An extended abstract of this work appears in Financial Cryptography and Data Security - 16th International Conference (FC 2013).

The PACE|AA Protocol for Machine Readable Travel Documents, and its Security

Jens Bender[1] Özgür Dagdelen[2] Marc Fischlin[2] Dennis Kügler[1]

[1] German Federal Office for Information Security (BSI), Germany
[2] Darmstadt University of Technology, Germany

Abstract. We discuss an efficient combination of the cryptographic protocols adopted by the International Civil Aviation Organization (ICAO) for securing the communication of machine readable travel documents and readers. Roughly, in the original protocol the parties first run the Password-Authenticated Connection Establishment (PACE) protocol to establish a shared key and then the reader (optionally) invokes the Active Authentication (AA) protocol to verify the passport's validity. Here we show that by carefully re-using some of the secret data of the PACE protocol for the AA protocol one can save one exponentiation on the passports's side. We call this the PACE|AA protocol. We then formally prove that this more efficient combination not only preserves the desirable security properties of the two individual protocols but also increases privacy by preventing misuse of the challenge in the Active Authentication protocol. We finally discuss a solution which allows deniable authentication in the sense that the interaction cannot be used as a proof towards third parties.

1 Introduction

Through ISO/IEC JTC1 SC17 WG3/TF5 [ICA10] the International Civil Aviation Organization (ICAO) has adopted the Password Authenticated Connection Establishment (PACE) protocol [BSI10] to secure the contactless communication between machine-readable travel documents (including identity cards), and a reader. Roughly, the protocol generates a secure Diffie-Hellman key out of a low-entropy password which the owner of the passport has to enter at the reader, or which is transmitted through a read-out of the machine-readable zone. The Diffie-Hellman key is subsequently used to secure the communication. In [BFK09] it has been shown that the PACE protocol achieves the widely accepted security notion of password-based authenticated key agreement of Bellare-Pointcheval-Rogaway [BPR00], in its strong form of Abdalla et al. [AFP05]. This holds under a variant of the Diffie-Hellman assumption, assuming secure cryptographic building blocks, and idealizing the underlying block cipher and the hash function.

According to European endeavors, the PACE protocol should be followed by the extended access control (EAC) authentication steps, called Chip Authentication (CA) and Terminal Authentication (TA), with high-entropic certified keys. This should ensure that access for either party is granted based on strong cryptographic keys (i.e., not relying on low-entropy passwords only). The security of the EAC protocols and of the composition with PACE has been discussed in [DF10, BDF12].

In the specifications of the ICAO 9303 standard [ICA06] for the border control scenario, the normative document about machine-readable travel documents, however, only passive authentication of the passport is mandatory, where the passport essentially merely sends its (authenticated) data. Active Authentication (AA) of the passport, implemented through a signature-based challenge-response protocol, is only optional. If AA is not enforced this potentially allows to bypass authentication through cloning of passports. Even if AA is used, then the (plain) challenge-response protocol introduces a potential threat to privacy, as discussed in [BSI10] (see also [BPSV08b, BPSV08a, MVV07]). Namely, if the terminal can encode a time stamp or the location into the challenge, then the signature on that challenge can be used as a proof towards third parties about the location or time of the border check. In this sense, the passport cannot deny this interaction. This problem has been explicitly addressed in the European Chip Authentication protocol (where a message authentication code for a shared key is used for the challenge-response step instead).

Combining PACE and AA. We discuss that, on the chip's side, we can re-use some of the (secret) data in the PACE step for the AA step to save the exponentiation for the signature in AA on the chip's side, giving Active Authentication (almost) for free.

To understand our technique, we need to take a closer look at the PACE protocol. The PACE protocol first maps the short password to a random group element through an interactive sub protocol $\mathsf{Map2Point}$, followed by a Diffie-Hellman key exchange step for this group element, and concludes with an authentication step. While the latter steps are somewhat canonical, the $\mathsf{Map2Point}$ step can be instantiated by different means and allows a modular design. The most common instantiations are based on another Diffie-Hellman step (used within the German identity card), or on hashing into elliptic curves as proposed by Icart [Ica09] and Brier et al. [BCI+10]. The security proof for PACE [BFK09] holds for general $\mathsf{Map2Point}$ protocols satisfying some basic security properties.

Our improvement works for the Diffie-Hellman based $\mathsf{Map2Point}$ protocol as implemented on the German identity cards, for example, since the chip can re-use its secret exponent from the Diffie-Hellman step of the $\mathsf{Map2Point}$ protocol. We discuss two alternatives how to carry out the AA step with this exponent more efficiently, one based on DSA signatures and the other one using Schnorr signatures. We note that the idea applies more generally to other discrete-log based signature schemes. The challenge in the new AA step is now the authentication data sent by the terminal in the PACE step.

Security of the Combined Protocol. Whenever secret data is used throughout several sub protocols great care must be taken in cryptography not to spoil the security of the overall protocol. We thus show that sharing the data between the PACE protocol and the new AA sub protocol preserves the desirable security properties. More precisely, we show that:

- In the combined PACE|AA protocol we still achieve the security of a password-based authenticated key exchange protocol (thus showing that the deployment of the randomness in the extra AA step does not violate the security of the PACE protocol), and
- the overall protocol still authenticates the chip securely (in a high-entropy sense), even when many executions of PACE|AA take place. To this end, we define a strong security model for authentication, essentially only excluding trivial attacks, e.g., if the adversary gets possession of the secret key, or simply relays information in executions.

It follows that the PACE|AA protocol achieves the previous security standards of the individual protocols but comes with a clear efficiency improvement. We note that the underlying assumptions are essentially the same as for PACE and AA, i.e., besides the common assumptions about secure encryption, signature, and MAC algorithms, we reduce the security of the combined protocol to the security of PACE (as an authenticated key-exchange protocol) and to a variant of the security of Schnorr signatures resp. DSA signatures (where the adversary now also gets access to a decisional Diffie-Hellman oracle and can decide upon the message to be signed after seeing the first half of the signature).

A Deniable Schnorr Version. As explained before, for privacy reasons it may be important that the terminal cannot derive a proof for others from the interaction with the passport or identity card *that* an interaction took place. Put differently, the protocol should provide *deniable authentication* [DDN00]. This roughly means that the terminal could have generated its view in the protocol itself from the public data, without communicating with the passport. This implies that the passport holder can deny any actual interaction and claim the terminal to have made up this conversation.

We note that the previously discussed signature based protocols do not support deniability. The reason is that the terminal could not have created the signature under the passport's key without the signing key —or without communicating with the actual chip. For the (ordinary) AA variant the terminal is even allowed to encode *any* information in the challenge, in our improved combinations the challenge is "only" a MAC computed over data provided by the passport and the shared Diffie-Hellman key. If this allows to encode information depends on the MAC.

In contrast, our proposed deniable variant does not rely on Schnorr signatures, but in some sense rather on the interactive Schnorr identification scheme for honestly chosen challenges. This identification scheme is deniable because one can simulate the interaction via the well-known zero-knowledge simulator.[1] Interestingly, our variant is essentially as efficient as the signature based one, but comes with the advantage of deniability.

Organization. In Section 2 we discuss the security model for authenticated key exchange and impersonation resistance. Section 3 then presents the PACE|AA protocol (variants). In Section 4 we discuss the relevant (number-theoretic and cryptographic) security assumptions, before we present our security proofs in Section 5. Finally, in Section 6 we discuss our deniable version and its security.

2 Security Model

We use the real-or-random security model of Abdalla et al. [AFP05] which extends the model of Bellare et al. [BPR00] for password-based key exchange protocols. Some changes are necessary, though, because we now incorporate a long-term signing key of the chip.

Attack Model. The security model assumes a set of honest participants, also called users. Each user may run several instances of the key agreement protocol, and the j-th instance of a user U

[1]It is this property which is not known to work for the DSA case and why we restrict ourself to the Schnorr scheme. Note also that Schnorr signatures are also somewhat simulatable but only if one programs the random oracle hash function; this, however, is not admissible for the notion of deniability. We nonetheless still use a hash function in the solution but use programmability only to show the unforgeablity/impersonation resistance property, not the deniability proof.

is denoted by U_j or (U, j). Each pair of users initially shares a secret password π —which in the passport scenario is transferred from the chip to the terminal by entering it at the reader or by reading the machine readable zone. The password π may be used multiple times to generate session keys. We assume that the password is chosen randomly from a (public) dictionary with N elements. A set of users, called the chip cards (or chips), also hold a long-lived key pair (sk, pk) and we assume that the public key is registered with a certification authority (e.g., some approved organization for identity cards). The certification authority somehow verifies the well-formedness of the keys, e.g., that they belong to an approved group. The other users in the scenario are called readers or terminals.

We consider security against active attacks where the adversary's goal is to distinguish genuine keys from random keys, which are picked independently of the actual protocol run. This corresponds to the so-called real-or-random setting [AFP05], a stronger model than the original find-then-guess model of [BPR00], where the adversary can see several test keys (instead of a single one only).

In the attack, each user instance is given as an oracle to which an adversary has access, basically providing the interface of the protocol instance. By assumption, the adversary is in full control of the network, i.e., decides upon message delivery. If honest parties may engage in many sessions at the same time, then we are in a *concurrent* setting; else, we are in a *non-concurrent* setting. Initially, the adversary is given all (registered) public keys of the users. These users are called *honest* whereas the other users, for which the adversary registers chosen public keys, are called *adversarially controlled*.[2]

The adversary can gain control over a user during the execution by issuing a Corrupt query with which the adversary obtains the secrets of an honest party. We need to modify here since we consider two types of secrets, i.e., the long-term secret and the password of the chip. Therefore, for sake of convenience, we split these queries into Corrupt.pw and Corrupt.key queries, where the former reveals the password only and the latter discloses the long-term key only (in case of a chip); in both cases, the other secret remains private. Note that we now can model Corrupt queries by both queries (since we work in the weak corruption model where the parties' internal states are not revealed upon corruption). An honest party gets adversarially controlled if it does not have any secrets left (i.e., if the adversary issues both Corrupt query types for a chip, or the Corrupt.pw query for the terminal). We occasionally speak of executions with honest chips or honest readers if the adversary (or an honest partner) runs an execution such that the party is not, and does not get, adversarial controlled during this execution.

The adversary can make the following queries to the interface oracles:

Execute(A, i, B, j) Causes the users A and B to run the protocol for (fresh) instances i and j. The final output is the transcript of a protocol execution. This query simulates a passive attack where the adversary merely eavesdrops the network.

Send(U, i, m) Causes the instance i of user U to proceed with the protocol when having received message m. The output is the message generated by U for m and depends on the state of the instance. This query simulates an active attack of the adversary where the adversary pretends to be the partner instance.

Reveal(U, i) Returns the session key of the input instance. The query is answered only if the session key was generated and the instance has terminated in accepting state. This query

[2]We remark that the adversary may register public keys chosen by honest parties on behalf of adversarially controlled users.

models the case when the session key has been leaked. We assume without loss of generality that the adversary never queries about the same instance twice.

Corrupt.pw(U) The adversary obtains the party's password π.

Corrupt.key(U) The adversary obtains the party's cryptographic key *sk* (if it exists).

Test(U, i) The oracle test is initialized with a random bit b (which is then fixed for all subsequent calls). Assume the adversary makes a test query about (U, i) during the attack and that the instance has terminated in accepting state, holding a secret session key k. Then the oracle returns k if $b = 0$ or a random key k' from the domain of keys if $b = 1$. If the instance has not terminated yet or has not accepted, then the oracle returns $\perp$. This query should determine the adversary's success to tell apart a genuine session key from an independent random key. We assume again without loss of generality that the adversary never queries about the same instance twice.

Register$(U^*, \boldsymbol{pk}^*)$ allows the adversary to register a public key pk^* in the name of a new user (identity) U^*. The user is immediately considered to be adversarial controlled and the password of the user is revealed to (or even chosen by) the adversary.

In addition, since the original PACE protocol was cast in the random oracle and ideal cipher model where oracles providing a random hash function oracle and an encryption/decryption oracle are available, the attacker may also query these oracles here. (We note that we only use the ideal cipher implicitly through the reduction to the security to PACE.)

Partners, Correctness and Freshness. Upon successful termination, we assume that an instance U_i outputs a session key k, the session ID *sid*, and a user ID *pid* identifying the intended partner (assumed to be empty in PACE for anonymity reasons but containing the chip's certificate in the combined PACE|AA protocol). We note that the session ID usually contains the entire transcript of the communication but, for efficiency reasons, in PACE it only contains a part thereof. This is inherited here. We say that instances A_i and B_j are *partnered* if both instances have terminated in accepting state with the same output. In this case, the instance A_i is called a partner to B_j and vice versa. Any untampered execution between honest users should be partnered and, in particular, the users should end up with the same key (this correctness requirement ensures the minimal functional requirement of a key agreement protocol).

Neglecting forward security for a moment, an instance (U, i) is called *fresh* at the end of the execution if there has been no $\mathsf{Reveal}(U, i)$ query at any point, neither has there been a $\mathsf{Reveal}(B, j)$ query where B_j is a partner to U_i, nor has somebody been corrupted (i.e., neither kind of Corrupt query has been issued). Else, the instance is called *unfresh.* In other words, fresh executions require that the session key has not been leaked (by neither partner) and that no Corrupt-query took place.

To capture forward security we refine the notion of freshness and further demand from a fresh instance (U, i) as before that the session key has not been leaked through a Reveal-query, and that for each $\mathsf{Corrupt.pw}(U)$- or $\mathsf{Corrupt.key}(U)$-query there has been no subsequent $\mathsf{Test}(U, i)$-query involving U, or, if so, then there has been no $\mathsf{Send}(U, i, m)$-query for this instance at any point.[3] In this case we call the instance *fs-fresh*, else *fs-unfresh.* This notion means that it should not help if the adversary corrupts some party after the test query, and that even if corruptions take

[3]In a stronger notion the adversary may even issue a Corrupt.key command for the user before the testing; Due to the entanglement of the PACE and the AA protocol here our protocol does not achieve this, though.

place before test queries, then executions between honest users are still protected (before or after a Test-query).

AKE SECURITY. The adversary eventually outputs a bit b', trying to predict the bit b of the Test oracle. We say that the adversary wins if $b = b'$ and instances (U, i) in the test queries are fresh (resp. fs-fresh). Ideally, this probability should be close to 1/2, implying that the adversary cannot significantly distinguish random keys from session keys.

To measure the resources of the adversary we denote by

t	the number of steps of the adversary, i.e., its running time, (counting also all the steps required by honest parties)
q_e	the maximal number of initiated executions (bounded by the number of Send- and Execute-queries),
q_h	the number of queries to the hash oracle, and
q_c	the number of queries to the cipher oracle.

We often write $Q = (q_e, q_h, q_c)$ and say that $\mathcal{A}$ is (t, Q)-bounded.

Define now the AKE advantage of an adversary $\mathcal{A}$ for a key agreement protocol P by

$$\mathbf{Adv}_P^{ake}(\mathcal{A}) := 2 \cdot \text{Prob}[\,\mathcal{A} \text{ wins}] - 1$$
$$\mathbf{Adv}_P^{ake}(t, Q) := \max \left\{ \mathbf{Adv}_P^{ake}(\mathcal{A}) \;\middle|\; \mathcal{A} \text{ is } (t, Q)\text{-bounded} \right\}$$

The forward secure version is defined analogously and denoted by $\mathbf{Adv}_P^{ake-fs}(t, Q)$.

IMPERSONATION RESISTANCE. This security property says that the adversary, in the above attack, *successfully impersonates* if an honest reader in some session accepts with partner identity *pid* and session id *sid*, but such that (a) the intended partner U in *pid* is not adversarially controlled or the public key in *pid* has not been registered, and (b) no Corrupt.key command to U has been issued before the reader has accepted, and (c) the session id *sid* has not appeared in another accepting session. This roughly means that the adversary managed to impersonate an honest chip or to make the reader accept a fake certificate, without knowing the long-term secret or relaying the data in a trivial man-in-the-middle kind of attack.

Define now the IKE advantage (I for impersonation) of an adversary $\mathcal{A}$ for a key agreement protocol P by

$$\mathbf{Adv}_P^{ike}(\mathcal{A}) := \text{Prob}[\,\mathcal{A} \text{ successfully impersonates}]$$
$$\mathbf{Adv}_P^{ike}(t, Q) := \max \left\{ \mathbf{Adv}_P^{ike}(\mathcal{A}) \;\middle|\; \mathcal{A} \text{ is } (t, Q)\text{-bounded} \right\}$$

Note that we do not need to define a forward secure version here.

3 The PACE|AA Protocol

In this section, we describe the PACE|AA protocol and both options for authentication in the last message, i.e., active authentication (AA) via Schnorr and via DSA. The deniable Schnorr variant and its security is addressed in Section 6.

A :		**B** :
password π		password π
secret x_A, public $X_A = g^{x_A}$		
certificate $cert_C$ for X_A, and $pk_{\mathcal{CA}}$		$pk_{\mathcal{CA}}$
authenticated group parameters $\mathcal{G} = (a, b, p, q, g, k)$		
	PACE	
$K_\pi = \mathcal{H}(0\|\|\pi)$		$K_\pi = \mathcal{H}(0\|\|\pi)$
choose $s \leftarrow \{0,1\}^\ell \subseteq \mathbb{Z}_q$		
$z = \mathsf{Enc}(K_\pi, s)$		
	$\xrightarrow{\mathcal{G}, z}$	abort if $\mathcal{G}$ incorrect
		$s = \mathsf{Dec}(K_\pi, z)$
choose $y_A \leftarrow \mathbb{Z}_q^*$		choose $y_B \leftarrow \mathbb{Z}_q^*$
$Y_A = g^{y_A}$		$Y_B = g^{y_B}$
	$\xleftarrow{Y_B}$	
abort if $Y_B \notin \langle g \rangle \setminus \{1\}$		
	$\xrightarrow{Y_A}$	abort if $Y_A \notin \langle g \rangle \setminus \{1\}$
$h = Y_B^{y_A}$		$h = Y_A^{y_B}$
$\hat{g} = h \cdot g^s$		$\hat{g} = h \cdot g^s$
choose $y'_A \leftarrow \mathbb{Z}_q^*$		choose $y'_B \leftarrow \mathbb{Z}_q^*$
$Y'_A = \hat{g}^{y'_A}$		$Y'_B = \hat{g}^{y'_B}$
	$\xleftarrow{Y'_B}$	
check that $Y'_B \neq Y_B$	$\xrightarrow{Y'_A}$	check that $Y'_A \neq Y_A$
$\mathcal{K} = (Y'_B)^{y'_A}$		$\mathcal{K} = (Y'_A)^{y'_B}$
$\mathcal{K}_{\text{ENC}} = \mathcal{H}(1\|\|\mathcal{K})$		$\mathcal{K}_{\text{ENC}} = \mathcal{H}(1\|\|\mathcal{K})$
$\mathcal{K}'_{\text{SC}} = \mathcal{H}(2\|\|\mathcal{K})$		$\mathcal{K}'_{\text{SC}} = \mathcal{H}(2\|\|\mathcal{K})$
$\mathcal{K}_{\text{MAC}} = \mathcal{H}(3\|\|\mathcal{K})$		$\mathcal{K}_{\text{MAC}} = \mathcal{H}(3\|\|\mathcal{K})$
$\mathcal{K}'_{\text{MAC}} = \mathcal{H}(4\|\|\mathcal{K})$		$\mathcal{K}'_{\text{MAC}} = \mathcal{H}(4\|\|\mathcal{K})$
$\text{T}_A = \mathsf{MAC}(\mathcal{K}'_{\text{MAC}}, (Y'_B, \mathcal{G}))$		$\text{T}_B = \mathsf{MAC}(\mathcal{K}'_{\text{MAC}}, (Y'_A, \mathcal{G}))$
	$\xleftarrow{\text{T}_B}$	
abort if T_B invalid	$\xrightarrow{\text{T}_A}$	
		abort if T_A invalid
....	Version: Schnorr Signature	....
$\sigma = y_A + \mathcal{H}(5\|\|Y_A, T_B) \cdot x_A$	$\xrightarrow{\mathsf{Send}(\mathcal{K}'_{\text{SC}}, (\sigma, cert_C))}$	recover and validate certificate
		abort if $g^\sigma \neq Y_A X_A^{\mathcal{H}(5\|\|Y_A, T_B)}$
....	Version: DSA Signature	....
$r = Y_A \bmod q$		
$\sigma = y_A^{-1}(\mathcal{H}(5\|\|T_B) + r x_A)$	$\xrightarrow{\mathsf{Send}(\mathcal{K}'_{\text{SC}}, (\sigma, cert_C))}$	recover and validate certificate
		$w = \sigma^{-1}$
		$r = Y_A$
		$v = g^{w\mathcal{H}(5\|\|T_B)} \cdot X_A^{rw}$
		abort if $v \neq Y_A$
key$=(\mathcal{K}_{\text{ENC}}, \mathcal{K}_{\text{MAC}})$		key$=(\mathcal{K}_{\text{ENC}}, \mathcal{K}_{\text{MAC}})$
$sid = (Y'_A, Y'_B, \mathcal{G})$		$sid = (Y'_A, Y'_B, \mathcal{G})$
$pid = cert_C$		$pid = cert_C$

Figure 1: The PACE|AA protocol (all operations are modulo q)

3.1 Protocol Description

Figure 1 illustrates the PACE|AA protocol with both options of authentication at the end. The scheme itself uses a block cipher $\mathcal{C}(K_\pi, \cdot) : \{0,1\}^\ell \to \{0,1\}^\ell$ and a hash function $\mathcal{H}$, with values $1, 2, \ldots$ in fixed-length encoding prepended to make evaluations somewhat independent.

The chip already holds a certificate $cert_C$ for its public key X_A under the authorities' public key $pk_{\mathcal{CA}}$, and (authenticated) group parameters $\mathcal{G} = (a, b, p, q, g, k)$ describing a subgroup of order q, generated by g, of an elliptic curve for parameters a, b, p for security parameter k. We also note that, throughout the paper, we use the multiplicative notation for group operations. It is understood that, if working with elliptic curves, multiplications correspond to additions and exponentiations to multiplications. Then the parties run the PACE protocol, with the chip sending a nonce encrypted under the password, running the Diffie-Hellman based $\mathsf{Map2Point}$ protocol to derive another generator $\hat{g}$ on which another Diffie-Hellman key exchange is then performed. In this $\mathsf{Map2Point}$ step the chip uses some secret exponent y_A to send $Y_A = g^{y_A}$. The parties in the PACE protocol finally exchange message authentication codes $\mathrm{T}_A, \mathrm{T}_B$.

The idea is now roughly to re-use the secret exponent y_A in the $\mathsf{Map2Point}$ sub protocol on the chip's side for the signature generation, and use the authentication value T_B of the terminal as the challenge on which the signature is computed. The chip then sends its certificate (along with the missing signature part) over the secure channel, via a Send command for the key $\mathcal{K}'_{\mathrm{SC}}$ derived from the Diffie-Hellman exchange. The reader may think for now of the secure channel as an authenticated encryption, but other channel instantiations work as well.

3.2 Instantiations

There are essentially two possible instantiations. One is based on the Schnorr signature scheme [Sch90] where the chip uses the values y_A and Y_A as the (private resp. public) randomness and T_B as the challenge for creating the signature under its long-term signature key X_A. We call this option *Active Authentication via Schnorr signatures.* Alternatively, the chip card might prove its authenticity by providing a DSA signature where again y_A and Y_A are used as the randomness for the signature generation [Kra95]. This version is called *Active Authentication via DSA signatures.* We note that the computation of the final signatures requires only modular multiplications (and, in case of DSA, an inversion) instead of exponentiations.

4 Security Assumptions

As remarked above we carry out our security analysis assuming an ideal hash function (random oracle model). Basically, this assumption says that $\mathcal{H}$ acts like a random function to which all parties have access. We do not make any explicit assumption about the cipher $\mathcal{C}$ here, but note that the security proof for PACE in [BFK09] (to which we reduce AKE security to) relies on an ideal cipher.

4.1 Cryptographic Primitives

Message Authentication. A message authentication code $\mathcal{M} = (\mathsf{KGen}, \mathsf{MAC}, \mathsf{Vf})$ consists of three efficient algorithms where we assume again that keys are just random strings in the range of the hash function —making KGen obsolete— and that $\mathsf{MAC}(k, m)$ maps any message to a MAC (resp. tag) T which is verifiable with the help of $\mathsf{Vf}(k, m, T)$ with binary output. Completeness demands again that for any key k and any message m the value $T \leftarrow \mathsf{MAC}(k,m)$ makes $\mathsf{Vf}(k,m,T)$ return 1.

We require that the message authentication code MAC is unforgeable under adaptively chosen-message attacks (see, for instance, [Gol04]). That is, the adversary is granted oracle access to $\mathsf{MAC}(k,\cdot)$ and $\mathsf{Vf}(k,\cdot,\cdot)$ for random key k and wins if it, at some point, makes a verification query (m,T) about a message m which has not been sent previously to MAC, and such that Vf returns 1 for this message. We denote by $\mathbf{Adv}^{forge}_{\mathcal{M}}(t,q_m,q_v)$ a (bound on the) value ϵ for which no attacker in time t can win (making at most q_m MACs queries and q_v verification queries) with probability more than ϵ.

Signatures and Certificates. A signature scheme $\mathcal{S} = (\mathsf{SKGen},\mathsf{Sig},\mathsf{SVf})$ consists of efficient algorithms for creating key pairs (sk,pk), signing messages $s \leftarrow \mathsf{Sig}(sk,m)$, and verifying signatures, $d \leftarrow \mathsf{SVf}(pk,m,s)$ with $d \in \{0,1\}$. It must be that for signatures created under valid key pairs SVf always returns 1 (correctness). Unforgeability says that no algorithm should be able to forge the signer's signature. That is, a signature scheme $\mathcal{S} = (\mathsf{SKGen},\mathsf{Sig},\mathsf{SVf})$ is (t,q_s,ϵ)-unforgeable if for any algorithm $\mathcal{A}$ running in time t the probability that $\mathcal{A}$ outputs a signature to a fresh message under a public key is $\mathbf{Adv}^{\mathrm{forge}}_{\mathcal{S}}(t,q_s)$ (which should be negligible small) while $\mathcal{A}$ has access (at most q_s times) to a singing oracle. We note that for Schnorr signatures and DSA signatures, we actually need a stronger notion discussed in the next section.

We also assume a certification authority CA, modeled like the signature scheme through algorithms $\mathcal{CA} = (\mathsf{CKGen},\mathsf{Certify},\mathsf{CVf})$, but where we call the "signing" algorithm $\mathsf{Certify}$. This is in order to indicate that certification may be done by other means than signatures. We assume that the keys $(sk_{\mathcal{CA}},pk_{\mathcal{CA}})$ of the CA are generated at the outset and that $pk_{\mathcal{CA}}$ is distributed securely to all parties (including the adversary). We also often assume that the certified data is part of the certificate. We define unforgeability for a certification scheme $\mathcal{CA}$ analogously to signatures, and denote the advantage bound of outputting a certificate of a new value in time t after seeing q_c certificates by $\mathbf{Adv}^{\mathrm{forge}}_{\mathcal{CA}}(t,q_c)$. We assume that the certification authority only issues unique certificates in the sense that for distinct parties the certificates are also distinct; we besides assume that the authority checks whether the keys are well-formed group elements and that certificates are of a fixed length (depending on the security parameter only).

Secure Channel. A secure channel $\mathcal{SC} = (\mathsf{KGen},\mathsf{Send},\mathsf{Rec})$ consists of algorithms for generating keys KGen (we assume in this paper that the keys are random strings and that the hash function $\mathcal{H}$ maps to such strings), a sending algorithm $\mathsf{Send}(k,m)$ wrapping the message usually in an encrypted and authenticated container C, and a recover algorithm $\mathsf{Rec}(k,C)$ which on input a key k and a container C returns a message m or an error symbol $\perp$. We assume the usual notion of completeness that any faithfully wrapped message under any key is recovered through the recover algorithms. Roughly, we demand that a secure channel hides messages (as encryption schemes) but at the same time also provides authenticity of the sent messages (as in MAC schemes). Below we cover both notions in a single security experiment.

As for security of the secure channel, we consider left-or-right security in the multi-user setting. That is, we assume that u users pick random keys $k_1,k_2,\ldots,k_u$ and that a secret challenge bit $b \leftarrow \{0,1\}$ is chosen. The adversary $\mathcal{A}$ gets no input and can then access a sending oracle $\mathcal{O}_{\mathsf{Send}}(b,k_1,\ldots,k_u,\cdot,\cdot,\cdot)$ which for triples (i,m_0,m_1) returns a container $\mathsf{Send}(k_i,m_b)$ of the left or right message under the i-th key. Here, we assume that $1 \le i \le u$ and that m_0 and m_1 are of equal length. In addition, the adversary can ask a recovering oracle $\mathcal{O}_{\mathsf{Rec}}(k_1,\ldots,k_u,\cdot,\cdot)$ about indices i

and containers C of its choice, recovering either a message under key k_i or the error symbol. The adversary eventually outputs a guess $a \in \{0,1\}$ for the challenge bit b. To measure simultaneously successful attacks against the authenticity property we set $a \leftarrow b$ if at some point during the attack the adversary manages to submit a container C to the recover oracle $\mathcal{O}_{\mathsf{Rec}}$ such that it receives $m \neq \perp$ and such that C has not been created by oracle $\mathcal{O}_{\mathsf{Send}}$ for the same user before.

Let $\mathbf{Adv}_{\mathcal{E}}^{lor}(\mathcal{A})$ be the probability that $a = b$ minus the pure guessing probability $1/2$, taking also into account the deliberate switch of a to b in case of successful attacks against the authenticity. Let $\mathbf{Adv}_{\mathcal{E}}^{lor}(t,u,q)$ be the maximal advantage for any adversary running in time t, with access to at most u users, and making in total q queries. We note that a standard hybrid argument shows that the advantage increases only by a factor u when moving to the single-user case, i.e., $\mathbf{Adv}_{\mathcal{E}}^{lor}(t,u,q) \leq u \cdot \mathbf{Adv}_{\mathcal{E}}^{lor}(t,1,q)$. Hence, the common notion of security for symmetric schemes implies security in the multi-user setting (with a loss of a factor u).

4.2 Number-Theoretic Assumptions

Our proof for the AKE security of the PACE|AA protocol follows by reduction to the security of the original PACE protocol (and from the security of cryptographic primitives for the channel). For the IKE security against impersonators, we nonetheless need two number-theoretic assumptions related to the Diffie-Hellman resp. discrete-log problems. The first one is the gap Diffie-Hellman problem [BLS01]. For a group $\mathcal{G}$ generated by g let $\mathrm{DH}(X,Y)$ be the Diffie-Hellman value X^y for $y = \log_g Y$ (with g being an implicit parameter for the function). Then the gap Diffie-Hellman assumption says that solving the computational DH problem for (g^a, g^b), i.e., computing $\mathrm{DH}(g^a, g^b)$ given only the random elements (g^a, g^b) and $\mathcal{G}, g$, is still hard, even when one has access to a decisional oracle $\mathrm{DDH}(X,Y,Z)$ which returns 1 iff $\mathrm{DH}(X,Y) = Z$, and 0 otherwise.

Definition 4.1 (GDH Hardness) *The Gap Diffie-Hellman (GDH) problem is (t, q_{DDH}, ϵ)-hard if any algorithm $\mathcal{A}$, running in time t and making at most q_{DDH} oracle queries, makes the following experiment output 1 with probability at most ϵ:*

pick a cyclic group $\mathcal{G}$ of order q with a generator g and pick $a, b \leftarrow \mathbb{Z}_q$
let $Z \leftarrow \mathcal{A}^{DDH(\cdot,\cdot,\cdot)}(\mathcal{G}, g, g^a, g^b)$
output 1 iff $Z = DH(g^a, g^b) = g^{ab}$

We let $\boldsymbol{Adv}^{GDH}(t, q_{DDH})$ denote (a bound on) the value ϵ for which the GDH problem is (t, q_{DDH}, ϵ)-hard.

Furthermore, for the Schnorr signature based solution we rely on the following version which (a) allows access to a decisional DH oracle for the forger, and (b) considers access to a signer in an online/offline fashion in the sense that the adversary may ask to see the public randomness part first before deciding on a message to be signed. Still, the goal is to create a signature on a new message for which the signing has not been completed. We note that the proof in [PS00] for Schnorr signatures still holds, assuming that computing discrete-logarithms relative to a DDH-oracle is hard. In particular, the hardness of this "gap discrete-log problem" is implied by the GDH hardness. We call this security notion *robust* unforgeability as it should still hold in presence of the DDH oracle and the delayed message choice.

Definition 4.2 (Robust Unforgeability of Schnorr Signatures) *The Schnorr signature scheme is (t, Q, ϵ)-robustly-unforgeable with $Q = (q_R, q_{DDH})$ if for any adversary $\mathcal{A}$ running in total time*

t, making at most q_{DDH} DDH oracle queries and at most q_R init-queries to oracle $\mathcal{O}$ the probability that the following experiment returns 1 is most ϵ:

```
pick $\mathcal{G}$ (including a generator $g$ of prime order $q$)
pick $sk \leftarrow \mathbb{Z}_q$ and let $pk = g^{sk}$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(sk,\cdot),DDH}(\mathcal{G}, g, pk)$
parse $(c^*, s^*) \leftarrow \sigma^*$
output 1 iff
    $c^* = \mathcal{H}(g^{s^*} pk^{c^*}, m^*)$
    and $m^* \notin \mathsf{M}$
```

```
Set $\mathsf{id} = 0$ and $\mathsf{R}, \mathsf{M} = \emptyset$.
If $\mathcal{A}$ queries $\mathcal{O}(sk, \mathsf{init})$,
    pick $r \leftarrow \mathbb{Z}_q$,
    set $\mathsf{id} = \mathsf{id} + 1$ ,
    add $(\mathsf{id}, r)$ to $\mathsf{R}$,
    return $(\mathsf{id}, g^r)$.
If $\mathcal{A}$ queries $\mathcal{O}(sk, (\mathsf{complete}, \mathsf{id}, m))$,
    if $(\mathsf{id}, r) \in \mathsf{R}$ for some $r$,
        update $\mathsf{R} \leftarrow \mathsf{R} \backslash \{(\mathsf{id}, r)\}$
        add $m$ to $\mathsf{M}$,
        return $r + \mathcal{H}(g^r, m) \cdot sk \bmod q$;
    else, return $\bot$.
If $\mathcal{A}$ queries $DDH(X, Y, Z)$,
    return 1 iff $DH(X, Y) = Z$.
```

We let $\boldsymbol{Adv}^{r-forge}_{Schnorr}(t, Q)$ be the maximal advantage for any adversary running in time t, making in total $Q = (q_R, q_{DDH})$ queries.

As it turns out to be useful for our deniable version, we remark that the proof of Pointcheval and Stern [PS00] holds as long as the input to the hash oracle in the forgery is new, i.e., one can extract the discrete-logarithm of the public key even if the hash function in signature requests is evaluated on quasi unique inputs, and the forgery, too, uses a previously unqueried hash function input. For the notion of signature unforgeability this holds because each signature request uses a high-entropic random group element and the message m^* in the forgery cannot have been signed before. We take advantage of this fact for our deniable version where we insert $(Y'_A, \mathcal{G})$ instead of (R, m) into the hash function for the random group element Y'_A chosen by the chip respectively, signer. We also show that for the proof of impersonation resistance the adversary cannot re-use one of these values $(Y'_A, \mathcal{G})$ but needs to pick a new value Y'_A, thus showing the second property.

For the DSA based solution we require an analogous assumption:

Definition 4.3 (Robust Unforgeability of DSA Signatures) *The DSA signature scheme is (t, Q, ϵ)-robustly-unforgeable with $Q = (q_r, q_{DDH})$ if for any adversary $\mathcal{A}$ running in total time t, making at most q_{DDH} DDH oracle queries and at most q_r init queries to oracle $\mathcal{O}$ the probability that the following experiment returns 1 is most ϵ:*

```
pick $\mathcal{G}$ (including a generator $g$ of prime order $q$)
pick $sk \leftarrow \mathbb{Z}_q$ and let $pk = g^{sk}$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(sk,\cdot),DDH}(\mathcal{G}, g, pk)$
parse $(c^*, s^*) \leftarrow \sigma^*$
output 1 iff
    $r^* = g^{w\mathcal{H}(m^*)} pk^{wr^*} \bmod q$
        for $w = (s^*)^{-1} \bmod q$,
    and $m^* \notin \mathsf{M}$
```

```
Set $\mathsf{id} = 0$ and $\mathsf{R}, \mathsf{M} = \emptyset$.
If $\mathcal{A}$ queries $\mathcal{O}(sk, \textsf{init})$,
    pick $k \leftarrow \mathbb{Z}_q$,
    set $\mathsf{id} = \mathsf{id} + 1$ ,
    add $(\mathsf{id}, k)$ to $\mathsf{R_L}$,
    return $(\mathsf{id}, k)$.
If $\mathcal{A}$ queries $\mathcal{O}(sk, (\textsf{complete}, \mathsf{id}, m))$,
    if $(\mathsf{id}, k) \in \mathsf{R}$ for some $k$,
        update $\mathsf{R} \leftarrow \mathsf{R} \backslash \{(\mathsf{id}, k)\}$
        add $m$ to $\mathsf{M}$,
        return $k^{-1}(\mathcal{H}(m) + g^k sk) \bmod q$;
    else, return $\perp$.
If $\mathcal{A}$ queries $DDH(X, Y, Z)$,
    return 1 iff $DH(X, Y) = Z$.
```

We let $\boldsymbol{Adv}_{DSA}^{r-forge}(t, Q)$ be the maximal advantage for any adversary running in time t, making in total $Q = (q_r, q_{DDH})$ queries.

It is currently not known if DSA signatures are still secure in the robust sense; likewise, it is neither known whether DSA can be proven unforgeable in the common sense. For an overview about (limited) security results on DSA and its elliptic curve version see [Vau03]. In particular, it is not known that the additional DDH oracle or the offline/online kind of attack facilitates the task of breaking the signature scheme.

5 Security Analysis of PACE|AA

In this section, we discuss the security of the PACE|AA protocol when active authentication is done via Schnorr signatures; the case of DSA signatures follows, too, because we do not use any specific properties of the underlying signature scheme (except for the robust unforgeability). That is, we assume that the chip, holding public key $X_A = g^{x_A}$ with certificate $cert_C$, signs the message Y_B with key x_A and randomness Y_A. The signature is given by $\sigma = y_A + cx_A \bmod q$ for $c = \mathcal{H}(5||Y_A, T_B)$. After the final authentication step of PACE, the chip sends (using already a secure channel) the values σ and $cert_C$ to the reader who verifies the signatures and the certificate (and aborts in case one of the verification fails).

As noted in [BFK09] using the derived keys already in the key agreement step does not allow for a proof in the Bellare-Pointcheval-Rogaway model. We hence also use the variant that the keys $\mathcal{K}'_{\mathrm{SC}}$ and $\mathcal{K}'_{\mathrm{MAC}}$ are independent from the keys output as the result of the key agreement.

5.1 Security as a Key Exchange Protocol

Theorem 5.1 *The protocol PACE|AA (with Schnorr or DSA signatures) satisfies:*

$$\boldsymbol{Adv}_{PACE|AA}^{ake}(t, Q) \quad \leq \quad \frac{q_e^2}{2q} + \boldsymbol{Adv}_{SC}^{lor}(t^*, q_e, q_e) + \boldsymbol{Adv}_{PACE}^{ake}(t^*, Q)$$

where $t^* = t + O(kq_e^2 + kq_h^2 + kq_c^2 + k^2)$ and $Q = (q_e, q_c, q_h)$.

We remark that the time t^* covers the additional time to maintain lists and perform look-ups. Since PACE is secure (under cryptographic assumptions) it follows together with the security of the underlying encryption scheme that the PACE|AA scheme is secure as well.

The idea of the proof is roughly that the additional Schnorr signature does not violate the security of the underlying PACE protocol as it is encrypted. This is shown through a reduction to the security of the original PACE protocol, mildly exploiting the structure of the original proof in [BFK09] and the properties of the Schnorr signature scheme. We roughly show that, in the PACE|AA protocol, we can simulate the final transmission of the signature token by sending dummy values through the channel, because the keys used to secure this transmission are "as secure as" the PACE keys. That is, even though the strength of the keys is only password-protected (i.e., one can try to guess the low-entropy password), this is sufficient for our purpose, as we do not plan to be more secure than that.

Proof. The proof uses the common game-hopping technique, gradually taking away adversarial success strategies and discussing that each modification cannot contribute significantly to the overall success probability. Note that the original proof of PACE in [BFK09] actually shows something stronger than indistinguishability of keys (from random). The proof rather shows that computing the Diffie-Hellman key $\mathcal{K}$ in an execution is hard (unless one knows or has guessed the password); key indistinguishability then follows from this. We will use this more fine-grained view on the proof below and also consider the adversary on the PACE|AA protocol in this regard, i.e., we measure its success probability according to the probability of making a hash query about $\mathcal{K}$ in a Test session (called target hash query).

Game$_0$: Corresponds to an AKE attack on the PACE|AA protocol (with the more fine-grained success notion).

Game$_1$: Abort GAME$_0$ if an honest chip would compute the same Diffie-Hellman key in two executions.

Note that, since the honest chip always goes second for the Diffie-Hellman key exchange step, sending Y'_A, the keys in such executions are random elements and the probability that such a collision occurs is thus at most $\frac{1}{2}q_e^2/q$.

Game$_2$: Change the previous game slightly such that, an honest chip when sending the encrypted signature, instead picks and uses random and independent (independent of the hash function output) keys $\mathcal{K}'_{\text{SC}}$.

Note that the only difference between the two cases can occur if the adversary makes a target hash query since Reveal and Test sessions never output these keys and Diffie-Hellman keys are distinct by the previous game. It follows that the adversarial success can only decrease by the probability of making a target hash query in this new game.

Game$_3$: Change the game once more and replace channeled transmissions of the signatures sent by an honest chip by encryptions of 0-bits of the same length and, at the same time, let any honest terminal reject any final message unless it has really been sent by the honest chip in the same session.

Note that the length (of the signature part and the certificate) is known in advance. Note also that the probability of making a target hash query in GAME$_3$ cannot be significantly larger, by the distinguishing advantage of genuine transmissions from all-zero transmissions. To make this claim more formally, assume that we mount an attack on the left-or-right security of the (multi-user) encryption scheme by simulating the entire G_{AME_2} with two exceptions: (1) If an honest chip is supposed to send the signature and certificate, then we simply call the next transmission challenge oracle about the signature part and the certificate and about an all-zero message of the same length. Then the challenge bit of the left-or-right oracle corresponds exactly to the difference between the two games. (2) If the adversary successfully modifies the final transmission of an honest chip and the honest terminal would accept the message, then this would also constitute a security breach of the channel protocol. Hence, if the success probabilities of the adversary dropped significantly, we would get a successful attacker against the secure channel scheme.

The final game can now be easily cast as an attack on the original PACE protocol. That is, if there was a successful attacker in GAME_3 (making a target hash query), then there was a straightforward attacker with the same probability in the original PACE protocol: this attacker would run the GAME_3-adversary and simulate the additional signature steps itself (i.e., creating keys and certificates), inject the values from the PACE protocol (i.e., relay the communication), but send dummy values $0 \dots 0$ through the channel on behalf of honest chips under independent random keys. It follows that the probability of making a target hash query in GAME_3 is also bounded by the PACE security.

Given that no target hash query is made, the advantage in the final game is now bounded from above by the advantage against PACE. Note that the advantage of breaking PACE simultaneously covers both the case of target hash queries and of breaks otherwise (such that we do not need to account for the advantage of target hash queries and then of other breaks, resulting in a factor 2). $\square$

On Forward Security. Note that the PACE|AA protocol inherits the forward security of PACE (when used as authenticated key exchange protocol). That is, even if the adversary knows the password, then executions between honest parties remain protected. Since the security of PACE|AA essentially reduces to the security of PACE any successful attack against the forward security of PACE|AA would yield a successful attack against PACE; the other protocol steps do not violate this property.

5.2 Security against Impersonation

It remains to show that the protocol is IKE-secure. Here, we only rely on the unforgeability of certificates and MACs and the robust unforgeability of the Schnorr/DSA signature scheme.

Theorem 5.2 *For the PACE|AA protocol (with Schnorr or DSA signatures) it holds:*

$$\begin{aligned} &\boldsymbol{Adv}^{ike}_{PACE|AA}(t, Q) \\ &\quad\leq \frac{q_e^2 + q_e q_h}{q} + \boldsymbol{Adv}^{forge}_{CA}(t^*, q_e) + 2q_e \cdot \boldsymbol{Adv}^{forge}_{\mathcal{M}}(t^*, 2q_e, 2q_e) \\ &\qquad + \boldsymbol{Adv}^{r-forge}_{\{Schnorr|DSA\}}(t^*, q_e) \end{aligned}$$

where $t^* = t + O(kq_e^2 + kq_h^2 + k^2)$ *and* $Q = (q_e, q_h)$.

The idea is to show first that the adversary cannot inject its own unregistered key (unless it breaks the unforgeability of the certification authority). Since any successful attack must be then for an uncorrupt party whose secret signing key was not revealed, it follows that the adversary must produce a signature under the (registered) public key of an honest user. Because the session id must be new and is somewhat signed via T_B, it follows that the adversary must forge Schnorr respectively DSA signatures in order to break the IKE property.

Proof. We again proceed in games. We can assume that the adversary (and all reductions below) know all passwords at the outset. For the adversary this can be achieved by issuing $\mathsf{Corrupt.pw}$ passwords in the beginning, and the reductions can choose the passwords themselves.

Game$_0$: Corresponds again to the original attack.

Game$_1$: Abort if an honest reader accepts an unregistered key as valid.

Abort and declare the adversary to lose if it manages to make the honest reader accept an unregistered key in any execution. It follows straightforwardly from the unforgeability of certificates that this can decrease the adversary's success probability only by a negligible term. It is straightforward to make this claim formally by simulating the attack (including the honest players, thus being also able to decrypt the final message with the certificate forgery), and using an external certificate issuing oracles for registering the user's public keys.

Game$_2$: Abort if (possibly distinct) honest chips in two executions derive the same key $\mathcal{K}$.

Note that, once a chip selects Y'_A at random in an execution, an honest or malicious reader has already sent Y'_B. Hence, the key $\mathcal{K}$ is a uniformly random element and the probability that it matches any of the previous i keys, is at most i/q. Summing over all maximal q_e executions shows that the adversary's success probability can only drop by $\frac{1}{2}q_e^2/q$.

Game$_3$: Abort if there are collisions among the Y'_B values of honest readers.

In case there appears the same value Y'_B chosen by (possibly distinct) honest readers in two executions also declare the adversary to lose. By the birthday bound and since there are at most q_e such values, this can only deduct $\frac{1}{2}q_e^2/q$ from the success probability.

Game$_4$: Abort if a malicious reader submits a valid T_B in an execution to the honest chip, and such that neither (A) the same valid T_B appears in an execution with an honest reader for the same session identifier, nor (B) the adversary has made a hash query to the key $\mathcal{K}$ derived by the honest chip in the execution before.

Let $sid = (Y'_A, Y'_B, \mathcal{G})$ be the session identifier in an execution with an honest chip in which the adversary submits a valid T_B without having made a hash query before and such that T_B does not appear in a session with an honest reader for the same session identifier. Call this a target execution and fix it for now. Let $\mathcal{K}$ be the Diffie-Hellman key derived by the chip in this execution.

According to the previous games we can assume that all keys in executions with honest chips are unique, and that there is at most one execution with an honest reader in which the same sid (because the values Y'_B chosen by honest readers are distinct). Consider now all executions between the adversary (as a chip) and honest readers in which the same values $Y'_A, \mathcal{G}$ as in the target execution appear. Since the Y'_B values are unique there is at most one execution with the same key $\mathcal{K}$ and the same pair $(Y'_A, \mathcal{G})$ —and this execution must then carry the same value Y'_B— and thus the same session identifier as the target execution. In other words, for a successful attack in the target execution the adversary must send a valid MAC for an unknown key $\mathcal{K}$, or for a new value $(Y^*_A, \mathcal{G}^*)$ (or both). We can therefore derive a contradiction to the unforgeability of the MAC as follows.

Simulate an attack against the MAC scheme by running the entire PACE|AA protocol and the adversary. Pick at random an execution among the at most q_e ones in advance and follow exactly the description of GAME$_3$, except with the following differences: Compute the key $\mathcal{K}$ in the pre-selected execution as before (abort if the execution aborts before or the party is corrupt) as well as the keys $\mathcal{K}_{\text{ENC}}, \mathcal{K}_{\text{MAC}}, \ldots,$ but not the key $\mathcal{K}'_{\text{MAC}}$. Proceed accordingly if the key appears in another execution and ignore, too, when asked to compute $\mathcal{K}'_{\text{MAC}}$. When it comes to verify or to compute a MAC under this key $\mathcal{K}'_{\text{MAC}}$ in any execution, call the external verification resp. MAC oracle instead. (Stop if a verification request for a new message is accepted.)

For the analysis note that the simulation is perfect if the adversary never makes a hash query for the target execution. Also, if the adversary at some point submits a valid MAC T_B under the key $\mathcal{K}$ in the target execution, then we guess the right execution in which this key appears for the first time with probability $1/q_e$. Given this, we successfully forge a MAC, i.e., submit a new message $(Y'_A, \mathcal{G})$ with a valid MAC to the verification oracle. To see this note that for the key $\mathcal{K}'_{\text{MAC}}$ in question we only compute MACs on behalf of honest readers, but then only for pairs $(Y'_A, \mathcal{G})$ different from the one used by the successful adversary resp. for a new key no MAC has been computed before (as discussed above). It follows that the adversary's loss when proceeding from GAME$_3$ to GAME$_4$ can only be q_e times the probability of forging a MAC.

Game$_5$ Abort if the adversary queries its hash function oracle about a Diffie-Hellman key $\mathcal{K}$ in an execution with the honest chip, before it is determined by the value Y'_A.

Note that, once Y'_B has been sent, the random and independent value Y'_A in such an execution makes the key $\mathcal{K}$ random and thus the probability of the adversary having queried $\mathcal{H}$ about $\mathcal{K}$ before is at most q_h/q times the number q_e of such keys.

Game$_6$ Abort if a malicious reader submits a valid T_B in an execution to the honest chip, but such that T_B has been sent by an honest reader before in a session for a different session identifier.

According to the previous games the adversary, when sending a valid T_B, must query the random oracle about the key before, when the session identifier is distinct. In this case, however, we can apply an information-theoretic argument based on the unforgeability of the MAC. Recall that in executions with an honest chip the key $\mathcal{K}$ is unique. Fix one such execution for the moment and call this the target execution. Let $\mathcal{T}_B$ be the set of (at most q_e) values T_B sent in executions with honest readers. Then, in the target execution with values $Y'_A, Y'_B, \mathcal{G}$, the probability that the (unique) key $\mathcal{K}$ hashes to a key $\mathcal{K}'_{\text{MAC}}$ such that $\mathsf{Vf}(\mathcal{K}'_{\text{MAC}}, T_B, (Y'_A, \mathcal{G})) = 1$ for some $T_B \in \mathcal{T}_B$, is negligible. This can be seen as follows: Since the $\mathcal{K}$ is uniquely determined from the target execution before, the value $\mathcal{K}'_{\text{MAC}}$, when returned to the adversary upon a hash query about $\mathcal{K}$, is random and independent from all other keys. If, by chance, $\mathsf{Vf}(\mathcal{K}'_{\text{MAC}}, T_B, (Y'_A, \mathcal{G})) = 1$ for a fixed $T_B \in \mathcal{T}_B$, then we can easily devise a forgery against the MAC scheme as follows. Mount an attack against the MAC scheme by simulating the security game, but record all values in $\mathcal{T}_B$ and make a verification query about $T_B, (Y'_A, \mathcal{G})$ for all values $T_B \in \mathcal{T}_B$ and the values $Y'_A, \mathcal{G}$ after having sent Y'_A in an execution on behalf of an honest chip. Note that the adversary cannot have made a hash query about this key $\mathcal{K}$ before, according to the previous game. Hence, it follows that the key $\mathcal{K}'_{\text{MAC}}$ is still an undetermined random value and the probability that a verification query in the simulation succeeds is exactly equal to the probability in the attack on the MAC (for an unknown random key).

Game$_7$ Abort if the adversary sends a valid signature on behalf of an honest chip for a fresh session.

Abort if the adversary manages to send a valid (encapsulated) signature on behalf of an honest chip to the honest reader (for a session with a fresh *sid* which does not appear in another accepting execution) for the challenge value T_B in this execution. By the previous game the adversary cannot have a value T_B be signed by an honest chip, before sending it in an execution with the honest reader, unless the session identifiers match. In other words, in order to impersonate successfully, the adversary needs to send a valid signature for a new value T_B. This can be straightforwardly turned into an attack against the signature scheme, as discussed next.

Pick at the outset one of the at most q_u active users. Our forger against the signature scheme injects the given public key of the signature scheme as the chosen user's public key. Whenever the adversary invokes a run of this user then we call the token step of the Schnorr signature scheme to get a value Y_A. Similarly, we run the token step in case of DSA signatures but take the output modulo q. We inject this value in the execution and later send a random value Y'_A on behalf of the user. Note that we thus do not know the key $\mathcal{K}$ in this execution, but according to the previous games we can check in executions with the adversary for a candidate among the hash queries via the Decisional Diffie-Hellman Oracle for a candidate and the two values Y'_A, Y'_B from the execution —if we do not find any match we can simply reject. In executions with an honest reader we can actually derive the key from the reader's view on the execution. It follows that we can still compute the right key and then complete the signature token once we have to send the encrypted signature for T_B.

Note that the simulation is perfect from the adversary's view. Hence, if the adversary in Game$_7$ eventually convinces an honest reader to accept a signature for the value T_B in the execution then it follows that this value has not been signed before (or the session identifiers are identical and the adversary cannot win then), we derive a successful forger against the signature scheme.

This concludes the description of the games. Note that in the final game the adversary cannot successfully impersonate anymore. $\square$

6 A Deniable Schnorr Variant

Deniability basically demands that for any (possibly malicious) party on either side, there exists a simulator which produces the same output distribution as the malicious party but without communicating with the honest party. This implies that the malicious party could have generated these data itself, without the help of the other party, and thus cannot use it as a proof towards a third party.

6.1 Defining Deniability

Unlike in the key exchange setting we assume a setting where only one chip and one terminal are present (but may run many executions concurrently), and that the adversary controls either party from the beginning and no further corrupt queries are allowed. We note that the deniability in the multi-user setting immediately follows via a hybrid argument if the parties' secret inputs are otherwise picked independently (as is the case here).

Since we work in the random oracle model, there is a peculiarity due to the (non-)programmability of the hash function [Pas03]. Roughly, it is important that the distinguisher (receiving either the view of the malicious party or the simulated view) cannot distinguish these two random variables, even if it gets access to the same random oracle as the parties and the simulator. The distinguisher's access to the same hash function prevents the simulator from programming the hash values (as it would be the case for a real-world hash function).

Definition 6.1 (Deniability) *A password-based key-exchange protocol P is deniable in the random-oracle model if for any possibly adversary-controlled party with access to the random oracle $\mathcal{H}$ there exists an efficient algorithm $\mathcal{S}^{\mathcal{H}}$ such that $\mathcal{S}^{\mathcal{H}}$ on input the party's secret and public input, as well as the other party's public input, generates the same output distribution as the malicious party in concurrent executions of the protocol. That is, for any algorithm $D^{\mathcal{H}}$ the output of $D^{\mathcal{H}}$ when receiving the public data of both parties and the secret input of the malicious party, in addition to either the output of $\mathcal{S}^{\mathcal{H}}$ or the output of the malicious party, is indistinguishable in both cases. We write $\boldsymbol{Adv}_P^{den}(T, Q)$ for a bound on the difference $\left|\text{Prob}\left[D^{\mathcal{H}}(\mathcal{A}^{\mathcal{H}}) = 1\right] - \text{Prob}\left[D^{\mathcal{H}}(\mathcal{S}^{\mathcal{H}}) = 1\right]\right|$, where $D^{\mathcal{H}}(\mathcal{A}^{\mathcal{H}})$ is the output of D (in time t^* with at most q_h^* hash queries) when run in the experiment with $\mathcal{A}$ (running in time t, invoking at most q_e protocol executions and with at most q_h hash queries); analogously, $D^{\mathcal{H}}(\mathcal{S}^{\mathcal{H}})$ is the output of D when run in the experiment with $\mathcal{S}$ (running in time t' with at most q_h' hash queries). Let $T = (t, t', t^*)$ and $Q = (q_e, q_h, q_h', q_h^*)$.*

Clearly, it wlog. suffices that the distinguisher outputs a bit only. Ideally, the advantage should be small for any efficient D and $\mathcal{A}$ and where the simulator's efficiency characteristics are close to the one of the adversary. We note that there are even stronger versions, called *online* deniability [DKSW09] where the distinguisher can communicate with the malicious party resp. the simulator while the protocol is executed. This notion, however, is much harder to achieve and not known to work here.

6.2 Deniability of Our Protocol

Our deniable version of the Schnorr schemes works as before, only that this time we hash $(Y_A', \mathcal{G})$ instead of T_B. We call this protocol the *deniable Schnorr-based PACE|AA protocol.* Roughly, the idea is now that the chip itself determines the challenge! Given that the challenge can be determined beforehand and that it is created independently of the first signature step one can simulate the final signature part as in the interactive Schnorr identification protocol [Sch91]. We only need to take care that the other security properties are not violated through this.

Note that security as an AKE protocol follows as in the Schnorr signature based version (with the very same bounds). It suffices to show impersonation resistance (which follows similar to the case of signatures) and deniability. We note that our deniability simulator will actually need some assistance in form of a decisional Diffie-Hellman oracle (which, for sake of fairness, we then also give the adversary and the distinguisher). We comment that this does not trivialize the task as such a decision oracle is not known to help compute discrete logarithms, such that the simulator cannot simply derive the chip's secret key from the public key and use this key to show deniability. We note that the query parameters Q thus take additional bounds q_{DDH}, q'_{DDH}, and q^*_{DDH}.

Theorem 6.2 *For the deniable Schnorr-based PACE|AA protocol it holds that:*

$$\begin{aligned}&\boldsymbol{Adv}^{ike}_{PACE|AA}(t,Q)\\ &\leq \quad \frac{2q_e^2+q_eq_h}{q}+\boldsymbol{Adv}^{forge}_{\mathcal{CA}}(t^*,q_e)+2q_e\cdot\boldsymbol{Adv}^{forge}_{\mathcal{M}}(t^*,2q_e,2q_e)+\boldsymbol{Adv}^{r-forge}_{Schnorr}(t^*,q_e)\end{aligned}$$

where $t^*=t+O(kq_e^2+kq_h^2+k^2)$ *and* $Q=(q_e,q_h)$.

Proof. The proof follows almost identically to the one of Theorem 5.2 (impersonation resistance). We start after the hop to Game$_6$ and make another game hop, aborting if in two executions with honest chips, the chips send the same value Y'_A. Since these values are random group elements we only lose a term $\frac{1}{2}q_e^2/q$, analogously to the hop to Game$_3$ in the previous proof. Analogously, we can assume that there are no collisions among the values Y'_A and Y'_B picked by honest chips or terminals, respectively. This decreases the adversary's success probability also by at most $\frac{1}{4q}(2q_e)^2$, since there are at most $\frac{1}{2}(2q_e)^2$ pairs and the probability that there is a collision among the values chosen by chip and terminal, is at most $1/2q$.

Assume now that the adversary at some point successfully impersonates as an honest chip to an honest terminal, by using a pair $(Y'_A,\mathcal{G})$ in the hashing step of our Schnorr version which has been used by an honest chip before. Since the values Y'_A are unique there exists at most one such execution. In this execution the adversary must have sent a different value than Y'_B in the successful impersonation, or else the session identifiers would be identical. It follows that both executions have distinct keys. It furthermore holds that $Y'_B\neq Y'_A$. We can now apply an argument analogously to the one in Game$_4$,Game$_5$, and Game$_6$ that the adversary cannot find a valid token T_A on behalf of the honest chip. Only here we need to make the MAC query about $\mathcal{K}'_{\mathrm{MAC}},(Y'_A,\mathcal{G})$ before to compute T_B on behalf of the honest terminal. But since Y'_A and Y'_B are then both picked by honest users and this implies by assumption that $(Y'_A,\mathcal{G})$ is different from $(Y'_B,\mathcal{G})$ in this execution, the MAC T_B does not help to forge the MAC T_A.

Hence, we can assume that the adversary uses a fresh pair $(Y'_A,\mathcal{G})$, not previously authenticated under the key of an honest chip. This, however, contradicts the unforgeability of the special Schnorr identification version, as discussed after Definition 4.2. □

Theorem 6.3 *The deniable Schnorr-based PACE|AA protocol is deniable in the random oracle model if the MAC is unforgeable (and the adversary, simulator, and distinguisher are granted access to a decision Diffie Hellman oracle). That is, for any malicious chip or terminal $\mathcal{A}$ (with access to the random oracle and a DDH oracle) there exists a simulator $\mathcal{S}$ (with the same oracle access) such that for any distinguisher D (with the same oracle access) we have*

$$\begin{aligned}&\boldsymbol{Adv}^{den}_{PACE|AA}(T,Q)\\ &\leq \quad \frac{q_e^2}{q}+2q_e\cdot\boldsymbol{Adv}^{forge}_{\mathcal{M}}(t',2q_e,2q_e)\end{aligned}$$

where $t'=t+t^*+O(kq_e^2+kq_h^2+k^2)$ *and* $q'_h=q_h+q^*_h$ *and* $q'_{DDH}=q_h^2+q_{DDH}+q^*_{DDH}$.

Proof. It is easy to see that one can simulate the view of a malicious chip easily, by just following the protocol on the terminal's side (since the password is considered a joint secret input it is easy to run the terminal's steps) and mount a black-box simulation of the malicious chip, outputting whatever this party outputs. The output distribution is identical and the simulator makes the same calls to the hash functions as the honest party.

The more interesting case is the one of a malicious terminal. We present our simulator $\mathcal{S}^{\mathcal{H}}$ for this case. Recall that, this time, the simulator only has access to the chip's public key X_A, the group data, and the password (but not the chip's secret key). The simulator now proceeds as follows, running a black-box simulation of the adversarial terminal (playing the honest chip). In each execution the simulator initially picks values $y_A, y'_A \leftarrow \mathbb{Z}_q$ and computes $Y'_A = g^{y'_A}$ as well as $c = \mathcal{H}(Y'_A, \mathcal{G})$ and $Y_A = X_A^{-c} g^{y_A}$. Note that both values are not computed according to the protocol description but still have the same distribution. In particular, even though the simulator may not be able to compute the shared Diffie-Hellman key $\mathcal{K}$ in the execution, it can later complete the signature generation by setting $s\sigma = y_A$ (such that $g^{\sigma} = Y_A X^{\mathcal{H}(Y'_A, \mathcal{G})}$). For the other steps the simulator proceeds as the chip would, using knowledge of the password. Only when the simulator receives T_B from the malicious token, it searches (with the decisional Diffie-Hellman oracle) in the list of hash queries of the malicious terminal for queries about a key $\mathrm{DH}(Y'_A, Y'_B)$. If no key is found then abort this execution; else use the found key $\mathcal{K}$ to finish the execution (using the signature tokens as computed above). If the adversary stops then let the simulator output the same value.

It remains to show that, with overwhelming probability, the malicious terminal cannot send a valid token unless it has queried the random oracle about the Diffie-Hellman key before. This follows as in the proof of Theorem 5.2 (impersonation resistance). There, in game hops from $\textsc{Game}_0$ to $\textsc{Game}_4$ it is shown that any such execution would yield a contradiction against the unforgeability of the MAC (some of the hops, namely to $\textsc{Game}_1$,$\textsc{Game}_3$, and case (A) of $\textsc{Game}_4$, do not apply here because only the chip is honest). Hence, unless the malicious terminal can forge MACs, the simulator will find a (unique) key in the list such that the behavior of the simulator is indistinguishable from the one of the honest party. Additionally, the simulator only queries the hash function as the chip would, allowing us to conclude that the output of D^H is also indistinguishable in both cases. □

Acknowledgments

We thank the anonymous reviewers of FC'12 for helpful comments. This work was supported by the German Federal Office for Information Security (BSI) and CASED (`http://www.cased.de`). The content and views expressed in this article are those of the authors and do not necessarily reflect the position or policy of the supporting authorities.

References

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, January 2005.

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In

Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, August 2010.

[BDF12] Christina Brzuska, Özgür Dagdelen, and Marc Fischlin. Tls, pace, and eac: A cryptographic view at modern key exchange protocols. In *Sicherheit*, pages 71–82, 2012.

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009*, volume 5735 of *LNCS*, pages 33–48. Springer, September 2009.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, December 2001.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, May 2000.

[BPSV08a] Carlo Blundo, Giuseppe Persiano, Ahmad-Reza Sadeghi, and Ivan Visconti. Improved security notions and protocols for non-transferable identification. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 364–378. Springer, October 2008.

[BPSV08b] Carlo Blundo, Giuseppe Persiano, Ahmad-Reza Sadeghi, and Ivan Visconti. Resettable and non-transferable chip authentication for e-passports. In *RFIDSec08*, 2008.

[BSI10] BSI. Advanced security mechanism for machine readable travel documents extended access control (eac). Technical Report (BSI-TR-03110) Version 2.05 Release Candidate, Bundesamt fuer Sicherheit in der Informationstechnik (BSI), 2010.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[DF10] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010*, volume 6531 of *LNCS*, pages 54–68. Springer, October 2010.

[DKSW09] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 146–162. Springer, March 2009.

[Gol04] Oded Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[ICA06] ICAO. Machine readable travel documents. Technical Report Doc 9303, Part 1 Machine Readable Passports, Sixth Edition, International Civil Aviation Organization (ICAO), 2006.

[Ica09] Thomas Icart. How to hash into elliptic curves. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 303–316. Springer, August 2009.

[ICA10] ICAO. Supplemental access control for machine readable travel documents. `http://www2.icao.int/en/MRTD/Pages/Downloads.aspx`, 2010.

[Kra95] D. W. Kravitz. Digital signature algorithm. *Computer Engineering*, 44(5):6–17, 1995.

[MVV07] Jean Monnerat, Serge Vaudenay, and Martin Vuagnoux. About machine-readable travel documents – privacy enhancement using (weakly) non-transferable data authentication. RFIDSEC '07, 2007.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, August 2003.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, August 1990.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Vau03] Serge Vaudenay. The security of DSA and ECDSA. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 309–323. Springer, January 2003.