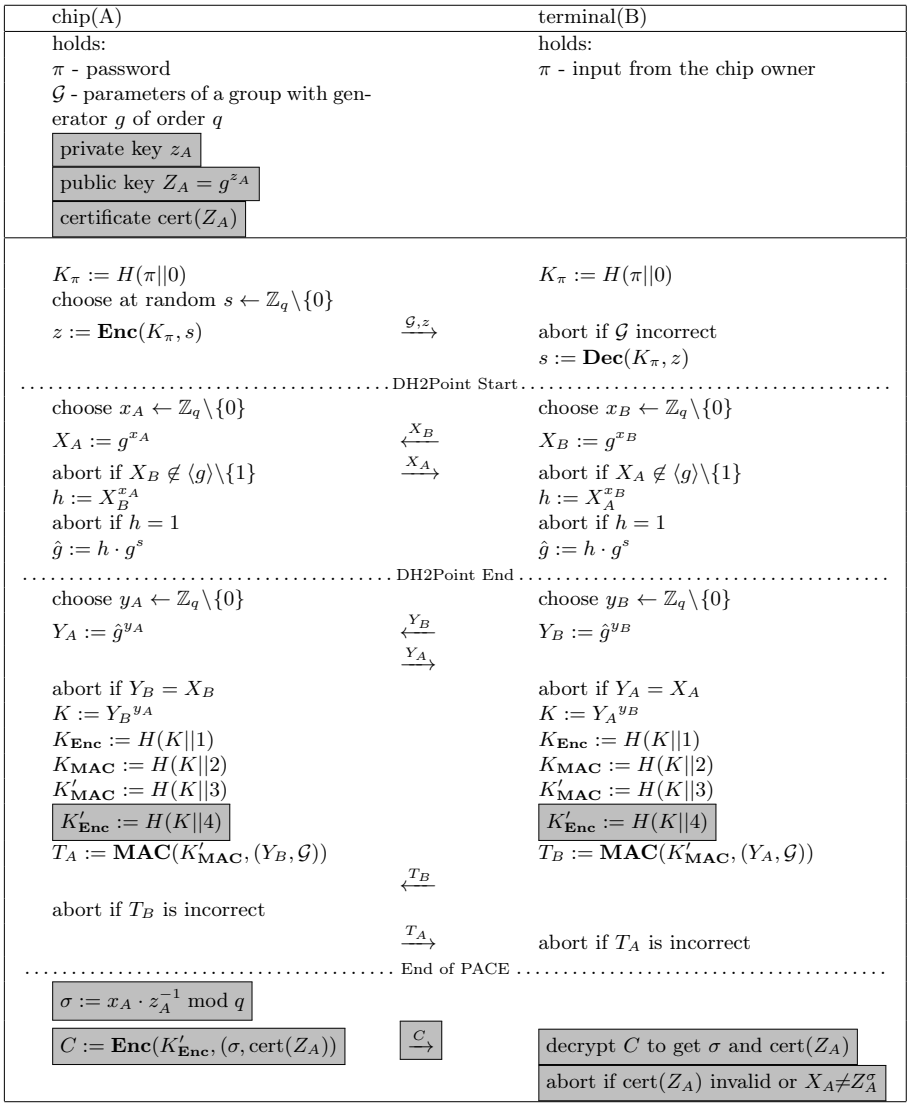

Figure 3.2: PACE and PACE CAM protocols. The output key is $(K_{\mathbf{Enc}}, K_{\mathbf{MAC}})$. CAM-specific extensions to the original PACE protocol are indicated in gray boxes. $\mathbf{Enc}(k, m)$ denotes a ciphertext of m obtained with key k, $\mathbf{MAC}(k, m)$ represents a MAC for m obtained with key k, H represents a hash function with image depending on the context.

3.2 PACE CAM

While for some scenarios, an authentication merely by means of knowing the right password π by the chip holder can be sufficient, it should be kept in mind

that for some applications the terminal can sometimes be only an intermediary in end-to-end communication between the chip and a remote server. In those cases, when the physical presence of the chip holder cannot be ensured, and authentication relies only on the fact of knowing a very low-entropy password, some stronger forms of authentication should be considered. Note also that a low-entropy password alone is a sufficient reason to look for other, stronger authentication options in many scenarios. There comes PACE CAM, a version of PACE with strong authentication of the chip. PACE CAM is designed to work with the General Mapping option to establish the common generator $\hat{g}$, however, there has been research on extending it to Integrated Mapping in the past [11] (see Section 1.4 for a short history of PACE CAM).

In PACE CAM, the chip possesses a pair of private, public keys: $(z_A, Z_A = g^{z_A})$ created in the same group $\mathcal{G}$ in which the protocol takes place. It also has $cert(Z_A)$, issued by a CA with a public known to the terminals, so that the terminals can verify the validity of the certificate.

PACE CAM adds an authenticating message sent from the chip to the terminal at the end of the original exchange of PACE. It requires the derivation of one additional key $K'_{\mathbf{Enc}} = H(K||4)$, some basic group operations, and one encryption more on the side of the chip. It reuses a message X_A to authenticate the chip using the secret key z_A. For reference, see Figure 3.2.

The design choices for PACE CAM can be an example of following good practices on how to extend a protocol - see Section 4.1.

3.2.1 Algorithms recommended by ICAO specification

Security analysis of PACE and PACE CAM is conducted using models for cryptographic primitives (building blocks) like random oracles for hash functions and MACs. In this subsection, the actual technical specifications recommended from [2] are recalled to understand what the models represent.

Groups

ICAO spefication allows:

1. *mod p* groups with 1024 bit p and 160 bit order of subgroup q.
2. *mod p* groups with 2048 bit p and 224 and 256 bit order of subgroup q.
3. Elliptic curve groups of field size from 192 to 521 bits.

Hash functions

The specification proposes two functions: SHA-1 and SHA-256. The actual choice depends on the other parameters in the system, namely what sizes of the generated keys are needed.

Message Authentication Codes

For MAC generating algorithm the following options are described:

1. DES3 in MAC mode according to [ISO/IEC 9797-1] with IV = 0.
2. AES-CMAC as described in Section 2.4.5 with key sizes of 128,192 and 256 bits.

3.3 Session Resilience and Active Adversaries

Throughout this section, the following scenario is considered:

Scenario 3.1 (Successful Session)**.** *A successful session of a protocol execution is an execution in which the terminal and the chip terminate in the accepting state and hold the same session key.*

It will be proven that if a session S is successful, then, with a small exception for PACE, the adversary could not manipulate the messages sent by the terminal and the chip during a protocol execution (Section 3.3.1). This means that the protocol is quite resilient to active attacks: (almost) any attempt to influence messages exchanged by the protocol participants by a man-in-the-middle results in a session failure.

In the next part of this section, it is shown that if a chip and a terminal establish a successful session, then the adversary controlling the communication between them cannot derive any information about the encryption key $K_{\mathbf{Enc}}$ (Section 3.3.2). This result is shown through a game in which the adversary has to distinguish between a random key and $K_{\mathbf{Enc}}$.

Another important issue to be discussed in this section is the reaction of protocol participants to deviations from a correct protocol execution. Recall that in many real-world systems, the most effective attacks are based on information that is leaked in case of failures and deviations from the regular behavior. Therefore, it is crucial to analyze the protocols from this point of view. According to the PACE and PACE CAM specification [2], a protocol execution is a chain of commands for which status codes are returned. In particular, the codes `0x9000` and `0x6300` are used. Their meaning is, respectively, *Normal processing - The protocol (step) was successful*, and *Authentication failed - The protocol (step) failed*. Thus, if one of the protocol parties receives a message that is inconsistent with the protocol specification (e.g., in the case that the other party sends an incorrect tag), then the information about the failure is sent back immediately. In general, a frequent recommendation for protocol designers is to avoid unnecessary explicit or implicit error messages, as they might provide valuable information to the adversary. For PACE and PACE CAM, the situation is relatively easy, since their designers postponed any message depending stochastically on the secret information to the final steps of the protocol. Before that moment, all messages are distributed uniformly independently of the password and the secret key of the chip (in the case of PACE CAM). This

property will significantly simplify the analysis of the protocol in case of a faulty execution.

3.3.1 Protocol Fragility

In this section, it is shown that PACE CAM (and to some extent PACE) is a protocol, for which any manipulation of messages exchanged by the chip and the terminal results in a session that is not successful in the sense of Scenario 3.1.

Definition 3.1 (Fragility)**.** *Assume that a protocol P is executed by a chip and a terminal so that an adversary $\mathcal{A}$ can arbitrarily modify any message sent by the chip and the terminal, as well as send its own messages. P is called fragile, if the probability of a session being successful is negligible in the case where the adversary has modified at least one message sent by the chip or the terminal during this session.*

The following result is essential to demonstrate that PACE and PACE CAM are resilient against active adversaries.

Theorem 3.2. *(a) PACE CAM is fragile.*

(b) PACE is fragile except for the following manipulation: the adversary $\mathcal{A}$ can raise messages X_A and X_B to an arbitrary exponent α known to $\mathcal{A}$.

Fragility is a property that reduces an active adversary to a passive one: while the adversary can always make the connection fail (e.g., by converting a message to random nonsense), he cannot have any influence on a successful session. Fragility turns out to be an important feature, since, among others, an adversary may attempt to retrieve some information on the private data (in particular, the private key of the chip, the password, and identity of the chip) by manipulating the messages and observing the reaction of the honest protocol participants. To this end, there are potentially many diverse strategies that can be applied by the adversary, and a security proof should take all of them into account. The fragility property implies that no matter what clever strategy is applied by the adversary, the result is the same with all but negligible probability (in practice, meaning "always"): the session fails independently of the private data used by the chip and the terminal. Although the failure message is always the same, the moment of reporting a failure can potentially leak some information. It will be proven that this is also not the case.

In this subsection, the possibilities of manipulating the messages by the adversary $\mathcal{A}$ are analyzed step by step. At each stage of the analysis, the set of manipulations for which $\mathcal{A}$ could potentially contradict the claim of Theorem 3.2 is narrowed down. Finally, there is only the single manipulation described in point (2) of Theorem 3.2 left.

In this subsection, it is assumed that the session considered is successful with a non-negligible probability, despite the actions of the adversary.

Manipulating messages Y_A, Y_B

First, it is assumed that Y_A or Y_B or both are manipulated by $\mathcal{A}$. In addition to that, other messages (e.g., X_A, X_B, and z) can also be manipulated.

Chosen approach in this analysis is to create an environment with carefully designed simulations of the protocol, with an observer having full control over the simulation and having access to all data generated by the simulation. The purpose of the simulation, in which $\mathcal{A}$ is used as a kind of plug-in procedure, is to break some hard cryptographic assumption. In this way, the impossibility of $\mathcal{A}$ with the properties claimed is proven.

Now, take a closer look at protocol executions with adversary $\mathcal{A}$ manipulating the messages exchanged by the chip and the terminal. Here, it is assumed that the probability of the successful session despite the manipulations exceeds some tolerance threshold (e.g., 2^{-20}), and this probability is denoted by p.

Assume that, among possibly other messages, $\mathcal{A}$ modifies Y_A, i.e. the terminal gets $Y'_A \neq Y_A$. Consequently, the tag T_B calculated by the terminal is equal to $\mathbf{MAC}(K'_{\mathbf{MAC}}, (Y'_A, \mathcal{G}))$. On the other hand, $\mathcal{A}$ must deliver the tag $T'_B = \mathbf{MAC}(K'_{\mathbf{MAC}}, (Y_A, \mathcal{G}))$ to the chip. Otherwise, the chip would abort the session.

Now it will be shown that the adversary can convert T_B to a valid tag T'_B, only if he can gain some knowledge about the key K. In this proof the focus is placed on 256-bit keys for creation of the tags, although other options are also permitted by the standard (however, it will be seen that security margin for them is not as good as one might wish).

Now, the correlation between the adversary's knowledge on the key K and the ability to produce a valid tag will be analyzed. The adversary receives a tag created with the key $K'_{\mathbf{MAC}} = H(K||3)$. Here, the function $H'(x) = H(x||3)$ is modeled as a standard random oracle, where x is the query to the oracle. As H outputs 256-bit strings, it gives no more than 2^{256} possible keys $K'_{\mathbf{MAC}}$. However, the size of the image of H' depends on the size of its domain, which is the number of all possible keys K, which is q, the order of the subgroup of $\mathcal{G}$. A simple calculation: for a given k bit string, the probability that it is not in the image of H' equals $(1 - \frac{1}{2^k})^q$. By linearity of expectation, the expected value of the number of different bit strings of length k obtained in this way is $2^k(1 - (1 - \frac{1}{2^k})^q)$. Using the expected value, one can find that for $q \geq 2^{159}$ the expected size of the image of H' is at least 2^{158}. The choice is not accidental, as the smallest "q" PACE specification allows is a 160 bit number, which means that allowable $q's$ are greater than 2^{159}. There are of course other, more secure options including groups based on elliptic curves with a group order up to 521 bit numbers, however, using here the smallest value gives a better insight into the security of the protocol. Now, one can use Chebyshev inequality $\Pr(|X - \mathbb{E}[X]| \geq \epsilon) \leq \frac{\mathbb{VAR}[X]}{\epsilon^2}$ to derive that with a probability $1 - 2^{-20}$ the number of the keys in the image of H' will be in an interval $[2^{158} - 2^{138}, 2^{158} + 2^{138}]$, which is more than 2^{157}. Variance was calculated from standard formula: $\mathbb{VAR}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$ and is exactly 2^{256} for $q = 2^{160}$. Although it is not the exact value of q, any deviation easily cancels out by the right choice of ϵ.

The construction of the MAC is based on a symmetric encryption scheme, however it does not behave as a family of random permutations - there is no need to reverse the computation of the tag like there is in case of encryption. Moreover, the values are truncated to 64 bits. From now on, the MAC generating algorithm is modeled as a random oracle (as was also discussed in Section 2.4.5), which on a new input (K, x) returns uniformly a random bit string of length 64 and for pairs (K, x) seen before it returns already assigned bit string. The tags are 64 bits long, so a randomly chosen key (but different from $K'_{\mathbf{MAC}}$) yields T_B on input $Y'_A, \mathcal{G}$ with probability 2^{-64}. Let $\mathcal{T}$ denote the set of keys that produce T_B. Because of the assumption that the tags are based on a random oracle one may conclude that with probability $1 - 1.244 \cdot 10^{-15}$ the set $\mathcal{T}$ contains at least $2^{93} - 2^3 \cdot 2^{46,5} > 2^{92}$ keys. This number is a result of approximating Binomial distribution that comes up when choosing the keys giving the right tag by the Gaussian distribution and using 8 standard deviations (the choice is motivated by the low probability of the complementary event). This approximation is appropriate, because of a satisfied condition that everything within 3 standard deviations of its mean ($\mu \pm 3\sigma$) is within the range of possible values - range $(0, n)$ where in this case n is the maximum number of "successes" - computing the right tag for the chosen key.

For a moment, assume that $\mathcal{A}$ is very powerful and capable of finding the keys from the set $\mathcal{T}$. The adversary's problem is that the tag T'_B, expected by the chip, is based on an input different from T_B (Y_A replaces Y'_A). As it is assumed that the algorithm for creating the tags is based on a random oracle, the probability that a given key from $\mathcal{T}$ creates the tag T'_B expected by the chip is 2^{-64}. The only exception is the key $K'_{\mathbf{MAC}}$ computed by the chip, where the probability equals 1. Nevertheless, as $|\mathcal{T}| \gg 2^{64}$, the probability that $\mathcal{A}$ would pick the right key from $\mathcal{T}$ to create T'_B differs from 2^{-64} by a negligible value, unless $\mathcal{A}$ has some additional knowledge from protocol execution. Observe that 2^{-64} is actually the success probability of a blind brute force attack, where $\mathcal{A}$ chooses a 64-bit string at random and sends it as the tag T'_B.

As $\mathcal{A}$ has the probability p of having a successful session, it means that, in the random oracle model for the hash function and for computing the tag, with probability $p' \approx p$, $\mathcal{A}$ has to derive K and thereby $K'_{\mathbf{MAC}}$.

Now it is a moment to simulate a protocol execution with $\mathcal{A}$ aiming to have a successful session and where $\mathcal{A}$ changes Y_A as discussed above. The execution of the protocol is emulated as follows:

- First, the execution of the protocol is emulated on behalf of the chip, the terminal, and $\mathcal{A}$, up to the moment when the terminal creates Y_B. (During this simulation, $\mathcal{A}$ can, in particular, manipulate X_A, X_B, and z).
- The terminal creating Y_B and $\mathcal{A}$ creating Y'_B are emulated. However, Y_B is not created by choosing y_B and then computing $Y_B := \hat{g}^{y_B}$. Instead, Y_B is taken as an external input.
- The chip chooses y_A at random and calculates $Y_A = \hat{g}^{y_A}$ (where $\hat{g}$ is computed according to the chip's point of view).

- The chip, the terminal, and $\mathcal{A}$ proceed with the execution of the protocol. The chip and the terminal use the key K, where $K = (Y'_B)^{y_A}$

  (so the terminal gets the key computed on the side of the chip from the simulation and does not compute it itself).

  $\mathcal{A}$ continues according to his strategy and his point of view on the execution of the protocol.

The next step is to set up an instance of the Inverse Computational Diffie-Hellman Problem, where a random Y_B is put on the side of the terminal without knowing its discrete logarithm (like described in the simulation).

Observe that the terminal would not be able to derive K, however, for the purpose of the simulation, it gets the key K from the chip. Such a setting may definitely create a discrepancy between the real executions corresponding to Y_B and all other parameters already exchanged and the simulation. The difference occurs if in the real case the session would be not sucessful due to different key K obtained by the terminal and by the chip. In these cases the derivations described below may fail or provide false results. However, with probability p, the session will be successful and the rest of the simulation will correspond to a real protocol execution.

As observed above, in case $\mathcal{A}$ succeeds in sending T'_B accepted by the chip, then except for a small probability $\mathcal{A}$ must be able to derive K. In this case, due to Assumption 2.5 (KEA1) the strategy of $\mathcal{A}$ enables the adversary to extract $y = \log_{\hat{g}} Y'_B$ with a non-negligible probability. In this case, the observer can retrieve y from $\mathcal{A}$, as he is in full control of each part of the simulation. This is the key point to break Assumption 2.4 (Inverse Computational Diffie-Hellman Assumption) with a non-negligible probability.

Now, see how the data from the simulation can be used. Recall that $K = (Y'_B)^{y_A}$ while at the same time $K = (Y'_A)^{y_B}$, since the session has to be successful. So $(\hat{g}^y)^{y_A} = (Y'_A)^{y_B}$, and $Y'_A = \hat{g}^{y \cdot y_A / y_B}$. Therefore, taking the data from the chip and $\mathcal{A}$, one can calculate $\hat{g}^{1/y_B}$. Recall that $Y_B = \hat{g}^{y_B}$ and Y_B have been arbitrarily chosen for the simulation, without knowing y_B. Thus, the conclusion is that the simulation using $\mathcal{A}$ enables to break Assumption 2.4.

Note 3.2.1. *It may happen that $Y'_B = Y_B$. In this case, the argument could be simplified, as one could directly obtain the discrete logarithm of Y_B.*

The conclusion is then as follows:

Claim 3.2.1. *Assume the random oracle model for the hash function and for the creation of tags. Then if the adversary can manipulate Y_A so that the session will be successful with probability $2^{-64} + \delta$ where δ is not negligible, then one can break Assumption 2.4 with a non-negligible probability.*

Note 3.2.2. *Observe that there is always a very small chance close to 2^{-64} that the adversary will guess the tag without knowing the key, which comes from the fact that the tags are 64 bits long.*

It can encompass situations where the adversary has done some manipulations that would not break the session, that is, the chip and the terminal would

establish the common key K, but the adversary would not be able to compute it himself. (For example, $\mathcal{A}$ can set $Y'_A := Y_A^\theta$ and $Y'_B := Y_B^\theta$ for an arbitrary θ and later guess the tags T'_B and T'_A.)

It could be disputed by some if the probability 2^{-64} is small enough to call it "negligible", however, in practical scenarios it should be enough as real-world attackers rarely have that much resources as assumed in the presented security analysis. Also, it is now somehow enforced to deem that sufficient as there is no other option for length of the tags in PACE (CAM) at the present moment in the specification [2] when it comes to AES-CMAC. In this interactive protocol, it is very likely that the terminal will also have a timeout set at some particular value to wait for a message from the chip excluding lengthy, computing heavy attacks. Similarly, there should be a limit for the number of sessions that a particular chip can start in a period of time. In practice, this excludes options for exhaustive search for the adversary as the time for manipulation is very limited and there is no second try as all the sessions are different. For those reasons, from now on the term ***practically negligible*** *will be used when the situation calls for it.*

Finally, what is left to inspect is the remaining case, where $Y_A = Y'_A$ but $Y'_B \neq Y_B$. In this case, $\mathcal{A}$ simply forwards T_B to the chip, as this tag has the correct form for the chip. Moreover, in this case, the chip will send T_A. Now, $\mathcal{A}$ must produce a tag T'_A for a different input than T_A that will be accepted by the terminal. $\mathcal{A}$ has two valid tags at hand: T_B for $(Y_A, \mathcal{G})$ and T_A computed for $(Y'_B, \mathcal{G})$. Here the analysis goes in a little bit different way - it is not assumed that the adversary can find the keys from the set $\mathcal{T}$, which at this point is of size approximately 2^{27} (by analogous computations as before). Here assume more realistic capabilities of the adversary, where without any extra power he needs to search through the whole set of keys of the cardinality greater than 2^{157} to find the key producing the valid MAC. The tags presented by the chip and the terminal may help in fishing out the candidate keys, but without knowing K the success probability is about $2^{27}/2^{157} = 2^{-130}$ in a single trial. Therefore, it is infeasible to find a matching key K even if the adversary can compute a large number of hashes (e.g. 2^{80}). However, even if a key $K \in \mathcal{T}$ is found, the probability that it is the right one is about 2^{-27} and the tag generated by the adversary will be correct is close to $2^{-27} + (1 - 2^{-27}) \cdot 2^{-64} \approx 2^{-27}$. This leads to the same conclusions as before.

To summarize:

Claim 3.2.2. *If $\mathcal{A}$ manipulates at least one of the values Y_A, Y_B, then the session will be successful with a practically negligible probability.*

It follows that if an active adversary $\mathcal{A}$ aims to obtain a successful session, then $\mathcal{A}$ has to let the messages Y_A and Y_B remain unchanged. Thus, the first step of narrowing down the possibilities left for $\mathcal{A}$ is completed. Still, there is a need to consider possible manipulations of the former messages. However, one thing is already proven: the possible discrepancies between the chip and the terminal cannot be compensated for by manipulations of Y_A and Y_B!

Common value of $\hat{g}$

After the first DH key exchange, both the chip and the terminal calculate $\hat{g}$. As $\mathcal{A}$ can arbitrarily modify and replace messages, it can happen that the values of $\hat{g}$ computed by the chip and the terminal are different. Let the value of $\hat{g}$ calculated by the chip be called $\hat{g}_A$, while the value of $\hat{g}$ calculated by the terminal be called $\hat{g}_B$.

First, note that if $\hat{g}_A \neq \hat{g}_B$, then the chip and the terminal cannot observe the same values of Y_A and Y_B in a successful session.

Indeed, assume that the chip and the terminal observe the same Y_A and Y_B but $\hat{g}_B = \hat{g}_A^{\delta}$, for $\delta \neq 1$. Let $Y_A = (\hat{g}_A)^{y_1}$ and $Y_B = (\hat{g}_A)^{y_2}$. The chip then computes the key K as $Y_B^{y_1} = (\hat{g}_A)^{y_1 \cdot y_2}$. On the other hand, the terminal must use y' such that $Y_B = (\hat{g}_B)^{y'}$. Since $y' = y_2/\delta$, the terminal would calculate the key K equal to $(Y_A)^{y_2/\delta} = (\hat{g}_A)^{y_1 \cdot y_2/\delta}$. So in this case, the session is not successful, as the chip and the terminal derive different values for the session key K.

At this moment compare the above conclusion with Claim 3.2.2: if $\hat{g}_A \neq \hat{g}_B$, then on one hand the values Y_A or Y_B must be manipulated to obtain a successful session, while on the other hand, they must be equal to obtain a successful session with all but negligible probability. Therefore, the conclusion goes as follows:

Claim 3.2.3. *If an adversary $\mathcal{A}$ modifies the messages exchanged by the chip and the terminal so that $\hat{g}_A \neq \hat{g}_B$, then the resulting session is successful with practically negligible probability.*

Again, the number of choices for the adversary is narrowed down: it cannot be excluded that $\mathcal{A}$ manipulates z, X_A, X_B, but to get a successful session, the chip and the terminal must calculate the same value for $\hat{g}$. It shall be seen that this is a severe limitation for the adversary $\mathcal{A}$.

The first DH key exchange

Now the first Diffie-Hellman key exchange will be considered, keeping in mind Claim 3.2.3. Of course, $\mathcal{A}$ can manipulate the messages X_A, X_B, and z before they are delivered. Let X'_A, X'_B, and z' denote these messages after (possible) modifications by $\mathcal{A}$. Let α, β denote the numbers such that $X'_A = X_A^{\alpha}$, $X'_B = X_B^{\beta}$. Observe that these numbers exist as the group used by the protocol is cyclic. At this point, it is not assumed that $\mathcal{A}$ is aware of α and β – maybe the modifications are performed in a way different from raising to a known power. However, observe that the chip computes

$$\hat{g}_A := (X'_B)^{x_A} \cdot g^s = g^{x_A \cdot x_B \cdot \beta} \cdot g^s$$

while the terminal computes

$$\hat{g}_B := (X'_A)^{x_B} \cdot g^{s'} = g^{x_A \cdot x_B \cdot \alpha} \cdot g^{s'}$$

where s' is a plaintext obtained from z'. As $\hat{g}$ calculated by the chip and the terminal must be the same acccording to Claim 3.2.3:

$$x_A \cdot x_B \cdot \beta + s = x_A \cdot x_B \cdot \alpha + s' \bmod q$$

and hence

$$\beta = \alpha + \tfrac{s'-s}{x_A \cdot x_B} \bmod q. \tag{3.1}$$

Claim 3.2.4. *If $s' = s$ mod q (that is, z has not been manipulated by $\mathcal{A}$), then $\alpha = \beta$ mod q and $\mathcal{A}$ can extract α with all but negligible probability.*

Proof. By (3.1), $\alpha = \beta$, if $s' = s$. Note that for given X_B, X'_B and X_A, where X_B and X_A have been created at random, $\mathcal{A}$ must create X'_A so that $\log_{X_A} X'_A = \log_{X_B} X'_B$. Otherwise, the resulting $\hat{g}_A$ and $\hat{g}_B$ would be different. At this moment, Assumption 2.5 (KEA) is applied, where X_B and X'_B play the role of, respectively, g and g^a, X_A plays the role of C, and X'_A plays the role of Y. Therefore, by Assumption 2.5 adversary $\mathcal{A}$ can extract $\log_{X_A} X'_A = \alpha$ with all but negligible probability. (In fact, in the scenario from Assumption 2.5 the adversary has more freedom, as he may choose arbitrary C while here C is fixed by the protocol execution.) □

According to Claim 3.2.4, if z is not manipulated, then the only manipulation that has the chance to lead to a successful session is to deliver X_A^α and X_B^α instead of X_A and X_B, where α is known to the adversary. As stated in Theorem 3.2, this is possible for the PACE protocol. Later it will be shown that for PACE CAM, even this manipulation results in a not successful session.

Modifications of the ciphertext z

As it has been seen, if z is not manipulated, then the possibilities to manipulate X_A and X_B are quite limited. Now for the remaining case, where the adversary has modified the ciphertext z:

Claim 3.2.5. *Assume that $s \neq s'$ (since either $z \neq z'$ or the chip and the terminal hold different keys K_π due to different passwords used), but the session remains successful with a non-negligible probability. Then $\mathcal{A}$ can break Assumption 2.4 - the Inverse CDH Problem.*

Proof. An environment for solving the Inv-CDH Problem will be created. In this environment, the observer controls not only $\mathcal{A}$, but also the terminal and the chip. Given a problem instance g^y, put $X_B = g^y$ (which results in $x_B = y$). The response of the chip X_A is set to X_B^r for r chosen at random (consequently $x_A = y \cdot r$). It can be done because the chip is controlled as well. Furthermore, the probability distribution of X_A from the point of view of $\mathcal{A}$ is the same as in a genuine protocol execution, so $\mathcal{A}$ should be able to attack the protocol as for a genuine execution. The adversary $\mathcal{A}$ somehow creates $X'_B = X_B^\beta$ and $X'_A = X_A^\alpha$ (where α and β are not necessarily known to $\mathcal{A}$, but $\alpha \neq \beta$ according to Eqn. (3.1)). Note that

$$X'_B = X_B^\beta = (X_B)^{\alpha + \frac{s'-s}{x_A x_B}} = (X_B)^\alpha \cdot g^{\frac{s'-s}{x_A}} = (X'_A)^{\frac{1}{r}} \cdot g^{\frac{s'-s}{y \cdot r}}$$

Therefore

$$g^{\frac{1}{y}} = ({X'_B}^r \cdot (X'_A)^{-1})^{\frac{1}{s'-s}} \tag{3.2}$$

Equation (3.2) can be used to compute $g^{\frac{1}{y}}$, since the data on the right hand side are available in the simulation. □

Note 3.2.3. *Note that in the simulation used in the proof of Claim 3.2.5, the execution after the first DH key exchange is not continued. In fact, it is impossible, as the chip and the terminal do not know y. However, the goal was to exploit the power of the adversary, for whom the execution was not distinguishable from the one, where the chip and the terminal follow the protocol and know the discrete logarithms of X_B, X_A.*

From Claim 3.2.5 the next Claim follows:

Claim 3.2.6. *If $\mathcal{A}$ manipulates z so that the plaintext is changed, then the resulting session is not successful with all but practically negligible probability.*

At this moment, it can be concluded that Theorem 3.2(b) concerning PACE has been proven. It remains to take care of the case of PACE CAM, where there is a stronger result.

The case of PACE CAM

Now consider again the first key exchange and modifications of X_A. It is known that the only strategy of $\mathcal{A}$ that does not lead to an unsuccessful session with all but practically negligible probability is to raise X_A and X_B to the same known power, say α, and to deliver z, Y_A, Y_B without any manipulation.

Recall that at the last stage of PACE CAM, a ciphertext C for the plaintext $x_A/z_A \bmod q$ is sent from the chip to the terminal. So, if $\mathcal{A}$ replaces X_A by $X'_A = (X_A)^\alpha$, in the early stage of the protocol execution, then finally $\mathcal{A}$ has to replace the ciphertext C by a ciphertext for $\alpha \cdot x_A/z_A$ to avoid rejecting the session by the terminal.

For the sake of protocol analysis, consider an artificial protocol $\mathcal{F}$, which is the same as PACE (CAM), except that instead of computing K as described in the protocol specification, the key K is chosen at random in the same way on the terminal's side and on the chip's side. Although in reality such a protocol $\mathcal{F}$ cannot be executed, it helps to analyze the capabilities of the adversary.

Claim 3.2.7. *If $\mathcal{A}$ has a non-negligible advantage in recognizing whether PACE (CAM) or $\mathcal{F}$ is executed, then $\mathcal{A}$ will have a non-negligible advantage in solving the Decisional Diffie-Hellman Problem (Assumption 2.3).*

In this subsection we focus on PACE CAM, however, later there will be references to Claim 3.2.7 for PACE. For this reason, the formulation is slightly more general.

Proof. Given a DDH Problem instance (g^a, g^b, L) emulate the protocol until the moment when Y_A and Y_B are sent. Then put $Y_A = g^a$, $Y_B = g^b$. The simulation

knows m such that $\hat{g} = g^m$. Hence, K should be equal to $L^{1/m}$ in the case where (g, g^a, g^b, L) is a Diffie-Hellman tuple. The rest of simulation is run with the key $K = L^{1/m}$ the messages are presented to $\mathcal{A}$.

Now, to solve the DDH problem, $\mathcal{A}$ is asked whether the genuine protocol or $\mathcal{F}$ has been executed. If $\mathcal{A}$ is able to respond correctly with probability non-negligibly greater than $\frac{1}{2}$, then solving the DDH problem is possible with the same non-negligible advantage. □

By Claim 3.2.7, it suffices to consider the protocol $\mathcal{F}$. The next artificial protocol $\mathcal{F}'$ is defined, where instead of deriving the remaining keys from K according to the protocol description, the chip and the terminal choose the same values of $K'_{\mathbf{MAC}}$ and $K_{\mathbf{Enc}}$ at random, where the choices for $K_{\mathbf{Enc}}$ and $K'_{\mathbf{MAC}}$ are stochastically independent.

Claim 3.2.8. *$\mathcal{A}$ has at most a negligible advantage in recognizing whether $\mathcal{F}'$ or $\mathcal{F}$ is executed.*

Claim 3.2.8 is based on the observation that otherwise Assumption 2.6 would be broken. Indeed, given candidate keys K_1, K_2 run the PACE CAM protocol with $K'_{\mathbf{MAC}} = K_1$ and $K'_{\mathbf{Enc}} = K_2$ – skipping the derivation steps for $K'_{\mathbf{MAC}}$ and $K'_{\mathbf{Enc}}$. Therefore, $\mathcal{A}$ observes an execution of either $\mathcal{F}$ or $\mathcal{F}'$. In this situation, any advantage of $\mathcal{A}$ to distinguish between these cases would mean an advantage for deciding whether there is a key K such that $K_1 = H(K||3)$, $K_2 = H(K||4)$.

For $\mathcal{F}'$ the encryption key used to create C is random. So $\mathcal{A}$ is faced with the following question: for a random key $K'_{\mathbf{Enc}}$ is it possible to derive $C' = \mathbf{Enc}(K'_{\mathbf{Enc}}, (\sigma/\alpha, \mathrm{cert}(Z_A)))$ given $C = \mathrm{Enc}(K'_{\mathbf{Enc}}, (\sigma, \mathrm{cert}(Z_A))$, α, and $\mathrm{cert}(Z_A))$. Even if $\mathcal{A}$ was granted the additional knowledge of σ, then in fact $\mathcal{A}$ cannot compute the required ciphertext with probability greater than negligible without the knowledge of the key, because encryption is modeled as the random keyed permutation model. Therefore, it can be concluded that except for a negligible probability, the session of $\mathcal{F}'$ would be not successful if $\alpha \neq 1$. Summarizing:

Claim 3.2.9. *If $\mathcal{A}$ executes PACE CAM so that X_A and X_B are replaced by X_A^α and X_B^α for $\alpha \neq 1$, then the session is successful only with a negligible probability.*

Claim 3.2.9 concludes the proof of Theorem 3.2(a). □

3.3.2 Key Confidentiality

Definition 3.2. *Consider the following game executed by Challenger and an adversary $\mathcal{A}$. Challenger is responsible for acting as the chip and the terminal of PACE (resp. PACE CAM) while $\mathcal{A}$ is an active adversary controlling the communication between the chip and the terminal. If either the chip or the terminal rejects the session or they derive different keys $K_{\mathbf{Enc}}$, then $\mathcal{A}$ looses the game immediately. Finally, the following steps are executed:*

step 1: *Challenger chooses a bit b uniformly at random,*

step 2: *Challenger presents Λ to $\mathcal{A}$, where $\Lambda = K_{\mathbf{Enc}}$ if $b = 1$, otherwise Λ is generated uniformly at random from the key space,*

step 3: *$\mathcal{A}$ presents a bit b'.*

$\mathcal{A}$ wins the game, if $b' = b$. $\mathcal{A}$ breaks confidentiality, if the probability to win by $\mathcal{A}$ is $\frac{1}{2} + \epsilon$ where advantage ϵ is non-negligible.

Note that according to Definition 3.2 $\mathcal{A}$ may win only in case of a successful session. Therefore, it does not cover the case that at the end of the session the session key is shared by the adversary and either the chip or the terminal.

Theorem 3.3. *For PACE and PACE CAM, the adversary $\mathcal{A}$ cannot break session confidentiality.*

Proof. From now on, assume that the session executed by the Challenger and $\mathcal{A}$ is successful. So, according to Theorem 3.2, $\mathcal{A}$ is passive (the case of PACE CAM), or can perform very limited manipulations (the case of PACE).

A potential complication for the Challenger and an opportunity for $\mathcal{A}$ is that the key $K_{\mathbf{Enc}}$ does not come alone – it is somehow related to the keys $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$ used for creating the messages exchanged by the chip and the terminal. Therefore, we must show that these potential information sources cannot be used by $\mathcal{A}$ to win the game.

The first observation is that $\mathcal{A}$ cannot distinguish between a genuine protocol execution and an artificial protocol execution $\mathcal{F}$, where K is chosen at random (the same value on the terminal side and on the chip side) instead of deriving the session key K according to the protocol specification. In fact, here Claim 3.2.7 applies from the proof of Theorem 3.2.

The next step is to consider an artificial protocol $\mathcal{F}'$, where the keys $K_{\mathbf{Enc}}$, $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$ are chosen at random instead of deriving them from K. $\mathcal{A}$ cannot distinguish between executions of $\mathcal{F}'$ and $\mathcal{F}$ with all but negligible probability. Indeed, otherwise given a tuple K_1, K_2, K_3, K_4, one could run the protocol ($\mathcal{F}$ or $\mathcal{F}'$) where K_1, K_2, K_3, K_4 would take the place of $K_{\mathbf{Enc}}$, $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$. Any non-negligible difference in protocol behavior observable by $\mathcal{A}$ would be used to break Assumption 2.6. This is possible since for $\mathcal{F}$ and $\mathcal{F}'$ the rest of the protocol is not related to the derivation of K.

For the game $\mathcal{F}'$, the key $K_{\mathbf{Enc}}$ is not correlated with the rest of the protocol, as it is for the second option considered in the game from Definition 3.2. However, the analysis went too far: the keys $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$ are independent, too. Fortunately, it is easy to step back: Let $\mathcal{F}''$ be the same as $\mathcal{F}'$, but now $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$ are derived from the same random K. Assumption 2.6 can be used again to show that $\mathcal{A}$ cannot distinguish between $\mathcal{F}'$ and $\mathcal{F}''$ with all but negligible probability. Now a potential distinguisher can be build by putting the tested keys K_1, K_2, K_3 in place of $K_{\mathbf{MAC}}$, $K'_{\mathbf{MAC}}$, $K'_{\mathbf{Enc}}$, but choosing the key $K_{\mathbf{Enc}}$ at random.

The last step is to convert the protocol $\mathcal{F}''$ to $\mathcal{F}'''$ where the key K is again not a random key but derived according to the protocol. At this moment, it can be argued as for Claim 3.2.7 that any advantage of $\mathcal{A}$ in distinguishing between $\mathcal{F}''$ and $\mathcal{F}'''$ could be used to break the Decisional Diffie-Hellman Problem (Assumption 2.3).

As presented, after starting with the genuine protocol, a sequence of steps was made, where each time the difference between the protocols is undetectable for $\mathcal{A}$ with all but negligible probability. The last protocol corresponds to the second option from Step 2 in the game from Definition 3.2. This concludes the proof. □

In particular, the following weak version of Theorem 3.3 can be formulated:

Corollary 3.3.1. *If the chip and the terminal establish successfully a session with a session key K, then K is not known to the adversary acting as man-in-the-middle between the chip and the terminal.*

Of course, Theorem 3.3 says much more than Corollary 3.3.1. Informally, it says that no useful property of $K_{\mathbf{Enc}}$ can be derived by $\mathcal{A}$ for a successful session. Therefore, it does not help $\mathcal{A}$ to break the ciphertexts exchanged by the chip and the terminal, except for a negligible probability.

Theorem 3.3 does not say that the adversary cannot establish a session with a chip or with a terminal. Note that in case of an interaction with the chip, $\mathcal{A}$ can establish a session, provided that $\mathcal{A}$ guesses the password used by the chip. In this case, the chip may reveal some data to $\mathcal{A}$, since the chip believes that it interacts with an authorized terminal. This and other situations are discussed in the next sections.

3.4 Password Security

In this section, we consider executions during which an adversary $\mathcal{A}$ interacts with a chip, or with a terminal, or simultaneously with a chip and a terminal sharing the same password. However, the assumption is that $\mathcal{A}$ does not know the password used by the other party or parties.

The adversary does not need to follow the protocol specification. Moreover, $\mathcal{A}$ may take advantage of messages exchanged with one party for communication with the other party. Note that the last case encompasses the scenario of a passive observer that monitors the communication between the chip and the terminal – in that case $\mathcal{A}$ simply forwards the messages sent by the chip and by the terminal. This also applies to all offline attacks, where transcripts of real interactions between the chip and the terminal holding the same password are analyzed.

In the scenario described, the adversary may have different goals. The following ones are now considered:

password breaking: learning the password used by the legitimate chip or terminal (or at least getting some information about the password),

session hijacking: setting up a session with the chip or with the terminal taking advantage of the fact that the chip and the terminal have initiated the communication using the same password,

tracing: it means not only recognizing that a given chip has participated in a given session, but generally getting any knowledge about the chips being active in observed interactions. For example, it might concern the question whether two observed executions correspond to the same chip or password.

In the following these scenarios are discussed in a bit more detail.

Breaking password

Note that in the case of PACE, there is an option for a brute-force attack against the password: an adversary $\mathcal{A}$ guesses the password and interacts with the chip or with the terminal that uses the correct password. A successful session in this case means that the guessed password is likely to be correct. If the attempted password π' is different from the password π used by the chip (or the terminal), then $K_\pi \neq K_{\pi'}$ with all but negligible probability. In this case, $\mathcal{A}$ uses a different value of s than used by the chip or by the terminal. Thus, by Claim 3.2.5 the session will not be successful with all but a negligible probability.

The situation is only slightly more complicated for PACE CAM, if the adversary plays the role of a chip (the case where the adversary plays the role of the terminal is the same as in the case of PACE). Guessing the password correctly would mean that $\mathcal{A}$ will be able to send the correct tag T_A, and the session will not be broken at this moment. However, it is extremely unlikely that $\mathcal{A}$ will be able to send the ciphertext C that would be accepted by the terminal, unless $\mathcal{A}$ knows the private key z_A of the chip. Anyway, a failure message received immediately after delivering T_A indicates that the password guess was incorrect, while the other situation means that the password guess was correct with very high probability.

In the described scenario, $\mathcal{A}$ can try the passwords one by one. As the entropy of the passwords is usually low, such an attack will eventually succeed. By a careful protocol design, it is a goal to achieve that the best strategy of the adversary to learn the password is to perform the brute force attack, where no more than one password can be effectively tested in one session. This would exclude, for example, offline attacks, where the adversary takes the transcripts of some interactions between the chip and the terminal using the same correct password and attempts to learn which passwords might have been used.

We shall see that for PACE and PACE CAM, we are (almost) in such an ideal situation.

Session hijacking

As we have shown in Section 3.3, except for one (useless) case, the adversary cannot modify the messages exchanged without breaking the session between

the chip and the terminal. However, it does not mean that the adversary cannot establish a session with either the chip or the server so that this party will believe that the session has been established with a party knowing the password. A situation of this kind would be considered a successful attack against password authentication, despite the fact that the adversary may still be unaware of the password value.

Note that session hijacking is a slightly more general problem than establishing a connection without knowing the password. It concerns any situation where a session is run by a legitimate party, but later the communication is taken over by the adversary that may even know the password used. This may correspond to a situation, where the holder of the chip enables it to start an interaction with a legitimate terminal (e.g. by typing in the password to the terminal or presenting the chip to the optical scanner of the terminal), but later the chip starts to interact with a different terminal, run by the adversary $\mathcal{A}$.

We shall see that in the case of PACE and PACE CAM, knowing the password does not help hijacking a session.

Tracking

Capturing the meaning of resilience to tracking attacks is a complex issue. One of the simplest scenarios is inability to decide whether an observed interaction corresponds to a given password. A slightly more complex scenario concerns the case where the adversary aims to decide whether two observed protocol executions correspond to the same password. However, there is a large palette of more sophisticated attack scenarios and goals: An adversary may interact with a chip not aiming to establish a session (we shall see that it is doomed to fail), but only to test the behavior of the chip and learn something about its identity. For example, an adversary may attempt to learn that two given executions correspond to *similar* passwords or that certain executions certainly do not correspond to the same password. The adversary can be both active (as in the scenario just mentioned) or passively operating a network of eavesdropping devices at public places.

We must be aware that the adversary may use some side channel information. For example, in some cases, the location and distance between the terminals, as well as execution times, may indicate the route of a user (or, conversely, exclude a route) without breaking the cryptographic protocol. Thus, all we may hope for is to show that the messages exchanged by the cryptographic protocol do not provide non-negligible additional knowledge to the adversary. In an ideal case, the protocol should guarantee that an adversary should not be able to distinguish between a genuine protocol execution from a fake protocol execution, where the participating parties exchange random messages.

Note that there is another tracking scenario, in which an adversary $\mathcal{A}$ holds the password of the tracked chip T. Of course, in this case, the capabilities of $\mathcal{A}$ are much stronger since the adversary may directly interact with the chips that are suspected to be T. These issues will be discussed in more detail in Section 3.5.1.

3.4.1 Adversary interacting with a chip

Let us consider the situation where an adversary $\mathcal{A}$ plays the role of a terminal and interacts with the chip without knowing the password π used by the chip. Note that this is the primary situation considered by the designers of the protocol – opening a session with a chip should be prevented unless the correct password is given to the terminal.

From now on, assume that Π is the set of all possible passwords. Without discussing the shape of Π we may assume that the cardinality of Π is relatively small. For example, for PACE used on Polish ID documents, the password is a 6-digit so-called CAN number. In biometric passports, the password is the personal data written on the so-called *machine readable zone* (MRZ). In the former case, this results in at most 10^6 possible keys for the encryption algorithm **Enc**. As the set of passwords is relatively small and the function H used for derivation $K_\pi := H(\pi\|0)$ should be a cryptographically strong hash function, we can safely make the following assumption:

Assumption 3.4. *We assume that the keys K_π for $\pi \in \Pi$ are all different.*

Note that if Π is a fixed set (such as the set of 6-digit CAN numbers), validity of Assumption 3.4 can be practically verified, and we did so using SHA256 and UTF-8 encoding for the CAN numbers. Note also that finding any hash function collision $K_\pi = K_{\pi'}$ would have far-reaching consequences, where security implications for PACE would be one of the least significant issues.

According to the specification, the nonce s is, in fact, coined as l-bit string, where l is a multiple of the block size of the chosen block cipher, so its entropy should be high.

Claim 3.4.1. *Assume that $\pi_0 \in \Pi$, $|\Pi| = 10^6$, the nonce s has been chosen at random, and $z = \mathbf{Enc}(K_{\pi_0}, s)$. Assume also that* **Enc** *is a family of random permutations on bit strings of length l, where each permutation $\mathbf{Enc}(K, -)$ is chosen independently, uniformly at random. Then the probability that*

$$\mathbf{Dec}(K_{\pi'}, z) = \mathbf{Dec}(K_{\pi''}, z) \tag{3.3}$$

for some $\pi' \neq \pi'' \in \Pi$ is practically negligible, i.e. finding such pair is almost impossible.

Proof. The probability that (3.3) does not hold equals

$$\prod_{j=1}^{10^6-1} \left(1 - \tfrac{j}{2^l}\right) > \left(1 - \tfrac{10^6-1}{2^l}\right)^{10^6-1} \tag{3.4}$$

Since 10^6 is negligibly small compared to 2^l, the value in (3.4) differs from 1 by a negligible value (e.g., for $l = 128$ the difference is of magnitude 10^{-26}). □

By Claim 3.4.1, we may assume that for the ciphertext z concerned, the plaintext $s = \mathbf{Dec}(K_\pi, z)$ is different for each password $\pi \in \Pi$ as the probability of the complementary event is negligible. As h is uniquely determined by X_A

and X_B, it follows that each choice of π would lead to a different value of $\hat{g} = \hat{g}(\pi)$ from the point of view of $\mathcal{A}$. Consequently, for each π there is a different $y_B(\pi)$ such that $Y_B = \hat{g}(\pi)^{y_B(\pi)}$. As $K = Y_A^{y_B(\pi)}$ on the side of $\mathcal{A}$, we may assume that for each $\pi \in \Pi$, the value of $K = K(\pi)$ is different with all but a negligible probability.

Recall that z, X_A, and Y_A are stochastically independent of the password used. The first message sent by the chip that depends on the password used by the chip is T_A. Before T_A is created by the chip, $\mathcal{A}$ must present T_B that depends on the MAC key $K'_{\mathbf{MAC}}$. However, $K'_{\mathbf{MAC}} = H(K||3)$, so it depends directly on $K = K(\pi)$ used. The probability that for two different values of K the same $K_{\mathbf{MAC}}$ will be derived is negligible because in random oracle model the probability of finding collisions is negligible. The number of the possible keys is many orders to small for the birthday paradox to occur (that would require e.g. approximately 2^{80} trials when there are 2^{160} possible keys). Thus, we may assume that each $K'_{\mathbf{MAC}}(\pi)$ is different. As the input of T_B is fixed, the probability that for $K_{\mathbf{MAC}}(\pi') \neq K_{\mathbf{MAC}}(\pi'')$, we get the same value of T_B is negligible. This comes from the fact that the number of possible keys $K_{\mathbf{MAC}}(\pi)$ is too small for the birthday paradox to occur as there is only 10^6 possible keys and 2^{64} possible outputs of the $\mathbf{MAC}(K, x)$ algorithm (remember that we model tag generating algorithm as a random oracle). Therefore, T_B sent by the adversary is bound to at most one value of π. Moreover, if $\pi \neq \pi_0$, then the chip will find that the tag T_B is invalid and will reject the session. In conclusion, apart from a probability $2^{-64} + \delta$, where δ is negligible, the only way to execute a successful protocol by $\mathcal{A}$ as the terminal is to use the correct π_0 on his side.

This discussion leads to the following result:

Theorem 3.5. *If adversary $\mathcal{A}$ playing the role of a terminal does not know the password π_0 used by the chip, then with all but a practically negligible probability, an interaction with the chip may result in one of the two following situations:*

1. *$\mathcal{A}$ correctly executes the protocol using the password π_0 (somehow guessed), the session is established correctly and thereby $\mathcal{A}$ gets confirmation that π_0 is the password used by the chip,*

2. *the session results in a failure on the side of the chip, and always at the same moment – namely, the moment of receiving T_B. Furthermore, all messages received by $\mathcal{A}$ from the chip are distributed uniformly at random, independently of the password π_0 of the chip and the messages sent by $\mathcal{A}$.*

Proof. By the considerations preceding Theorem 3.5, with all but practically negligible probability there are only two situations possible:

1. $\mathcal{A}$ proceeds exactly as a terminal knowing π_0: applies the same s during the derivation of $\hat{g}$, and thereby can extract π_0 by finding π such that $\mathbf{Enc}_{K_\pi}(s) = z$,

2. the chip aborts after receiving T_B.

So, all we have to show is that in the second case $\mathcal{A}$ learns nothing about π_0.

First, consider a ciphertext z. Note that for each password π, there is exactly one s out of 2^l possibilities such that $z = \mathbf{Enc}(K_\pi, s)$, namely $s = \mathbf{Dec}(K_{\pi'}, z)$. Therefore, the probability to get z given password π is exactly $\frac{1}{2^l}$. As the probability is the same for each password, z is stochastically independent of the password used by the chip.

The next message sent by the chip is X_A. Recall that x_A is chosen at random independently of the other events during the protocol run, so X_A has the same probability $\frac{1}{q-1}$ for each password (recall that the group is cyclic with a prime order q and that we choose x_A uniformly at random from $\{1, \ldots, q-1\}$).

For Y_A, the situation is only slightly more complicated. The value of $\hat{g}$ computed by the chip depends on the password π_0 used by the chip (via the value of s corresponding to z) and on X_B sent by $\mathcal{A}$. However, the resulting element $\hat{g}$ is an element of a cyclic group and, therefore, it is a generator of this group. As the chip chooses y_A uniformly at random from $\{1, \ldots, q-1\}$, the resulting Y_A is again uniformly distributed in the set of group elements different from 1 and X_A.

In the interaction considered, the message T_A is not sent, as the chip rejects the session after receiving T_B. We may conclude that in this case all messages received by $\mathcal{A}$ are stochastically independent of π_0. □

Note 3.5.1. *In the above proof, there is a negligible probability of a false positive: T_B created for a wrong password π could be correct for π_0 and accepted by the chip. Then $\mathcal{A}$ will get a valid T_A from the chip and possibly gain some knowledge about π_0. $\mathcal{A}$ may realize at this moment that the password π is wrong (with overwhelming probability T_A is inconsistent with π unless $K'_{\mathbf{MAC}}$ is the same for π_0 and π). In Section 3.4.2, we shall investigate likelihood of this event more closely for the case where $\mathcal{A}$ has more capabilities to ensure that this situation may occur. For now, it suffices to say that this case occurs with a practically negligible probability.*

Of course, we cannot assume that the probability distribution of passwords from Π is always uniform from the point of view of the adversary. $\mathcal{A}$ may have some side-channel information about the password used. Moreover, the a priori distribution of the passwords may be different for each location and time and a particular implementation. This a priori knowledge might be available to the adversary. Thus, the real question is about difference of the following probability distributions:

- the probability distribution D resulting from the a priori knowledge,
- the probability distribution D' equal to the probability D conditioned by the protocol execution observed by $\mathcal{A}$.

Some differences between D and D' are inevitable: an adversary may guess a password, execute the protocol with the guessed password, and learn if the guess was correct. The key question is whether the adversary can learn more

from the execution of the protocol. We concern here not only changing the set of passwords with non-zero probability, but also changing the probabilities in a nontrivial way. For an incautious protocol design, an attacker would gain a lot if the probabilities in D' are significantly more skewed than in D.

Luckily, Theorem 3.5 says that apart from an event of a negligible probability, either $\mathcal{A}$ gets an implicit confirmation that the guessed password is correct, or the data gained by $\mathcal{A}$ (including the termination moment) are stochastically independent of the correct password and the only information leaked is that the guessed password is wrong. Therefore, if the guessed password π is wrong, the probability distribution D' is obtained from D as conditional probabilities, where the condition is that π is incorrect. So for any $\pi' \neq \pi$

$$\Pr_{D'}(\pi') = \frac{\Pr_D(\pi')}{1-\Pr_D(\pi)}.$$

3.4.2 Adversary interacting with a terminal

In this subsection, we will show that the adversary acting as a chip in the communication does not have any strategy significantly more advantageous than just a brute force approach in trying out password candidates, similarly to Section 3.4.1. We shall also show how many passwords can be checked in a single trial. In this setting, the adversary starts to interact with the terminal after the correct password π_0 is conveyed to the terminal.

This scenario is needed as a stepping stone for a more complicated scenario in the next subsection; however, let us remark that it corresponds also to a stand-alone attack. That is, a person holding a chip may enter the correct password into the terminal, hoping that the terminal will start an interaction with the chip. However, the adversary may transmit a much stronger signal and thereby replace the chip in communication with the terminal. The terminal will be unaware of the situation and will run the protocol.

Unlike in Section 3.4.1, the adversary $\mathcal{A}$ now has the freedom to choose z. Note that for each password $\pi \in \Pi$, the adversary $\mathcal{A}$ can compute $s_\pi(z) = \mathbf{Dec}(K_\pi, z)$. If we treat the mapping $\pi, z \to s_\pi(z)$ as a random function and z is chosen at random, then the probability of a collision should be small. However, the situation is slightly more complicated since $\mathcal{A}$ now has the freedom to choose one of the 2^l possible ciphertexts z. Therefore, from the point of view of $\mathcal{A}$, it might be beneficial to solve the following problem:

Problem 3.6. *For $m > 1$, find z such that there are m different passwords $\pi_1, \ldots, \pi_m$ in Π such that $s_{\pi_1}(z) = s_{\pi_2}(z) = \cdots = s_{\pi_m}(z)$.*

Note that given passwords π_1 and π_2, the probability that $s_{\pi_1}(z) = s_{\pi_2}(z)$ (that is, $\mathbf{Dec}(K_{\pi_1}, z) = \mathbf{Dec}(K_{\pi_2}, z)$) for a randomly chosen z is $\frac{1}{2^l}$, provided that we model the encryption function as a family of random keyed permutations. Therefore, the expected number of collisions for $s_{\pi_1}(z) = s_{\pi_2}(z)$ for a single z is $\frac{1}{2^l}$. By summing up over all ciphertexts z, we find that the expected number of ciphertexts z for which $s_{\pi_1}(z) = s_{\pi_2}(z)$ is equal to 1. Finding such z

is a different issue; however, the search can be done only once for all attacks of this kind.

Now, consider three different passwords π_1, π_2, π_3 and calculate the expected number of ciphertexts z such that $s_{\pi_1}(z) = s_{\pi_2}(z) = s_{\pi_3}(z)$. The probability that a single z yields such a solution is $\frac{1}{(2^l)^2}$. Thus, the expected number of ciphertexts z such that $s_{\pi_1}(z) = s_{\pi_2}(z) = s_{\pi_3}(z)$ equals $\frac{1}{2^l}$, which is a negligible value. Therefore, the expected number of triples where such a solution exists is $\binom{10^6}{3} \cdot \frac{1}{2^l}$. For realistic values of l the value $\frac{(10^6)^3}{2^l}$ is negligibly small. Therefore, we may assume as follows:

Assumption 3.7. *There are no solutions to Problem 3.6 with $m \geq 3$ for the set of passwords that can be used for PACE (CAM).*

By Assumption 3.7, we may conclude as follows:

Corollary 3.7.1. *An adversary $\mathcal{A}$ sending z is aware of at most two passwords π_1, π_2 such that $s_{\pi_1}(z) = s_{\pi_2}(z)$.*

It follows that $\mathcal{A}$ cannot test directly 3 or more passwords in one interaction with the terminal, while testing 2 passwords at a time is possible, but conditioned by an extremely large precomputation.

Security analysis of this scenario requires introducing an additional assumption that we will call Extended KEA1:

Assumption 3.8 (Extended KEA1)**.** *Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

Phase 1: $\mathcal{C}$ chooses ζ after performing some interactive protocol with $\mathcal{A}$.

Phase 2: $\mathcal{C}$ randomly chooses y, calculates $Y := \zeta^y$, and provides it to $\mathcal{A}$.

Phase 3: $\mathcal{A}$ presents a pair C, D such that $D = C^y$.

If $\mathcal{A}$ can present such a pair with a non-negligible advantage, then there is an extractor which can be used to return x and ζ such that $C = \zeta^x$ and $D = Y^x$ on the same input as $\mathcal{A}$.

Note 3.8.1. *Observe that Assumption 3.8 extends Assumption 2.5 (KEA1) by the fact that $\mathcal{A}$ is not provided with ζ, but rather infers it himself by participating in the protocol with $\mathcal{C}$.*

Assumption 3.8 captures the very fundamental security requirement for PACE and possibly was implicitly assumed by its designers.

The construction of PACE seems to rely heavily on unforgebility and pseudorandomness of the tags T_B and T_A. These properties can be derived in the Random Oracle Model, but generally hang on the concrete design for the tags.

Assumption 3.9. *(a) If given a valid tag T_B created by the terminal following the protocol, the adversary $\mathcal{A}$ can derive T_A for the same key $K'_{\mathbf{MAC}}$ with practically non-negligible probability p, then the adversary can extract K derived by the terminal.*

(b) If, with practically non-negligible advantage p, the adversary $\mathcal{A}$ can distinguish a random string 64-bit string from a valid tag T_B created by the terminal

following the protocol, then $\mathcal{A}$ can extract K such that $K'_{\mathbf{MAC}} = H(K\|3)$ and where $K'_{\mathbf{MAC}}$ is the key to be used to create T_B.

Let us observe that Assumption 3.9 is fulfilled in the Random Oracle Model. Indeed, for property (a) note that the value of T_A can be obtained by $\mathcal{A}$ by a call to the random oracle. For this query, the argument $K'_{\mathbf{MAC}}$ must be presented by $\mathcal{A}$ to the oracle. As $K'_{\mathbf{MAC}} = H(K\|3)$, except for a negligible probability, $K'_{\mathbf{MAC}}$ may appear in the calculations of $\mathcal{A}$ only if $\mathcal{A}$ queries the oracle for the value of $H(K\|3)$. Thus, $\mathcal{A}$ must be able to derive the argument K. The argument for part (b) is similar.

Now, let us resume analyzing a protocol execution performed by $\mathcal{A}$ and the terminal. In the setup discussed, $\mathcal{A}$ acts as the chip, so at some point $\mathcal{A}$ sends X_A to the terminal. Let us prove the following lemma needed for the subsequent analysis:

Lemma 3.9.1. *If $\mathcal{A}$ can establish a successful session, then in this case $\mathcal{A}$ must be able to recover the discrete logarithm of X_A except for a practically negligible probability.*

Lemma 3.9.1 confirms the intuition that the only chance to create a successful session is to know x_A. There might be different strategies to create X_A, but nevertheless they must lead to the same result: $\mathcal{A}$ must be able to derive x_A.

Proof. In this proof, we will build a simulation and show that if $\mathcal{A}$ can establish a successful session of PACE with the terminal (with a non-negligible probability), then $\mathcal{A}$ can also recover the discrete logarithm of X_A. First, let us define phase 0 of the simulation: having full control over the simulation, which means controlling the terminal and the adversary, we start an execution of the protocol up to the point where $\mathcal{A}$ creates X_A. Then we proceed to phase 1, still having full control, up to the moment when the chip should create T_A. In this phase, we assume that $\mathcal{A}$ can produce a valid tag T_A for $(Y_B, \mathcal{G})$.

Observe that at the moment of calculating T_A, the adversary $\mathcal{A}$ already received a valid T_B created by the terminal with the key $K'_{\mathbf{MAC}}$. Now, by Assumption 3.9(a), we counclude that $\mathcal{A}$ is in possession of the key K. This is a solution to an instance of the Computational Diffie-Hellman Problem, where $(\hat{g}, Y_A, Y_B)$ play the role of (g, g^a, g^b). The simulation is built in such a way that Y_B is randomly chosen on the side of the terminal.

Let us check the situation from the point of view of Assumption 3.8: $\mathcal{A}$ has a pair $(C, D) = (Y_A, K)$, where $Y_A = \hat{g}^{y_A}$ and $K = Y_B^{y_A}$ for some y_A. So according to Assumption 3.8, $\mathcal{A}$ in particular knows $\hat{g}$ calculated by the terminal.

Now, $\mathcal{A}$ can guess the password π and deduce s used by the terminal (recall that the number of possible passwords is limited so that $\mathcal{A}$ can guess correctly with a non-negligible probability). For s, the adversary $\mathcal{A}$ derives $h = \hat{g}/g^s$. So, $\mathcal{A}$ gets the solution to the Diffie-Hellman Problem for the tuple (g, X_A, X_B) with a nonnegligible probability. By Assumption 2.5, this is only possible if $\mathcal{A}$ can extract x_A. □

One can get a slightly stronger result with the same proof but using Assumption 3.9(b) instead of Assumption 3.9(a).

Lemma 3.9.2. *If $\mathcal{A}$ can distinguish between T_B created correctly by the terminal and a random 64-bit string, then $\mathcal{A}$ must be able to recover the discrete logarithm of X_A except for a practically negligible probability.*

Although Lemma 3.9.1 says that $\mathcal{A}$ eventually knows x_A, it does not yet say when $\mathcal{A}$ learns x_A for the first time. Let us take a closer look at this question. Definitely, as shown in the proof of Lemma 3.9.1, $\mathcal{A}$ has to know x_A after receiving T_B.

According to the random oracle model, the oracle may choose the value for $K'_{\mathbf{MAC}}$, and thus for T_B, in advance and before the terminal presents a query for $H(K||3)$ and before $\mathcal{A}$ presents Y_A. The choice of $K'_{\mathbf{MAC}}$ is random. Hence, $\mathcal{A}$ can start the procedure of retreiving x_A already after receiving Y_B by choosing $K'_{\mathbf{MAC}}$ at random and passing this value into the oracle for H. In this scenario, Y_A can be disregarded, as it is determined by $\mathcal{A}$.

Now, recall that the probability distribution of Y_B is uniform (except for X_B, which is forbidden). Thus, $\mathcal{A}$ can start the procedure to retrieve x_A not after receiving Y_B, but before – the input Y_B can be simulated by $\mathcal{A}$ by a random choice.

We conclude that $\mathcal{A}$ must be able to find x_A immediately after sending X_A. This means that the strategy of $\mathcal{A}$ is to send X_A for which $\mathcal{A}$ can determine its discrete logarithm. This leads to the following conclusion.

Corollary 3.9.1. *For the considered strategies of $\mathcal{A}$, we may assume that $\mathcal{A}$ knows x_A.*

Now let us turn our attention to the main result in this subsection:

Theorem 3.10. *If $\mathcal{A}$ has a non-negligible probability in succeeding in establishing a session with some passwords π_1, π_2, giving different decryptions of z, he can solve Discrete Logarithm Problem for X_B.*

Proof. Let us create another simulation of the protocol between $\mathcal{A}$ and the terminal. This time the terminal does not choose x_B at random. Instead, we take $X_B = A^{\kappa}$, where κ is chosen at random, and A is an instance of the Discrete Logarithm Problem we want to solve using $\mathcal{A}$. Of course, it suffices to find the discrete logarithm of X_B.

According to Corollary 3.9.1, $\mathcal{A}$ can extract x_A at the time of creation of X_A. As the terminal cannot compute h itself, the value of h is passed to the terminal as $X_B^{x_A}$. This is possible, since we control the simulation and have access to the memory of $\mathcal{A}$. Moreover, note that for each password π_i, the adversary $\mathcal{A}$ can calculate $\hat{g}(\pi_i) = h \cdot g^{s_i}$, where $s_i = \mathbf{Dec}(K_{\pi_i}, z)$ and $K_{\pi_i} = H(\pi_i||0)$.

Now, assume that $\pi_i \neq \pi_j$, $s_i \neq s_j$ and that $\mathcal{A}$ can create valid sessions with the terminal using π_i and π_j, when the initial messages exchanged are z, X_B, X_A, Y_B, Y_A. This means that $\mathcal{A}$ can create a valid tag T_A in both cases –the tag corresponding to π_i and the tag corresponding to π_j – in response

to T_B obtained from the terminal. By Assumption 3.9, in either case $\mathcal{A}$ must be able to obtain the key K (which depends on the password). In turn, by Assumption 3.8, it means that $\mathcal{A}$ can derive $y_{A,i}$ and $y_{A,j}$ such that

$$Y_A = (h \cdot g^{s_i})^{y_{A,i}} \quad \text{and} \quad Y_A = (h \cdot g^{s_j})^{y_{A,j}}$$

Hence,

$$h^{y_{A,j}-y_{A,i}} = g^{s_i \cdot y_{A,i} - s_j \cdot y_{A,j}}$$

Note that $y_{A,j} \neq y_{A,i}$ (because those are discrete logarithms of Y_A for different generators), we have:

$$h = g^{(s_i \cdot y_{A,i} - s_j \cdot y_{A,j})/(y_{A,j}-y_{A,i})}$$

As $h = X_B^{x_A}$, we conclude that the discrete logarithm of X_B is equal to

$$\frac{(s_i \cdot y_{A,i} - s_j \cdot y_{A,j})}{(y_{A,j} - y_{A,i}) \cdot x_A} \bmod q$$

□

Theorem 3.10 shows that $\mathcal{A}$ acting as a chip and aiming to get a successful session has no option but to follow the protocol with a guessed password. If the guess is wrong, then the only gain will be to learn that the guess was wrong. The only optimization might be to test two passwords at once (see Corollary 3.7.1) exploiting the fact that a carefully chosen z may decrypt to the same s with these passwords.

Still, $\mathcal{A}$ may attempt to learn something about the password used by the terminal without actually hoping to establish a session. The point is that T_B may carry some information about the password. Recall that this is the first (and only) element sent by the terminal that depends stochastically on the password.

Theorem 3.11. *If $\mathcal{A}$ can distinguish between T_B created by the terminal and a random 64-bit string, then $\mathcal{A}$ must have followed the protocol with the same password as the one used by the terminal.*

Theorem 3.11 says in particular that $\mathcal{A}$ cannot learn anything about the password used by the terminal, unless $\mathcal{A}$ has guessed this password in advance and executed the protocol according to the specification with the guessed password.

Proof. Since the probability to distinguish between random strings and T_B is practically non-negligible, for a non-negligible fraction of tuples:

$$(z, X_A, X_B, Y_A, Y_B)$$

the adversary $\mathcal{A}$ can distinguish between valid T_B and random strings. As X_B, Y_B are independent of the password, in a practically non-negligible number

of cases $\mathcal{A}$ may distinguish between random strings and T_B created for a practically non-negligible number of passwords. If we select two passwords π_1, π_2 from this set, then with high probability (see Assumption 3.7) $s_1 = \mathbf{Dec}(K_{\pi_1}, z) \neq \mathbf{Dec}(K_{\pi_2}, z) = s_2$. In this case, the keys K corresponding to π_1 and π_2 are different.

By Lemma 3.9.2, we may assume that $\mathcal{A}$ knows x_A and thus can compute the same h as the terminal. On the other hand, by Assumption 3.9(b), $\mathcal{A}$ must be able to compute K corresponding to π_1 and π_2. By Assumption 2.5, computing $K = K(\pi_i)$ implies that $\mathcal{A}$ can compute $y_{A,i}$ such that $Y_A = (h \cdot g^{s_i})^{Y_{A,i}}$.

Now we can proceed as in the proof of Theorem 3.10: we build a simulation that enables to solve the Discrete Logarithm Problem for X_B.

□

The above proofs transfers smoothly to PACE CAM, as in PACE CAM the adversary acting as the chip in the communication also finds out if the password is right at the moment of exchanging the tags. It remains only to show that in this case $\mathcal{A}$ the session run by $\mathcal{A}$ and the terminal will finally fail due to a wrong last message sent to the terminal.

Theorem 3.12. *Assume that adversary $\mathcal{A}$ acting as a chip in PACE CAM interacts with the terminal using password π, where $\mathcal{A}$ knows π but not the private key z_A. If the strategy of $\mathcal{A}$ enables him to establish a session with the terminal, then the Discrete Logarithm Problem can be solved.*

Proof. As observed above in Lemma 3.9.1, $\mathcal{A}$ has to know x_A in order to create a valid tag T_A. Moreover, the session would be terminated prematurely in case of an invalid tag. At the last step, $\mathcal{A}$ has to provide a ciphertext of $x_A/z_A \bmod q$.

In order to solve the Discrete Logarithm Problem, on instance Z, we take $Z_A = Z^\kappa$ for a random κ. We choose arbitrary password π, and run PACE CAM for $\mathcal{A}$ executing the discussed strategy and impersonating the owner of Z_A. We mimic the terminal using the password π and finally examine the memory of $\mathcal{A}$ to find the value of x_A, and the memory of the terminal to find the value of $x_A/z_A \bmod q$. This enables us to derive z_A and $z_A/\kappa \bmod q$ which is equal to the discrete logarithm of Z. □

3.4.3 Adversary interacting with the terminal and the chip

Finally, we have to consider the most general case in which the adversary interacts with both a chip and a terminal using the same password. Namely, we consider the following scenario:

Scenario 3.13. *In this scenario, the adversary $\mathcal{A}$ interacts with the chip $\mathcal{C}$ and a terminal $\mathcal{T}$, there is no direct communication between $\mathcal{C}$ and $\mathcal{T}$, the chip and the terminal start the protocol with the same password π_0, while $\mathcal{A}$ has no information about the password used by the chip and the terminal.*

Scenario 3.13 may occur in practice, when a legitimate user enters the correct password π_0 to the terminal (thus, both the chip and the terminal use the same

password). At the same time, the adversary may control communication over the radio channel as a man-in-the-middle, simply by sending stronger signals and blocking any direct communication.

We already know (see Theorem 3.3) that the adversary cannot learn the session key if the connection between the chip and the terminal is successfully established. However, the aim of the attack may be limited: instead of breaking into the channel established by the chip and the terminal, $\mathcal{A}$ may attempt to learn the password π_0 or impersonate the chip or the terminal knowing the password. The latter option, called *session hijacking*, will be discussed in Sect. 3.4.4.

For the interaction between the chip and $\mathcal{A}$ let us use the same notation as in the protocol description, while for the interaction between $\mathcal{A}$ and the terminal, we mark the variables with "(r)", e.g. we use $X_A^{(r)}$ instead of X_A.

First, we may assume that the messages appear in the following order:

$$z, z^{(r)}, X_B^r, X_B, X_A, X_A^{(r)}, Y_B^{(r)}, Y_B, Y_A, Y_A^{(r)}, T_B^{(r)}, T_B, T_A, T_A^{(r)}$$

(with the exception that the chip can break up the communication before sending T_A). In fact, any other order of execution can be converted to the above schedule by postponing the messages sent by $\mathcal{A}$. This works to the advantage of the adversary, since $\mathcal{A}$ has more data at hand when sending the next message.

$T_B^{(r)}$ is the first message received by $\mathcal{A}$ that is not stochastically independent of the password π_0 used by the chip and the terminal. At this moment, $\mathcal{A}$ has to send the message T_B that depends on π_0. Note that at this moment of execution both the terminal and the chip have already calculated the key K. By Theorem 3.3, there are only two cases:

case 1: either the chip and the terminal share the same key K and this key cannot be derived by $\mathcal{A}$, or

case 2: the terminal and the chip calculate different keys K.

Case 1: In this situation, we may concern $\mathcal{A}$ as man-in-the-middle between the chip and the terminal manipulating their communication, but attempting to establish a session with the same master key K. In Theorem 3.2, we have shown that the session will be successful from the point of view of both the terminal and the chip, if and only if $\mathcal{A}$ does not manipulate any message. The only exception is to raise X_A and X_B to the same power in the case of PACE. As the adversary does not know the key K used to derive $K'_{\mathbf{MAC}}$, the tags $T_B^{(r)}$ and T_A created by the terminal and the chip correspond to random oracle queries, where the adversary does not know the arguments. Consequently, $\mathcal{A}$ does not gain any information about π_0.

Now, we have to consider the situations where the session between the tag and the terminal is not successful. The reason for that would be an invalid tag delivered either to the chip or to the terminal. Potentially, this scenario may create an opportunity for $\mathcal{A}$ to learn something about π_0. Definitely, as the adversary does not know the key shared by the chip and the terminal, any attempt to create a valid tag by $\mathcal{A}$ will fail with all but practically negligible probability.

So, the only situation that remains to be considered is forwarding the tags $T_B^{(r)}$ and T_A created, respectively, by the terminal and the chip. However, in this case the situation is again oblivious: except for a practically negligible probability, $T_B^{(r)}$ will be accepted by the chip if and only if $Y_A = Y_A^{(r)}$. Similarly, T_A will be accepted if and only if $Y_B = Y_B^{(r)}$. Therefore, the outcome does not depend on the password used.

Case 2: In this case, T_B^r is created with a different key than the key used for verification of T_B. Therefore, for all but a practically negligible probability, $\mathcal{A}$ has to create T_B presented to the chip.

Note that by Theorem 3.11, the tag T_B^r can be distinguished from a random 64-bit string only if $\mathcal{A}$ follows the protocol with correctly guessed password π_0. Thus, the whole interaction between the terminal and $\mathcal{A}$ can be simulated by $\mathcal{A}$. One case is when $\mathcal{A}$ does not follow the protocol with π_0 – then the tag T_B^r can be simulated by a random 64-bit string. The second case is when π_0 is used by $\mathcal{A}$ following the protocol specification – now $\mathcal{A}$ can simulate the responses of the terminal. The only information that is not known in advance is the password. If $\mathcal{A}$ correctly guesses π_0 and uses the guess for the interaction with the chip, it will get confirmation of the right guess. Otherwise, it will know that the guess was wrong.

Note that the messages sent by the chip until the moment after receiving T_B may correspond to any password used. A strategy of $\mathcal{A}$ that would succeed to create valid T_B in practically non-negligible number of cases must ensure that the correct T_B will be created for a non-negligible number of passwords. However, recall that according to Assumption 3.7, each value of s corresponds to at most two different passwords from this set. Therefore, we conclude that there are 2 passwords π, π', for which $\mathcal{A}$ can create valid T_B. However, this contradicts Theorem 3.5. Therefore, the only option to create a valid T_B was to guess π_0 and follow the protocol with password π_0 in the interaction with the chip. In this case, an interaction with the terminal may only confirm at an early moment that the guess was correct. The answer T_A of the chip brings only one information: it confirms the correctness of the guess of the password, while the value T_A can be calculated by $\mathcal{A}$ himself.

Let us now check what can happen in the interaction between $\mathcal{A}$ and the terminal. If $\mathcal{A}$ has followed the protocol for the correctly guessed password π_0, then of course $\mathcal{A}$ knows the correct key to create $T_A^{(r)}$, independently of what has happened in the interaction with the chip. The question is whether this is the only case where $T_A^{(r)}$ will be accepted by the chip.

Let us assume that the strategy of $\mathcal{A}$ enables him to provide $T_A^{(r)}$ that would be accepted by the terminal for non-negligibly many passwords. As we have seen, apart from a practically negligible number of cases, the interaction with the chip is interrupted after sending T_B, or $\mathcal{A}$ has followed the protocol with correctly guessed π_0. In the second case, the interaction can be simulated by $\mathcal{A}$ (even if PACE CAM s concerned). Therefore, the only advantage for $\mathcal{A}$ is, possibly, confirmation of the password used by the chip. So, at this moment,

we can use Theorem 3.10 and conclude that the assumption was false.

Note that if PACE CAM is executed, then, for case 2, in the last step $\mathcal{A}$ will fail. Indeed, as we have seen, in this case the interaction between the chip and the terminal can be fully simulated. So the success of $\mathcal{A}$ in this case would contradict Theorem 3.12.

Corollary 3.13.1. *After analyzing all the cases, we may conclude that interacting with both the chip and the terminal at the same time brings no additional advantage for the adversary. Except for a practically negligible number of events either*

- *the interactions with the chip and the terminal are independent, or*
- *$\mathcal{A}$ simply acts as passive man-in-the-middle and learns nothing abut the password, or*
- *the protocol is aborted by either the chip or the terminal in a oblivious way, that is, at the moments that do not depend on the password used by the terminal and the chip.*

3.4.4 Session hijacking

By Corollary 3.3.1 we already know that the session cannot be established so that the chip, the terminal, and an adversary $\mathcal{A}$ acting as a man-in-the-middle hold the same session key. This does not automatically mean that it is impossible to establish a session between $\mathcal{A}$ and the chip, or between $\mathcal{A}$ and the terminal, or even that $\mathcal{A}$ gets two separate connections: one with the chip and one with the terminal. In the case of PACE, the last situation is possible if $\mathcal{A}$ holds the password shared by the chip and the terminal (or guesses this password correctly). In principle, we may fear that $\mathcal{A}$ takes advantage of the fact that both the chip and the terminal use the same password, and that at the right moment (when the password is already checked), $\mathcal{A}$ takes over the communication and succeeds in connecting regardless of the lack of the password.

Session hijacking might be beneficial for many malicious purposes. For example, when a chip starts a session with a legitimate terminal, the adversary may try to hijack the session and learn the data that the chip reveals to the terminal, believing that this is the terminal authenticated by the password. Similarly, $\mathcal{A}$ may attempt to provide false information to the terminal, allegedly originating from the chip.

Fortunately, in Sect. 3.4.3 we have shown that $\mathcal{A}$ can successfully create a session with either the chip or the terminal or both, if and only if $\mathcal{A}$ has guessed the correct password before receiving any message dependent on the password shared by the chip and the terminal. Thereby, session hijacking does not make any practical sense: the adversary may try his luck directly by guessing the password and interacting either with the chip or with the terminal.

In the case of PACE CAM, we have observed even more. While $\mathcal{A}$ can establish a session with the chip (assuming that $\mathcal{A}$ uses the correct password),

$\mathcal{A}$ will fail to send the correct message in the last step of PACE CAM and the session will be aborted by the terminal. The point is that the authentication of the chip based on the secret key is not transferrable.

3.5 Privacy

3.5.1 User tracking

An adversary may initiate wireless communication with a chip in their vicinity, attempting to learn the identity of the chip. There might be a whole network of malicious terminals interacting with the chips. As wireless communication may be initiated by the terminal, chip holders may be unaware of the situation. In fact, this is one of the main concerns frequently expressed by opponents of electronic identity documents.

Although entering the password to the terminal is regarded as a consent of the chip holder, it is likely that the network of malicious terminals uses passwords presented during legitimate interactions and retained (illegally). This is a problem that occurs for any system based on passwords, where the terminal is authorized only by knowledge of the password. (Let us note that this problem does not appear in the version of PACE we have presented in [15].)

In the case of PACE, the password is the only identifying information available, as long as the chip does not transfer any identity-related information after establishing the session with PACE. In the case of PACE CAM, the identifying information is not only the password, but also the public key. Unlike for the password alone, the identity based on the public key uniquely defines the chip holder.

If an adversary $\mathcal{A}$ controlling a network of terminals does not know the password of the tracked person, then according to the results of Section 3.4.1 the chip will abort the session unless the terminal decrypts the ciphertext z as intended by the chip. According to Claim 3.4.1, with all but practically negligible probability, this happens only if $\mathcal{A}$ uses the correct password.

According to the results of Section 3.4.3, a similar situation occurs when the adversary acts as an active man-in-the-middle. The only difference is that the adversary can probe at the same time the chip and the terminal holding the password of the chip. As at most two passwords can be tested when interacting with the terminal (see Section 3.4.2) at most three passwords can be tested at a time. Note that tracking attacks of this kind are not purely theoretical and could be similar to skimming ATM chips on ATM machines.

Given the large number of passwords, it seems that the described threats are limited to the case when the adversary has substantial a priori knowledge on who may appear at a given location, or the adversary is focused on tracking a small number of chips. Obviously, some false positives are possible, if the passwords are short enough compared to the number of users. In the case of CAN numbers used in Germany (Zugangsnummer) consisting of 6 digits, given the size of the population, the average number of personal ID cards with the

same CAN in the country is probably less than 100 (taking into account ID cards issued to non-citizens). Unless the CAN numbers are issued in a heavy non-uniform way, the probability of false positives in tracking is quite limited.

Note that PACE CAM does not contribute any protection against user tracking. This is due to two factors: first, the error messages are generated immediately after detecting an invalid tag. Second, the terminals are not authenticated, so they can freely probe the chips.

3.5.2 Proof of presence

One of the major issues to be considered for authentication protocols is whether a terminal can create a proof of interaction with a chip after executing the protocol with this chip. The proof may be composed of any values known to the terminal – not only the messages exchanged, but also private ones. The session concerned need not to succeed – in some cases also a session aborted by one of the protocol participants might be enough to prove the chip's participation.

Such a proof might be an explicit and useful functionality: a good application area is creating cryptographic evidence to prove that a patient has visited a healthcare unit. This can be useful to reduce the number of cases where a healthcare unit charges an insurance company for fake services. Let us note that such a proof-of-presence can be created on top of PACE [16] as will be discussed in Section 4.3.

In most cases, creating a proof-of-presence is not a legitimate goal of an interaction with the chip. If, nevertheless, creating a proof-of-presence is technically possible, then a serious security problem emerges. For example, according to GDPR [22] the entity that runs the terminals must guarantee that there will be no cases of misuse of personal data based on the proofs-of-presence. In general, the declarations are insufficient, as the system must be secure-by-design. Consequently, a pragmatic solution might be to backtrack and redesign the protocol in order to eliminate the threat of a proof-of-presence.

One of the fundamental properties of PACE and PACE CAM considered by their designers was *simulability*:

Definition 3.3. *A protocol transcript from the point of view of a protocol participant X is a list of all messages exchanged during a protocol execution together with all ephemeral values created by X during this execution.*

A protocol S is simulatable by participant X if for any strategy $\mathcal{A}$ (which may deviate from the protocol specification), participant X can generate protocol transcripts of X that are indistinguishable from transcripts generated during real protocol executions where X follows the strategy $\mathcal{A}$.

In the above definition, we can talk about computational indistinguishability or information-theoretic indistinguishability. In this case, we use information-theoretic indistinguishability, as the simulated transcripts have the same probability distribution.

Let us check the situation from the point of view of the terminal. First, note that the messages z, X_A, Y_A are independent of the password and uniformly

distributed. By Theorem 3.5, there are two outcomes of the protocol execution: either messages X_B, Y_B and T_B correspond to the right password and the terminal knows x_B and y_B, or the protocol is interrupted by the chip after receiving an invalid T_B. Both options are easy to simulate. In the first case, the transcript contains the values created by the terminal during protocol execution. For the values allegedly generated by the chip:

- s is created at random, then z is calculated as $\mathbf{Enc}(K_\pi, s)$,
- X_A is calculated as Z_A^σ for σ chosen at random,
- Y_A is chosen at random,
- T_A is calculated according to the protocol specification (recall that the terminal uses the same key $K'_{\mathbf{MAC}}$),
- in the case of PACE CAM, the last message C is calculated as $\mathbf{Enc}(K'_{\mathbf{Enc}}, (\sigma, \mathrm{cert}(Z_A))$ where the key $K'_{\mathbf{Enc}}$ is determined as on the terminal's side.

It is easy to see that such simulated responses of the chip have exactly the same probability distribution as in the case of a real computation.

The second case is even easier. The messages z, X_A, Y_A can be generated at random, and an error message appears instead of T_A.

Note that the above simulations can be performed by any party. For this purpose, it suffices to mimic the terminal. In particular, it can be a third party or a chip (regardless whether it holds z_A or not). Finally, we can conclude as follows:

Theorem 3.14. *PACE and PACE CAM are simultable by the chip, by the terminal, and by an external observer.*

3.6 Chip Authentication with CAM

While in the previous section we have shown that the password protection provided by PACE is relatively strong, it does not help in the situation when an adversary knows the password corresponding to a given chip. As argued in Sect. 3.5.1, it is inevitable that an adversary can collect the passwords of the attacked persons and then use these passwords for impersonation. Therefore, PACE alone might be regarded only as a very weak authentication of a chip. The design goal of PACE CAM was to provide a reliable chip authentication integrated with a password-based authenticated key exchange.

Although intuitively the mechanism introduced in PACE seems to be secure, and previous papers have provided arguments for that, we investigate the protocol taking into account the full spectrum of active attacks. Fortunately, given the results of the previous sections, the following theorem is a straightforward corollary.

Theorem 3.15. *If PACE CAM protocol terminates successfully, then the chip knows the secret key z_A.*

The crucial observation for analyzing the situation is the observation formulated in Corollary 3.9.1. Namely, we can assume that if the terminal does not abort a session after exchanging the tags, then the chip knows x_A, the discrete logarithm of X_A. As the terminal can test that σ equals x_A/z_A by checking that $Z_A^\sigma = X_A$, the terminal may conclude that its interlocutor knows both x_A and x_A/z_A. It means that the chip may extract z_A as well. In this way, the proof-of-possession of the secret z_A is completed.

Another issue to be concerned is whether executing PACE CAM by the terminal leaks information about z_A so that an attacker can derive it. However, a simple argument based on simultability leads to the following theorem:

Theorem 3.16. *If an attacker $\mathcal{A}$ aiming to break public key Z_A interacts with the chip running PACE CAM using the key z_A, then the same result can be obtained by $\mathcal{A}$ without interacting with the chip.*

Proof. The result follows directly from Theorem 3.14: instead of running an interaction with the chip, it is simulated. So, the attack can be performed offline by $\mathcal{A}$. □

Corollary 3.16.1. *Impersonating the chip in the PACE CAM protocol is equivalent to breaking the Discrete Logarithm Problem.*

Proof. By Theorem 3.12, an adversary $\mathcal{A}$ that succeeds to authenticate itself for public key Z_A, can extract the private key z_A. On the other hand, by Theorem 3.16 any prior interaction can be simulated by $\mathcal{A}$. Therefore, $\mathcal{A}$ can be used to break the Discrete Logarithm Problem for Z_A. □

Chapter 4

Proposed extensions of PACE

In this chapter, guidelines for extending a protocol are presented as well as some previous work. Security analysis of the proposed extensions contains mostly arguments reducing the security of the discussed schemes to the security of PACE or PACE CAM, providing a clear path to reductions, using the results from the previous chapter. The main focus of this chapter is to present possible functionalities that can be achieved and to understand the general requirements behind them. All the extensions are based on version of PACE with General Mapping, so suffix "GM" will be omitted in this section.

4.1 Guidelines for extending a protocol

In this section, strategies are discussed on how to make a modification to a protocol to increase its chances of acceptance by researchers and industry. A general "golden rule" would be to keep the necessary software changes to a minimum from the technological perspective and to reuse the security analysis for the new protocol as much as possible so that the introduction of a new security analysis can be facilitated. Some of the most beneficial rules that should be followed are the following:

- **Backwards compatibility**: The design of the modified protocol should allow to quickly switch to the original version when one of the parties participating in the protocol notices that the other one does not support the extended version. In terms of PACE, it means that connection should be established even if the chip or terminal runs the plain PACE if the extra functionality offered by the extended protocol is not crucial for the specific application.
- **Minimal changes**: For widely recognized protocols, their core functionalities are often enclosed in well-tested libraries. For that reason, the original protocol should be fine-tuned in such a way that the effort to build, test, and incorporate new functionalities into existing solutions is

minimal. This also means that the order and general structure of the messages should be maintained if possible, with new steps coming at the end and keeping the number of new messages to a minimum, which reduces the probability of a failure and promotes robustness. There will be examples presented in which a new functionality could be obtained without changes to the number of messages from the original protocol.

- **Reusing of code and cryptographic operations:** Usually in the interactive protocols the cryptographic operations are the parts of the program that take the most time to compute and are most complicated, like secure exponentiation. For that reason, the code and time-consuming cryptographic operations should be reused when possible. Chips that execute PACE are usually low-power devices with limited memory and computational power, so the executable code must be deeply optimized.

- **Security arguments:** Arguably, this is the most important principle in designing modifications from the point of view of researchers. The security features should not be compromised by the modifications introduced; rather, the opposite should be true. Design of the new functionalities should take into account the existing security analysis to facilitate future security arguments. Preferably, security arguments for the original scheme and for the extension should be composed to provide security arguments for the extended scheme. Unfortunately, this requires very careful design of both the original scheme and of the extension and in many cases it is not possible.

The presented extensions are intended to follow as much of these rules as possible. While discussing the designs, comments will be provided on the rules followed.

4.2 PACE Mutual Authentication

The extension discussed below was presented at ESORICS'21 conference and was published there ([15]).

4.2.1 Description of PACE MA

PACE MA is in a way an extension of the functionality of PACE CAM. The acronym MA stands for Mutual Authentication. In this extension, not only the chip authenticates itself but also the terminal. There are many scenarios where such functionality might be critical. Modern financial and healthcare systems exchanging sensitive data with a user, are good examples: a chip should have firm guarantees that sensitive data from it are not transmitted to unauthorized parties. In general, all scenarios where there is a need for authentication of both sides before sending sensitive data over a secure channel could potentially benefit from this extension.

There are proposed two versions of PACE MA – PACE MA and PACE MA-light. The second one achieves the same extended functionality as the first one, however they represent different approaches in design - the first one adds messages at the end of the communication between the chip and the terminal, and the other one follows the minimal approach (in terms of communication) of preserving the number of messages from the original PACE (hence the suffix "light"). Both of them realize the same functionality beside establishing secure channel – strong authentication of both sides of the protocol by means of introducing public key cryptography authentication via static Diffie-Hellman authentication.

Setting

In this extension, both sides use a pair of public and private keys in the group $\mathcal{G}$, where for the chip we denote them $(x, X = g^x)$ and for the terminal $(y, Y = g^y)$. There are certificates created for both public keys by a Certificate Authority (CA) that the other side must be able to verify. Protocol participants need to also agree on the hash functions and MAC algorithm used in this instance of the protocol. As this protocol is not part of a standard, there are many more options available, in particular one could choose to use longer tags, which lowers the probability of forging them[1]. The fact that AES-CMAC is used in PACE only in 64-bit option could raise concerns. Here those concerns could be alleviated.

Changes - PACE MA

In PACE MA (see Figure 4.1) the following changes to the original protocol were introduced:

- Messages from the first Diffie-Hellman exchange X_A and X_B are reused for the static Diffie-Hellman authentication protocol, where the counterparty of the protocol raises the received group element to the power of its private key, for example, the chip computes $K_A := X_B^x$. Later, using the public key of the chip and its own ephemeral value x_B, the terminal can verify if the exponentiation was done with the secret key x. An analogous (symmetric) operation is performed on the side of the terminal.
- The token K_A is hashed: $h_A := H(K_A)$ mainly to save memory, but also to make algebraic attacks more difficult for the adversary in case the token h_A is revealed. The same operation is performed on the side of the terminal with K_B.
- h_A and h_B are encrypted with the relevant certificates of X and Y using a key K'_{SC} and sent to the other participant as C_A and C_B. This also protects the identities of the parties participating in the protocol, as the

[1]That is, assuming a random oracle model for tags. In general, choosing an algorithm that produces longer tags does not automatically guarantee a higher level of security.

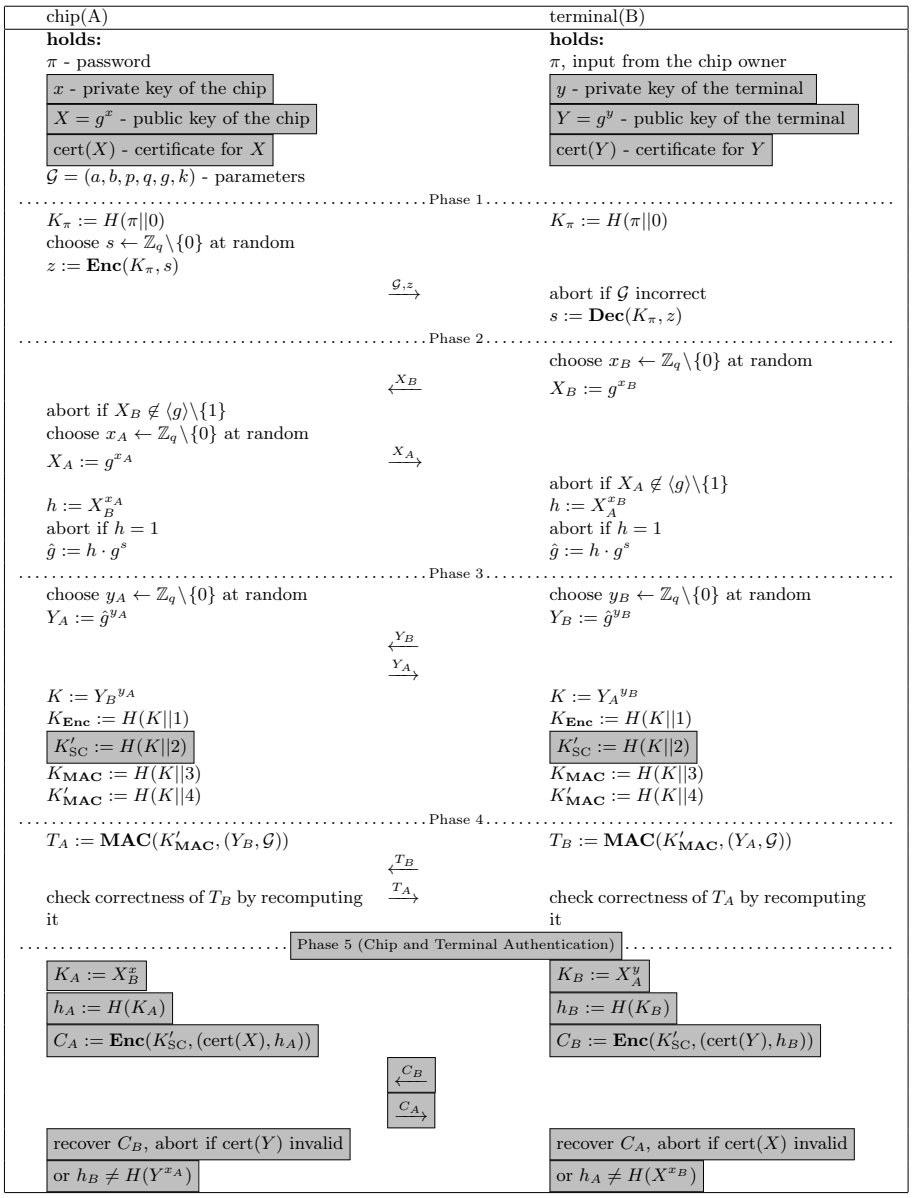

Figure 4.1: PACE MA – an extension of PACE with mutual authentication. H denotes a hash function, with its range given by the context. Gray boxes indicate the changes specific to PACE MA, the rest of the protocol is the original PACE.

authentication tokens are encrypted and a transcript of the protocol itself would not be sufficient to reveal the identities.

- Upon receiving C_A and C_B, the chip and terminal decrypt them with the

key K'_{SC} known only to them (as it follows from the execution of the protocol), verify the validity of the certificates and recompute h_A and h_B and check if they are correct using their x_A and x_B. If the equalities are satisfied, they complete the protocol successfully and can proceed to the subsequent communication with secure keys established.

The order of exchanging the messages (first C_B and then C_A) is not accidental. It follows from the order once preferred by the German information security authority BSI. It is the terminal that should present its identity first. Then the chip can make a decision if it wants to disclose its identity, too. The reason is that the identity of the chip usually falls to the category of personal data of the chip owner and the identity of the terminal not, as a part of infrastructure. Therefore, this approach is a good example of following the rules from the GDPR regulation [22]. Nevertheless, for specific applications, where it is the chip that needs to authenticate first, this order could be reversed.

Changes - PACE MA-light

PACE MA-light preserves the number of messages from PACE. It is done by merging two phases into one: exchange of the tags T_A and T_B and the authentication. For reference, look at Figure 4.2. The last phase of PACE CAM is also added for easier comparison between authentication mechanisms. The detailed changes are as follows:

- Exchange of tags T_A and T_B is now merged with authentication. In addition to ensuring that the session key K was correctly derived by including Y_A and Y_B on the tags, the authentication tokens K_A and K_B are also included.
- Here static Diffie-Hellman authentication is also used, like in PACE MA. However, instead of comparing hash values of recomputed authentication tokens, MACs are compared, verifying both the session key and the identity of the other party at the same time The tokens need to be recomputed on each side proving that the verifying party is in the possession of ephemeral secret (respectively, x_A or x_B).
- Like in the case of PACE MA, the identities of the participants are protected by encrypting the certificates.
- Protocol is completed successfully when no participant aborts the session during the verification process.

Observe that the identity information is sent after establishing the session key and using that session key for encryption. Note that the chip has an option to abort the session after learning the identity of the terminal.

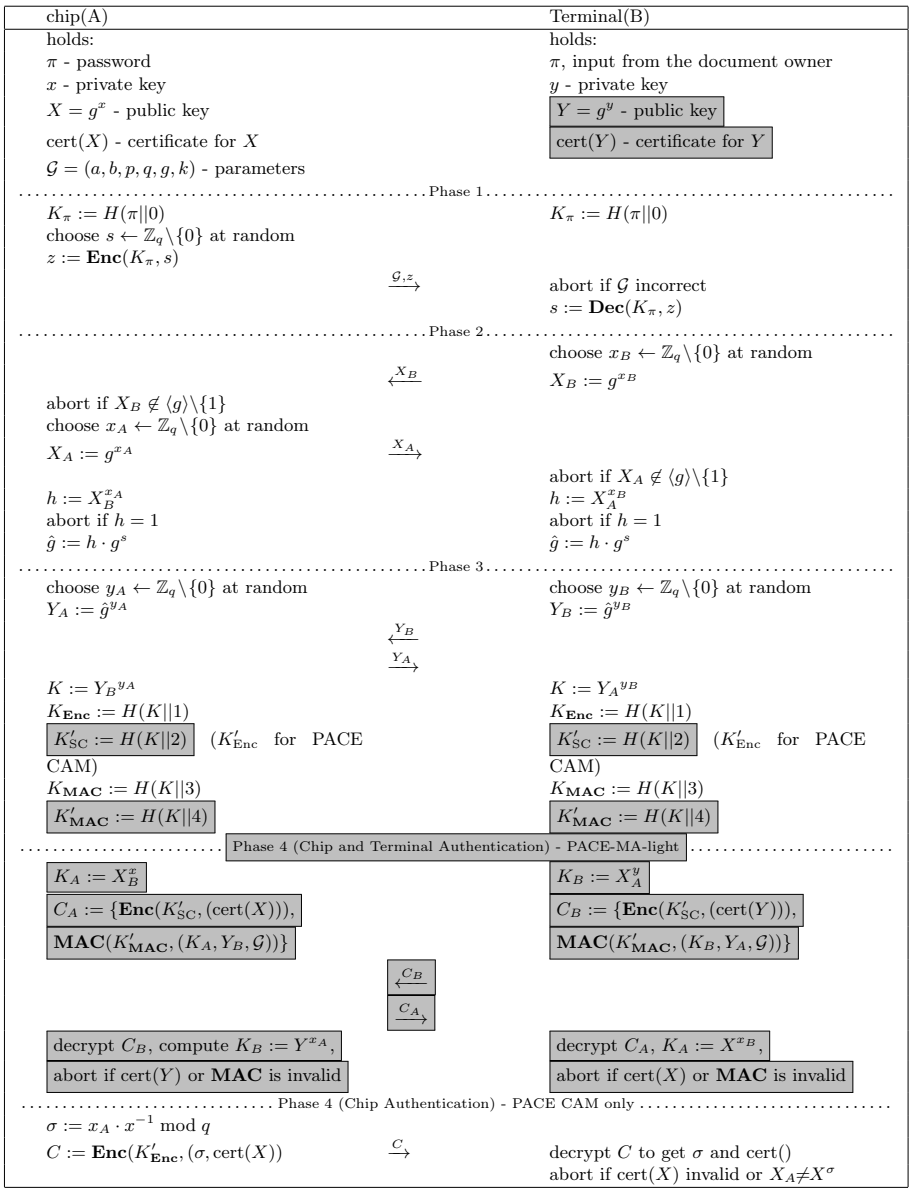

Figure 4.2: PACE MA-light – an extension of PACE with mutual authentication with the same number of messages as for the plain PACE. PACE MA-light changes to PACE CAM are in gray boxes, and additionally the last phases for PACE MA-light and PACE CAM are separated.

4.2.2 Discussion on the security and privacy

In this subsection, the security analysis of PACE (CAM) will be heavily reused and coupled with additional security arguments to discuss the features of the

presented extensions. Here are recalled crucial security properties from the work presented at ESORICS conference. These seem to be the most fundamental requirements, however there is always a potential to explore more application-specific scenarios. For the purpose of presentation of the main potential security issues of PACE MA(-light) protocols, the following discussion should be sufficient.

Fragility

The property that simplifies the security analysis for a protocol is *fragility* (as defined in Section 3.3.1).

Claim 4.0.1. *PACE MA and PACE MA-light are fragile.*

For PACE MA: as the first phases are the same as in plain PACE, the same conditions for aborting the session apply. That leaves $\mathcal{A}$ only with an option to raise X_A and X_B to the same power $\alpha \neq 1$. Consider now the message C_B and let $\mathcal{A}$ have the power to decipher it somehow with a non-negligible probability, without the knowledge of the key. Thanks to that, he can retrieve h_B. To have C_B accepted by the chip, $\mathcal{A}$ would have to replace h_B with $H(g^{x_A \cdot y})$, because at this point it is equal to $H(g^{x_A \cdot y \cdot \alpha})$ due to the adversary's manipulation of X_A. As H is modeled as a random oracle, that means that $\mathcal{A}$ has to produce $g^{x_A \cdot y}$, which he can do with only negligible probability without solving the Diffie-Hellman Computational Problem for (g, X_A, Y), which is the basis of the security of static Diffie-Hellman authentication.

For PACE MA-light: Any manipulations other than raising to the power $\alpha \neq 1$ values X_A and X_B lead to a session failure, as (augmented) T_A and T_B are sent between the parties in a ciphertexts C_A and C_B encrypted with K'_{SC}. Even if we gave the power to decrypt C_B to the adversary, he would need to transform tags $\mathbf{MAC}(K'_{\mathbf{MAC}}, g^{x_A \cdot y \cdot \alpha}, Y_A, \mathcal{G})$ to $\mathbf{MAC}(K'_{\mathbf{MAC}}, g^{x_A \cdot y}, Y_A, \mathcal{G})$ which is practically infeasible in random oracle model for tags, based on the same arguments as above.

That leads to a corollary that any active man-in-the middle attack againts PACE MA and PACE MA-light can be reduced to a passive one.

Protection of secrets

With an authentication, there may be a risk of leaking the secrets of the authenticating parties. Observe that for PACE MA and PACE MA-light those secrets are never sent in clear, they are used to compute tokens K_A and K_B which are later put into a hash function or a MAC. If an adversary is able to recover x or y with a non-negligible probability, this means that there is a reduction solving the Discrete Logarithm Problem (Assumption 2.1) for X or Y.

There is also another concern related to the authentication, different from recovering x or y by $\mathcal{A}$. Can $\mathcal{A}$ authenticate himself as a protocol party with a non-negligible probability? This would effectively mean that the adversary could be used as a plug in-procedure in solving an instance of the Computational

Diffie-Hellman Problem (Assumption 2.2) with a non-negligible probability is some specific setting.

When it comes to testing the password π by the adversary the situation is reduced to the situation of PACE CAM in PACE MA. Messages C_A and C_B do not provide the adversary any non-negligible advantage when it comes to the knowledge of π, because the only way to decipher them is via the knowledge of K'_{SC}, which requires the knowledge K (because of random oracle model for hash functions). Probability of the adversary deriving the right key K using $\pi' \neq \pi$ is negligible as shown in Section 3.4, so the only way to get information about the correct π is brute-force testing.

In PACE MA-light, the adversary has even less information than in PACE MA, as the tags T_A and T_B that could be potentially useful in some way are now encrypted, and the argument above is still valid.

Session key security

The main goal of the protocol is to establish a secure channel, meaning that $\mathcal{A}$ cannot derive keys $K_{\mathbf{Enc}}$ and $K_{\mathbf{MAC}}$. To show that, it is enough to consider a security experiment in which the adversary is presented with a bit string at the end of the protocol and needs to distinguish if it is a random bit string or if it is the key $K_{\mathbf{Enc}}$. While this is of course impossible for PACE as was shown by Theorem 3.3, in the case of PACE MA, the adversary has additionally the messages C_A and C_B to maybe provide him some non-negligible advantage. Let us bring the capabilities of the adversary to the ones from the original PACE. As usual, assume the random oracle model for hash functions. Let us give $\mathcal{A}$ an extra power: he would be given K'_{SC} in case of PACE MA and additionally $K'_{\mathbf{MAC}}$ in case of PACE MA-light and x and y with certificates for the public keys and thus he will to be able to fully recompute the messages C_A and C_B himself in both cases. That brings the situation of $\mathcal{A}$ to the one from the original PACE, and the adversary still has no means to derive K as the parameters given to him are independent from it (x and y) or they are not useful because of the random oracle model (K'_{SC} and $K'_{\mathbf{MAC}}$).

Simultability

In PACE MA and PACE MA-light it is possible to generate protocol transcripts without knowing the secret values x and y in the sense of Definition 3.3 as a chip, or a terminal or a third party, so this is not a proof that a successful session has occurred. Observe that it is possible to choose all ephemeral values for both sides and knowing them, compute the authentication tokens K_A and K_B without knowing the secrets x and y, in the same way as the verifier recomputes them in the protocol – using the public key X or Y and their ephemeral value x_B or x_A. For the extensions considered here, Phase 1 is identical to the one from PACE. Consequently, revealing both s and z always leads to revealing of π, as it is feasible to list all possible z's for a particular s for different password candidates. However, even if the terminal holder decides to reveal the right π

this way (which therefore compromises the chip if the password is not temporary for the chip, as in the case of e-passports), it is still not a proof that the session has occurred, as they could simulate it themselves.

4.2.3 Discussion on design

Both PACE MA and PACE MA-light are examples of following good practices when extending a protocol from a technical point of view. For PACE MA:

- The changes are introduced after the messages exchanged in the original version, which makes it easier to fall back to the original protocol (backward compatibility) and to argue about security.
- The original messages X_A and X_B are reused for different purposes, which result in reusing executable code and saving memory.
- The order and structure of messages is the same as in the original PACE for the most part of the protocol.
- There are not many changes to the scope of software/hardware operations. In addition to what is already implemented, what could be needed is potentially an additional hash function and a procedure for checking the certificates issued by CA.

For PACE MA-light:

- The number of messages is the same as in PACE.
- Original messages are reused – X_A and X_B.
- The structure of messages is almost the same for the most part of the protocol.
- There are not many new operations, like in PACE MA.

Observe that some design features are almost identical, which makes sense as similar strategies and tools to achieve new functionalities are used and also core functionality is preserved. There is also high degree of reusing what is already deployed in PACE – reusing already exchanged messages X_A and X_B for the authentication..

4.3 PACE Proof of Presence

In this section, another extension is discussed – PACE Proof of Presence. This protocol was presented at IFIP Networking conference in 2021 as a short presentation and was published in [16].

4.3.1 Design and functionality of PACE PoP

PACE Proof of Presence (PACE PoP) is an extension of the original PACE that allows the chip to obtain a *proof of presence* - a cryptographic confirmation that a successful session of PACE has taken place between the chip and the terminal. This is obtained by introducing a version of Schnorr identification protocol into the messeges of PACE, which results in producing a Schnorr signature by the chip for the session that can be stored and later used as a proof of successful session in front of a third party.

This is a meaningful contribution and indeed an extension, as in the original protocol the transcript itself is not a proof that can be used by the chip that the interaction occurred. While it is a required feature from the point of view of privacy protection and GDPR, in various scenarios that could be a desired functionality. For example, an inspector controlling technical installations in the field may be obliged to provide proof that he has actually visited certain physical locations when presenting the bill. This extension is a very good example of how a protocol can be slightly altered with some parts reused, providing a new functionality at very low cost in terms of rework required.

The extension is based on PACE where one additional message is added at the end, and there are some computational modifications introduced on the side of the terminal in the phase of establishing the key and computations on the side of the chip at the end. A visual description of the modified protocol is presented in Figure 4.3 with gray boxes indicating changes compared to PACE. The idea to achieve the required functionality is to reuse some of the messages and internal data as parts of the produced Schnorr signature (see Section 2.4.3 for a reference to the Schnorr signature). In some sense, it is a reversion of PACE CAM, where it is the terminal that authenticates itself instead of the chip with the added functionality of providing the proof of interaction by means of the "signature of the session" with the terminal. It is also different from PACE MA(-light) where both sides authenticate themselves, but the transcript itself is not a proof of interaction of the chip and the terminal, as it can be simulated.

For reference of the described modifications, see Figure 4.3.

Setting

In PACE PoP, in addition to the setting needed for PACE, the terminal has a pair of private and public keys: $(z_B, Z_B = g^{z_B})$ and also a certificate $cert(Z_B)$ for key Z_B. This certificate should be created/signed by a Certificate Authority whose public key is known to the chip so that it can verify its validity (technical details concerning PKI framework can be omitted here). Furthermore, the chip and the terminal agree on a secure hash function $H : (\{0,1\}^*, \mathcal{G}, \mathcal{G}) \longrightarrow \mathbb{Z}_q \backslash \{0\}$ among other hash functions and a MAC algorithm. This can be done during the initial phase of communication, which is used to establish the domain parameters (ICAO specification [2] provides details on that). Observe that in the original Schnorr Signature, the public key is of the form g^{-z_B}, which could be easily switched, as $-z_B$ would be just another element of the group. The procedure

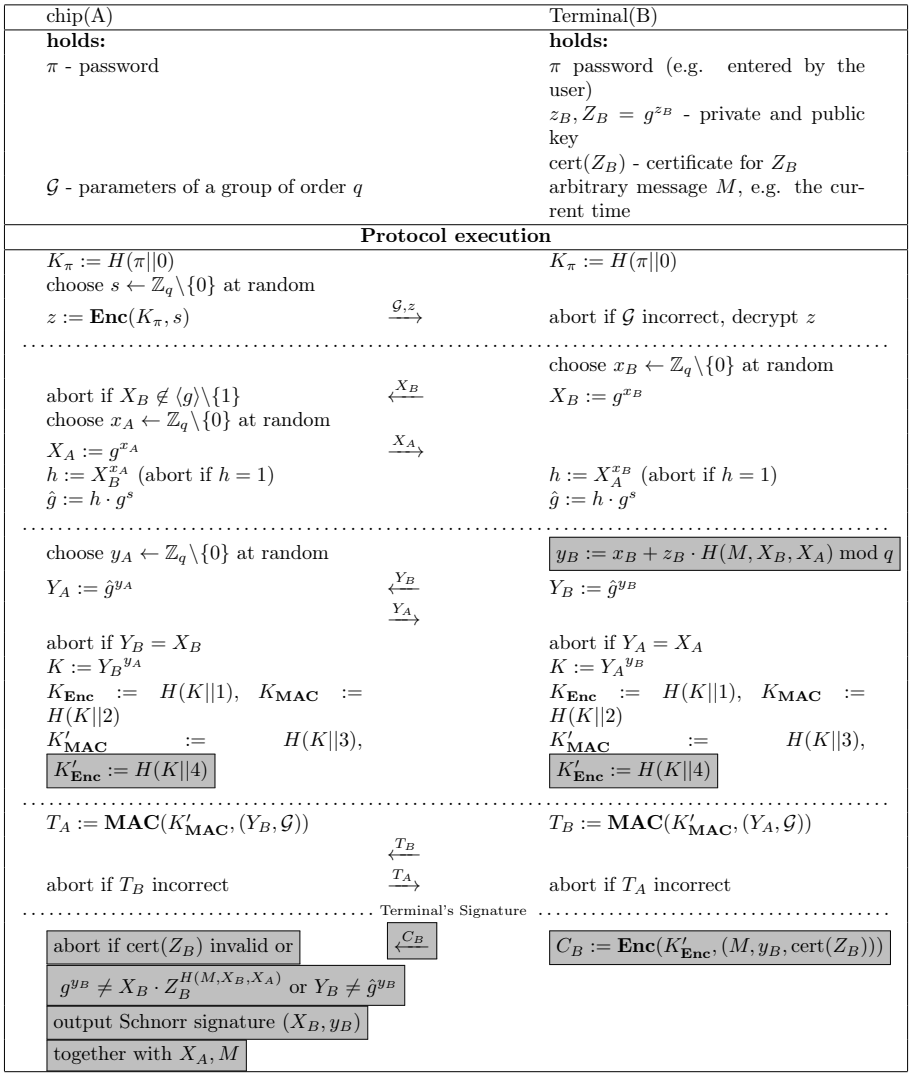

Figure 4.3: PACE PoP – a proof of presence for the chip. H stands for a hash function where the choice of the function depends on the context – the target image of the hash function. Grey boxes indicate the changes to the original PACE GM protocol.

to generate the signature would be slightly different, but this version is chosen for better readability and to fit the notation used in PACE.

Proof of Presence

The protocol is not modified up to the phase of establishing the common key K. In this phase, on the side of the terminal the value Y_B is created differently from the original protocol. In the original version $Y_B := \hat{g}^{y_B}$, where y_B is chosen at random from the appropriate group. Here, y_B is created to be a component of the Schnorr signature and calculated as $y_B := x_B + z_B \cdot H(M, X_B, X_A) \bmod q$.

Observe that this modification inserts elements of the interactive Schnorr identification protocol (Section 2.4.4) between the terminal and the chip into PACE. In addition to that, messages from both sides are used to form a Schnorr signature for the session. The message M can be an arbitrary string of bits of non-zero length that suits the purposes of the particular protocol needs. As the content of the message is arbitrary, it can be, for example, a timestamp, the physical location of the terminal, or maybe even some other cryptographic token or an encrypted message. This may open some possibilities to extend the protocol further. y_B is sent to the chip together with the certificate of the terminal $\text{cert}(Z_B)$ and the message M and encrypted with $K'_{\mathbf{Enc}}$ created as $H(K||4)$, where K is the key resulting from the Diffie-Hellman key exchange. C_B is the ciphertext.

Verification and finishing the protocol session

After decryption of C_B, the chip obtains three components: y_B, M and $\text{cert}(Z_B)$. Then it performs the following steps:

1. Check validity of $\text{cert}(Z_B)$ - abort the session if it is invalid.
2. Check if $g^{y_B} = X_B \cdot Z_B^{H(M,X_B,X_A)}$ and if $Y_B = \hat{g}^{y_B}$. If at least one of those is false, abort the session.
3. Output (X_B, y_B) as a signature for X_A, M and/or store it for future reference.

After returning the signature, with the knowledge of the public key of the terminal, one can verify that the signature is valid by checking if

$$g^{y_B} = X_B \cdot Z_B^{H(M,X_B,X_A)}.$$

This is also the part of the procedure above when the chip verifies that the returned signature is valid and there was no tempering with the session.

4.3.2 Analysis of the extension

Observe that the signature is tied not only to the identity of the terminal by means of the secret key x but also to the session itself, as it binds the signature to the values X_A, X_B, y_B. When participants follow the protocol specification, the probability that those values reappear at the same terminal in some other session is negligible, as it is 2^{-3q}.

Note that any possibility of forging the session signature would effectively mean that it is possible to forge the Schnorr Signature for some specific messages. Therefore, if the chip can present a valid signature (X_B, y_B) for some X_A, M it means that the terminal having a public key Z_B must have participated in the creation of the signature, also using X_A provided by the chip. Note also that the signature is created only after the exchange of valid tags T_B, T_A, so after a successful session with the key K is established (with all but negligible probability).

Observe also that this is a proof of presence only in case when the chip presents the signature and not in case when the terminal does it. The terminal is capable of simulating the whole execution of the protocol for some chosen π'. For some situations, like e-passports, the chip may deny its participation at the cost of revealing the correct password if for $\pi \neq \pi'$, $z \neq \mathbf{Enc}(K_\pi, s)$, at the same time possibly compromising the document with password π. However, if terminal knows the correct π or if the chip is not bound to a constant π, the simulated transcript will be indistinguishable from a real one, so this proof should be considered useless.

Those considerations can be summarized with a claim for PACE PoP:

Claim 4.0.2. *The probability of forging a valid signature (X_B, y_B) over a message (X_A, M) is negligible without the knowledge of z_B, and therefore a chip presenting it to the third party confirms that a successful session occurred at some point in time with the terminal with all but negligible probability.*

Note that the proof is useful only in situations where the terminal wants to deny that a session has occured. When the chip and the terminal cooperate, it is not a proof that the session took place, as the terminal is able to produce the transcript and signature itself and provide it to the chip. However, this might be technically problematic, if the chip has a fixed state machine for communication protocols, and no protocol execution deviating from the chip specification is possible.

Now, the following points can be potential concerns about the security of the scheme:

Revealing y_B

As y_B is revealed as a part of the signature, that could raise concerns that it could lead to leakage of the secret key z_B of the terminal, as $y_B = x_B + z_B \cdot H(M, X_B, X_A) \mod q$ and X_B, while X_A and M are made public (or could be when presenting the proof). Observe however, that $z_B = \frac{y_B - x_B}{H(M, X_B, X_A)} \mod q$ and x_B is never revealed. For $\mathcal{A}$, calculating x_B would mean breaking the Discrete Logarithm Problem for X_B, which can be done with negligible probability only.

y_B being not random

Security of Diffie-Hellman key exchange relies on the fact that $\mathcal{A}$ is not able to break the Computational Diffie-Hellman Problem with a non-negligible probability. There may be therefore a concern that if y_B is not chosen uniformly at random, it could somehow make it easier for $\mathcal{A}$ to find the discrete logarithm of Y_B or provide him with some other non-negligible advantage. Observe, however, that y_B in PACE PoP is created using a randomly chosen value x_B which is not revealed at any point which makes it impossible to distinguish y_B computed according to the protocol from a randomly chosen y_B. Y_B could potentially reveal that information, as $Y_B = \hat{g}^{y_B}$, but y_B is the discrete logarithm of Y_B which by itself is solvable with negligible probability only, and additionally no participant knows the discrete logarithm of $\hat{g}$. As there is no non-negligible advantage in distinguishing a random y_B from non-random, it follows that the adversary has the same advantage in this problem as in PACE.

Comment about X_A

There may be a doubt about how X_A fits in the Schnorr Identification Scheme, as it comes from a different group – $\mathcal{G}$ than intended for the calculation of the particular part of the signature – $\mathbb{Z}_q$, however, X_A together with X_B and M are later treated as input for H and this transforms it to the right group.

4.3.3 Reduction to the security of the original PACE

Consider now the ciphertext C_B created with $K'_{\mathbf{Enc}}$. Let us consider an artificial protocol $\mathcal{F}$, where $K'_{\mathbf{Enc}}$ is chosen at random instead of being computed as $K'_{\mathbf{Enc}} = H(K||4)$. Following an analogous discussion as in the proof of Theorem 3.3 we can claim that $\mathcal{A}$ cannot distinguish between a protocol $\mathcal{F}$ and PACE PoP, which in turn reduces his capabilities to those proved in Chapter 3 and by that reduces security of PACE PoP to security of PACE in terms of properties discussed - fragility (with a comment below), key confidentiality and password security.

4.3.4 Fragility

In the original PACE, the only thing the adversary is able to do without breaking the session (in the sense of Scenario 3.1) is to raise X_A and X_B to the same power $\alpha \neq 1$ known to him. Observe that in PACE PoP such a modification is impossible, as $H(M, X_B, X_A)$ is computed on both sides at some point. For the chip to verify the value y_B as valid with the modification described, $\mathcal{A}$ needs to change somehow value of Y_B, to make it verifiable for the chip later. However, as there was proven in Claim 3.2.2 any change of Y_A or Y_B is impossible with practically negligible probability without breaking the session. Therefore, a claim can be formulated:

Claim 4.0.3. *PACE PoP is fragile with all but practically negligible probability.*

4.3.5 Simultability

For PACE PoP one of the most important considerations is that someone can create a protocol session transcript in a way that is indistinguishable from a real transcript of the protocol (in the sense of information theory), like it is possible in plain PACE, PACE CAM (Theorem 3.14), or even PACE MA as discussed in Section 4.2.2. In this case, however, providing such a transcript without z_B means effectively forging Schnorr Signature for some specific messages, which can be done with negligible probability only.

4.3.6 Design with reusability in mind

The design of PACE PoP follows many rules described in Section 4.1. Note that computations concerning extended functionality are added at the end of the protocol, within a separate message, which could make it easier to implement than in the case where there are added messages in the middle of the original protocol. This also makes it easily backward compatible on the side of the terminal, as it can simply turn off the extra functionality when the chip does not support it. It can be recognized at the moment of establishing domain parameters for the protocol even before the protocol starts, when the chip responds without providing an option for H used in the signature. It could also be the case that the chip simply disregards C_B as junk and follows through as in the regular PACE.

In terms of additional operations, the chip needs to be able to verify the terminal certificate and possibly evaluate H if the engineering choice would be different from other hash functions already implemented. All other operations are already deployed, like operations in the group and exponentiation.

In addition to all the considerations above, the design choice of reusing messages y_B and X_A to serve two different purposes is a primary example of a smart design mindset. Thanks to that approach, there is only one message added at the end to provide a powerful, nontrivial functionality. For smart cards with limited capacity in terms of memory and computational power, that can make a real difference.

Chapter 5

Summary

In this thesis, they were presented refined and very detailed proofs of security features for PACE and PACE CAM, developed from sketches and ideas presented first in "Privacy and security analysis of PACE GM protocol" by Mirosław Kutyłowski and Przemysław Kubiak in 2019. The requirement-driven approach was proven to work well in this security analysis, filling the gaps for crucial security concerns for the discussed protocols, like many scenarios where password security could be endangered in different ways. Among others, a property called $fragility$ was proven to be true for PACE and PACE CAM and it turns out that it helps tremendously in the security analysis of other security features. Reducing all possible active adversaries to a passive one greatly simplifies other proofs and could be used as a general approach when analyzing interactive protocols. The discussion on negligible probability in the classical cryptographic sense versus practically negligible probability was also provided and argumented that the latter can be sufficient in particular scenarios.

Outside of obvious security features needed to be considered, like key confidentiality, other potential threats were discussed with strong results, like discussion on hijacking the session and privacy in terms of simultability of the protocol in the context of proof-of-presence. Such a security analysis could be regarded by some as more valuable, because of the focus on very real requirements instead of theoretical models.

Except for filling the gap of discussion on security issues of widely used protocols, there is also a useful contribution provided with proposed extensions of the protocol. Although interesting as research topic, what is shown by the fact that many publications with modifications exist, the extensions are motivated above all by the practical needs and legal context. Recent EU regulations provide an opportunity to develop a wide system of protocols based on PACE, enclosed in one hardware token, which could be universal across the member states as parts of electronic layer of the issued eIDs. Work presented in this thesis follows good design practices, making it easy to deploy the extensions and earn the general public trust for them by means of reusing when possible the robust security analysis. This particular contribution included two modifica-

tions extending functionalities of the original protocols PACE and PACE CAM by proposing mutual authentication in two variants and also a functionality of creating a cryptographic signature for the session, the so-called proof of presence. These are examples of a good level of reusing what is already accessible that could prompt other research to follow this approach.

Some of the future research directions include further development of the extensions, by providing more detailed security analysis and creating new extensions for well-motivated scenarios, leaving the realm of eIDs when needed. When it comes to the security analysis itself, a potential challenge could be to provide the proofs in different settings, replacing random oracle models with other assumptions, bringing the proofs a little closer to the technical reality.

Bibliography

[1] The European Parliament and the Council of the European Union. Regulation (EU) 2019/1157 — strengthening the security of identity cards and of residence documents issued to EU citizens and their family members exercising their right of free movement. Official Journal of the European Union, 188(67), 2019.

[2] ICAO. Machine Readable Travel Documents - Part 11: Security Mechanism for MRTDs. Doc 9303, 2015.

[3] BSI. Advanced security mechanism for machine readable travel documents extended access control (eac). In Technical Report (BSI-TR-03110) Version 2.05 Release Candidate. Bundesamt fuer Sicherheit in der Informationstechnik (BSI), 2010.

[4] Jens Bender, Marc Fischlin, and Dennis Kügler. Security Analysis of the PACE Key-Agreement Protocol. In Information Security, pages 33–48, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In Serge Vaudenay, editor, Public Key Cryptography - PKC 2005, pages 65–84, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Miroslaw Kutylowski and Przemyslaw Kubiak. Privacy and Security Analysis of PACE GM Protocol. In 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pages 763–768, 2019.

[7] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. The PACE|AA Protocol for Machine Readable Travel Documents, and Its Security. In Proc. of 16th International Conference Financial Cryptography and Data Security, pages 344–358, 2012.

[8] Lucjan Hanzlik, Lukasz Krzywiecki, and Miroslaw Kutylowski. Simplified PACE|AA Protocol. In Prof. of 9th International Conference Information Security Practice and Experience, pages 218–232, 2013.

[9] Jens Bender, Marc Fischlin, and Dennis Kügler. The PACE|CA Protocol for Machine Readable Travel Documents. In Prof. of 5th International Conference Trusted Systems, pages 17–35, 2013.

[10] Nicolas Buchmann, Roel Peeters, Harald Baier, and Andreas Pashalidis. Security considerations on extending PACE to a biometric-based connection establishment. In Proc. of the 12th International Conference of Biometrics Special Interest Group, pages 15–26, 2013.

[11] Lucjan Hanzlik and Miroslaw Kutylowski. Chip Authentication for E-Passports: PACE with Chip Authentication Mapping v2. In Prof. of 19th International Conference Information Security, pages 115–129, 2016.

[12] Jean-Sébastien Coron, Aline Gouget, Thomas Icart, and Pascal Paillier. Supplemental Access Control (PACE v2): Security Analysis of PACE Integrated Mapping, pages 207–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[13] Adam Bobowski and Mirosław Kutyłowski. Derandomized PACE with Mutual Authentication. In Network and System Security, pages 697–705, Cham, 2019. Springer International Publishing.

[14] E. E. Trukhina N. N. Shenets. X-PACE: Modified Password Authenticated Connection Establishment Protocol. AUTOMATIC CONTROL AND COMPUTER SCIENCES, 51(8):972–977, 2017.

[15] Patryk Kozieł, Przemysław Kubiak, and Mirosław Kutyłowski. PACE with Mutual Authentication – Towards an Upgraded eID in Europe. In Computer Security – ESORICS 2021, pages 501–519. Springer International Publishing, 2021.

[16] Mirosław Kutyłowski, Przemysław Kubiak, Patryk Koziel, and Yanmei Cao. Poster: eID in Europe - Password Authenticatio Revisited. In 2021 IFIP Networking Conference (IFIP Networking), pages 1–3, 2021.

[17] Feng Bao, Robert H. Deng, and HuaFei Zhu. Variations of Diffie-Hellman Problem. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, Information and Communications Security, pages 301–312, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[18] Ivan Damgård. Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. In Joan Feigenbaum, editor, Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings, volume 576, pages 445–456. Springer, 1991.

[19] Mihir Bellare and Adriana Palacio. The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. Cryptology ePrint Archive, Report 2004/008, 2004.

[20] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In Gilles Brassard, editor, Advances in Cryptology — CRYPTO' 89 Proceedings, pages 239–252, New York, NY, 1990. Springer New York.

[21] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical Report 800-38B, NIST, 2005.

[22] The European Parliament and the Council of the European Union. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union, 119(1), 2016.