Security Analysis of the Extended Access Control Protocol for Machine Readable Travel Documents

Özgür Dagdelen[1] and Marc Fischlin[2]

[1] Center for Advanced Security Research Darmstadt - CASED
oezguer.dagdelen@cased.de

[2] Darmstadt University of Technology, Germany
marc.fischlin@gmail.com

Abstract. We analyze the Extended Access Control (EAC) protocol for authenticated key agreement, recently proposed by the German Federal Office for Information Security (BSI) for the deployment in machine readable travel documents. We show that EAC is secure in the Bellare-Rogaway model under the gap Diffie-Hellman (GDH) problem, and assuming random oracles. Furthermore, we discuss that the protocol achieves some of the properties guaranteed by the extended CK security model of LaMacchia, Lauter and Mityagin (ProvSec 2008).

Keywords provable security, authenticated key exchange, German electronic ID card, machine readable travel document.

1 Introduction

Authenticated Key Exchange (AKE) is an important cryptographic primitive to establish a secure key between two parties. It is currently deployed in practical protocols like SSL and TLS, and it will, for instance, also be used in the future German identity cards, and for machine readable travel documents [6].

Security Models for AKE. Bellare and Rogaway [5] were the first to provide a profound model to analyze AKE protocols (BR model). Security according to their notion provides strong guarantees, ensuring that the derived keys remain secure even in presence of active adversaries and multiple concurrent executions. Alternative models have later been suggested to guarantee further desirable security properties.

The most prominent alternatives stem from Canetti and Krawczyk [7], mainly augmenting the BR model by modeling leakage of session states of execution (CK model), and from LaMacchia et al. [14] extending the CK model (eCK model) to include also, for example, forward secrecy and key-compromise impersonation resilience. The former property guarantees that session keys are still protected, even if the adversary later learns long-term secrets like a signature key. The other property ensures that leaking the long-term secret does not help to make the party spuriously believe to talk to a different party. Although seemingly stronger, all these models do not form a strict hierarchy, due to technical details [2,8,9,13].

The Extended Access Control Protocol. The Extended Access Control (EAC) protocol was proposed by the German Federal Office for Information Security (BSI) for the German passports (ePASS) in 2005. It is meant to provide a secure key establishment between a chip card and a terminal, using a public-key infrastructure. The new version of EAC, recommended for the German ID card to be introduced in November 2010, is presented in this paper (with some slight simplifications for the sake of presentation, but without violation of security properties of the overall protocol). EAC serves the purpose to give access control to the sensitive data (e.g., stored finger prints). The BSI has planned to integrate EAC in the German ID card (ePA) to have a complete protection of all recorded personal data.

The EAC protocol consists of two phases: The Terminal Authentication (TA) which is a challenge-response protocol in which the terminal signs a random challenge (and an ephemeral public key) with its certified signing key; and the Chip Authentication (CA) in which both parties derive a Diffie-Hellman key from the terminal's ephemeral key and the chip's static certified key, and where the chip finally computes a message authentication code to authenticate.

We note that the EAC key exchange protocol is one component in the security framework for the identity cards and passports. See Figure 1 for an overview. Another sub protocol is the password authenticated connection establishment (PACE) [3, 6] to ensure a secure key exchange between the card and a reader (which sits in between the card and the terminal). The PACE protocol should be executed first, and the communication between the card and the terminal is then secured through the EAC protocol. We note that the reader and the terminal should also be connected securely through, say, SSL/TLS, before the keys in the EAC protocol between the chip and the terminal are derived. We comment on security issues related to the composition at the end of the paper.

Analyzing the EAC Protocol. We analyze the EAC protocol as an authenticated key exchange protocol in the BR security model. We show that the protocol is secure in this model, assuming that the underlying cryptographic primitives are secure (signatures, certification, and message authentication codes), that the deployed hash function for key derivation behaves as a random oracle, and assuming the gap Diffie-Hellman assumption [4]. The latter assumption says that it is infeasible to solve the *computational* Diffie-Hellman problem even if one has access to a *decisional* Diffie-Hellman oracle. This assumption is for example equivalent to the standard computational DH assumption for pairing-friendly elliptic curves since the *decisional* Diffie-Hellman problem is easy in such groups [11].

Our analysis is in terms of concrete security, identifying exactly how weaknesses of the EAC protocol relate to attacks on the underlying assumptions

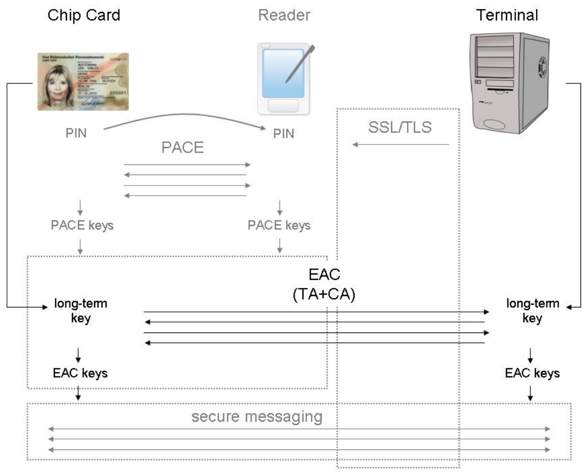

Fig. 1. EAC Protocol for Machine Readable Travel Documents

and primitives. We note that the eCK model is not applicable to show security of the EAC protocol. This is mainly because the chip card does not use ephemeral secrets due to its limited resources, and consequently forward secrecy cannot be achieved without further assumptions (like tamper-resistant hardware). However, we still show that the EAC protocol achieves key-compromise impersonation resilience.

Organization. In Section 2 we define the BR security model (including a registration step). In Section 3 we describe the EAC protocol, and show its security and the underlying assumptions in Section 4. We discuss further security properties in Section 5.

2 Security Model

We analyze the EAC protocol in the real-or-random security model of Bellare and Rogaway [5]. Our notation follows the one in [3] for PACE closely. Some adaptations from the password-based setting to the certified-key scenario are necessary, though.

Attack Model. The model considers a set of honest participants, also called users. Each participant may run several instances of the key agreement protocol, and the j-th instance of a user U is denoted by U_j or (U, j). Each user holds 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 the well-formedness of the keys, e.g., that they belong to an approved group. To obtain a session key the protocol P is executed between two instances of the corresponding users. An instance is called an initiator or client (or resp. respondent or server) if it sends the first (resp. second) message in the protocol. For sake of distinctiveness we often denote the client by A and the server by B.

Upon successful termination we assume that an instance U_i outputs a session key *key*, the session ID *sid*, and a user ID *pid* identifying the intended partner. In the case of the EAC protocol we will assume that the partner identity is determined through the certificates exchanged during the protocol. We note that the session ID usually contains the entire transcript of the communication but, for efficiency reasons, in the EAC protocol it only contains a fraction thereof. We discuss the implications in more detail in Section 3.4.

We consider security against active attacks where the adversary's goal is to distinguish between genuine keys, derived in executions between honest parties, and random keys. This is formalized by allowing a (single) test query in which the adversary either sees the genuine key of the session, or a randomly and independently chosen key (real-or-random). It suffices to consider a single test query only since the case for multiple test queries for many sessions follows by a hybrid argument [1], decreasing the adversary's advantage by a factor equal to the number of test queries.

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. 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*.[3] The adversary can make the following queries to the oracles:

Execute(A, i, B, j) causes the honest 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 honest 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 and the user is not controlled by the adversary. This query 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(U) enables the adversary to obtain the party's long-term key *sk*. This is the so-called *weak-corruption* model. In the *strong-corruption* model the

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

adversary also obtains the state information of all instances of user U. The corrupt queries model a total break of the user and allow to model forward secrecy. Henceforward, user U is considered to be adversarial controlled.

Test(U, i) is initialized with a random bit b. 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 *key*. Then the oracle returns *key* if $b = 0$ or a random key key' from the domain of keys if $b = 1$. If the instance has not terminated yet or has not accepted or the user is adversarial-controlled, 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 that the adversary only makes a single Test query during the attack.

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.

In addition, since we work in the random oracle model, the attacker may also query a random hash function oracle.

We assume that the adversary always knows if an instance has terminated and/or accepted. This seems to be inevitable since the adversary can send further messages to check for the status. We also assume that the adversary learns the session id and the partner id immediately for accepting runs.

Partners, Correctness and Freshness. We say that instances A_i and B_j are *partnered* if both instances have terminated in accepting state with the same output for *sid* and each *pid* identifies the other party as the alleged partner. 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* if U is not controlled by the adversary, there has been no Reveal(U, i) query at any point, neither has there been a Reveal(B, j) query where party B_j is a partner to U_i, nor is (U, i) partnered with an adversarial-controlled party, nor has somebody been corrupted. 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 Corrupt(U)-query there has been no subsequent Test(U, i)-query involving U, or, if so, then there has been no Send(U, i, m)-query for this instance at any point. 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 place before test queries, then executions between honest users are still protected (before or after a Test-query).

AKE Security. The adversary $\mathfrak{A}$ 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 instance (U, i) in the test query is 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 $\mathsf{Execute}$-queries); q_h the number of adversarial queries to the hash oracle. We often write $Q = (q_e, q_h)$ and say that $\mathfrak{A}$ is (t, Q)-bounded.

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

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

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

3 The Extended Access Control (EAC) Protocol

The EAC protocol, or more precisely, the composition of the Terminal Authentication (TA) protocol and the Chip Authentication (CA) protocol, allows mutual authentication between a terminal and a chip and the establishment of an authenticated and encrypted connection. The next subsections present the two main components of the Extended Access Control protocol. Afterwards we define the EAC protocol and comment on deviations in the presentation here compared to the original protocol.

3.1 Protocol Description of the Terminal Authentication

The Terminal Authentication demands a proof of authority by the terminal and thereby allows the chip to check whether the terminal is allowed to access sensitive data. This evidence works with the use of a Public Key Infrastructure (PKI). Terminals are given a certificate specifying the access authority and a signed public key. By a challenge-response step, the terminal renders the chip the permission to read (some of) the chip's data. In this step the terminal signs, together with a nonce sent by the chip, its compressed ephemeral key, which is used in the following Chip Authentication protocol. This step, therefore, somewhat "connects" the TA and the CA phase.

From a cryptographic point of view the TA protocol requires a compression function $\mathsf{Compr} : \{0,1\}^* \to \{0,1\}^\lambda$, and a secure signature scheme, consisting of three efficient algorithms $\mathcal{S} = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{SVf})$ to generate a key pair $(sk, pk) \leftarrow \mathsf{SKGen}(1^\lambda)$ such that $s \leftarrow \mathsf{Sig}(sk, m)$ allows to sign arbitrary messages m with the secret key sk, and such that the signatures can subsequently be verified via the verification algorithm and the public key pk, returning a bit $d \leftarrow \mathsf{SVf}(pk, m, s)$. The scheme should be correct in the sense that for any $(sk, pk) \leftarrow \mathsf{SKGen}(1^\lambda)$, any message m, any $s \leftarrow \mathsf{Sig}(sk, m)$ we always have $\mathsf{SVf}(pk, m, s) = 1$. We will later describe the security requirements for these primitives.

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 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 note that the CA may, as usual, check for correctness of the data it certifies, e.g., verifying well-formedness of certified keys or checking the identity of a key owner; however, for security purposes we allow the adversary to register arbitrary (well-formed) keys.

In the Terminal Authentication protocol the terminal $\mathcal{T}$ and the chip $\mathcal{C}$ perform the following steps:

1. $\mathcal{T}$ sends a certificate chain to $\mathcal{C}$ including its certificate along with the certificates of the Document Verifier (DV) and Country Verifiying CA (CVCA).
2. $\mathcal{C}$ is able to verify the certificate chain of CVCA, DV and the certificate $cert_T$ of the terminal. Then $\mathcal{C}$ extracts $\mathcal{T}$'s public key $pk_{\mathcal{T}}$ from the certificate.[4]
3. $\mathcal{T}$ generates an ephemeral Diffie-Hellman key pair $(esk_{\mathcal{T}}, epk_{\mathcal{T}}, \mathrm{D}_{\mathcal{C}})$ for domain $\mathrm{D}_{\mathcal{C}}$ and sends the compressed ephemeral public key $\mathsf{Compr}(epk_{\mathcal{T}})$ to $\mathcal{C}$.
4. $\mathcal{C}$ randomly chooses a nonce $r_1 \leftarrow \{0,1\}^\lambda$ and sends it to $\mathcal{T}$.
5. $\mathcal{T}$ signs the identifier $\mathrm{ID}_{\mathcal{C}}$ of $\mathcal{C}$ along with the nonce r_1 and the compressed public key $\mathsf{Compr}(epk_{\mathcal{T}})$, i.e.

$$s = \mathsf{Sig}(sk_{\mathcal{T}}, (\mathrm{ID}_{\mathcal{C}}, r_1, \mathsf{Compr}(epk_{\mathcal{T}}))).$$

   The signature s is sent to $\mathcal{C}$ by $\mathcal{T}$.
6. $\mathcal{C}$ checks if $\mathsf{Vf}_{\mathsf{Sig}}(pk_{\mathcal{T}}, s, m) = 1$ for $m = (\mathrm{ID}_{\mathcal{C}}, r_1, \mathsf{Compr}(epk_{\mathcal{T}}))$, using the static public key $pk_{\mathcal{T}}$ of the terminal.

Remarks. We do not consider auxiliary data sent by the terminal as specified in [6], since it does not offer any additional security for the key exchange. The delivery of auxiliary data is insignificant and omitting these data from the description above facilitates the analysis and understanding of the TA protocol.

The static domain parameter $\mathrm{D}_{\mathcal{C}}$ contains the (certified) group description for which the chip and terminal execute the Diffie-Hellman computations. The identifier of the chip $\mathrm{ID}_{\mathcal{C}}$ is defined by the compressed ephemeral public key $\mathsf{Compr}(epk_{\mathcal{C}})$ used in the PACE protocol before the Terminal Authentication is invoked. Thus, one establishes a link between the PACE protocol and Terminal Authentication, but decoupling of protocols like PACE and EAC again eases

[4] For sake of simplicity, we will sometimes use $cert_T$ and mean therewith the whole certificate chain including the certificates of CVCA and DV.

the analysis of EAC. The composition of the protocols does not lead to any significant advantage in terms of the BR model, since active adversaries are potentially able to control the card reader and act genuinely for the PACE and SSL/TLS protocol executions.

Further we assume that the parties abort an execution whenever they receive an unexpected message including either wrong format or false sequence of messages. This holds also for the following protocols.

3.2 Protocol Description of the Chip Authentication

The Chip Authentication protocol provides an authenticity check of chips, as well as a secure session key for encryption and integrity of subsequently transmitted messages. Unlike TA, there is no challenge-response action. Instead, the chip computes the Diffie-Hellman value with its static key and the ephemeral key chosen by the terminal, and both parties then hash this value together with a random nonce. Thereby, the chip obtains the session key for encryption and authentication (and an extra key for authentication in the key confirmation phase). Now, the chip computes a message authentication code over the ephemeral public key of the terminal, using the additional authentication key as the secret, and sends this authentication token to the terminal. The terminal can verify the authenticity by checking the validity of the token with the newly derived key.

In this step we need a message authentication code which, similar to signature schemes, is modeled by a tuple $\mathcal{M} = (\mathsf{MKGen}, \mathsf{MAC}, \mathsf{MVf})$ of efficient algorithms and works like a signature scheme, except that *pk* equals the secret key *sk* and is also kept secret. We also let $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ denote the hash functions modeled as random oracles. For implementations we assume that we are given a random oracle $\mathcal{H}$ and then set $\mathcal{H}_i(\cdot) = \mathcal{H}(\langle i \rangle \,||\cdot)$ for some fixed-length encoding $\langle i \rangle$ of $i = 1, 2, 3$.

In the Chip Authentication protocol the terminal $\mathcal{T}$ and the chip $\mathcal{C}$ perform the following steps:

1. $\mathcal{C}$ sends its static public key $pk_{\mathcal{C}}$, and the domain parameters $\mathrm{D}_{\mathcal{C}}$ to $\mathcal{T}$, together with a certificate for $pk_{\mathcal{C}}$.[5]
2. After $\mathcal{T}$ has checked the validity of $pk_{\mathcal{C}}$, $\mathcal{T}$ sends its ephemeral public key $epk_{\mathcal{T}}$ to $\mathcal{C}$.
3. $\mathcal{C}$ applies the compression function Compr to the received ephemeral public key by $\mathcal{T}$ and compares this to the compressed public key received during the Terminal Authentication execution.
4. Both $\mathcal{C}$ and $\mathcal{T}$ compute the shared key as $\mathcal{K} = \mathrm{DH}(epk_{\mathcal{T}}, sk_{\mathcal{C}})$ resp. $\mathcal{K} = \mathrm{DH}(pk_{\mathcal{C}}, esk_{\mathcal{T}})$.[6]

[5] Formally, the chip card sends further data which are irrelevant for the security of EAC as an AKE protocol.

[6] Here, $\mathrm{DH}(g^a, b) = g^{ab}$ for group element g^a and exponent b. In the following we overload the function and occasionally also write $\mathrm{DH}(g^a, g^b) = g^{ab}$, where g is clear from the context.

5. $\mathcal{C}$ picks a random $r_2 \leftarrow \{0,1\}^\lambda$ and derives session keys computing

$$\mathcal{K}_{\mathrm{ENC}} = \mathcal{H}_1(\mathcal{K}, r_2), \quad \mathcal{K}_{\mathrm{MAC}} = \mathcal{H}_2(\mathcal{K}, r_2), \quad \mathcal{K}'_{\mathrm{MAC}} = \mathcal{H}_3(\mathcal{K}, r_2)$$

   where we assume that $\mathcal{H}_3$ generates the same distribution on keys as $\mathsf{MKGen}(1^\lambda)$. Afterwards $\mathcal{C}$ prepares an authentication token $\mathrm{T} = \mathsf{MAC}(\mathcal{K}'_{\mathrm{MAC}}, (epk_{\mathcal{T}}, \mathrm{D}_{\mathcal{C}}))$ and sends it along with r_2 to $\mathcal{T}$.
6. After receiving r_2 the terminal $\mathcal{T}$ is able to derive the session keys as well by computing the secret keys

$$\mathcal{K}_{\mathrm{ENC}} = \mathcal{H}_1(\mathcal{K}, r_2), \quad \mathcal{K}_{\mathrm{MAC}} = \mathcal{H}_2(\mathcal{K}, r_2), \quad \mathcal{K}'_{\mathrm{MAC}} = \mathcal{H}_3(\mathcal{K}, r_2).$$

   Next the validity of authentication token T is checked by $\mathcal{T}$ with $\mathcal{K}'_{\mathrm{MAC}}$.

3.3 Protocol Description of the Extended Access Control

After the introduction of the Terminal Authentication and Chip Authentication protocols, we define the Extended Access Control protocol. See Figure 2 for an overview. When viewed as an authenticated key exchange protocol the parties output $\mathcal{K}_{\mathrm{ENC}}, \mathcal{K}_{\mathrm{MAC}}$ as the session key(s), the session id consists of the authenticated values $(epk_{\mathcal{T}}, pk_{\mathcal{C}}, r_2, \mathrm{D}_{\mathcal{C}})$ in the final messages, and the partner id is assumed to be empty (see below for remarks).

We remark that in this paper we place a scope on the security analysis of EAC. Performance analysis and design choices of the protocol are not investigated here.

3.4 Remarks

Some remarks about the changes compared to the original protocol in [6] and about underlying assumptions are in order.

Session and Partner IDs. We substitute the common definition of session IDs which include the whole transcript in *sid* by a "more loose" version. In order to spare the parties from storing the transcript data in the execution —see [10] for solutions to this problem— we define the session IDs by the ephemeral public key of the terminal, the chip's static key, and the nonce r_2 chosen by the chip (and the domain $\mathrm{D}_{\mathcal{C}}$). This loose partnering approach here may allow an adversary now to run a man-in-the-middle attack making the honest parties assume they communicate with someone else, even though they hold the same key.

We note that the terminal identifies the chip as a partner in *pid* (i.e., its public key), whereas the chip outputs an empty partner ID. The latter is necessary if the adversary can register adversarial-controlled terminals (as is the case in our model), because such a terminal could basically act in a man-in-the-middle attack substituting the honest terminal's data in the TA phase by its own. If the adversary cannot register terminals then the chip can output the certified public-key $pk_{\mathcal{T}}$ as the reliable partner ID.

Chip :	TERMINAL AUTHENTICATION	**Terminal** :
$pk_{\mathcal{CA}}, \mathrm{ID}_{\mathcal{C}}, sk_{\mathcal{C}}, pk_{\mathcal{C}}, cert_{C}$		$pk_{\mathcal{CA}}, cert_{T}, \mathrm{ID}_{\mathcal{C}}, sk_{\mathcal{T}}$
	$\xleftarrow{\quad cert_{T} \quad}$	
If $\mathsf{CVf}(pk_{\mathcal{CA}}, cert_{T}) = 0$, abort		
extract $pk_{\mathcal{T}}$ (from $cert_{T}$)		
		generate $(esk_{\mathcal{T}}, epk_{\mathcal{T}}, \mathrm{D}_{\mathcal{C}})$
	$\xleftarrow{\mathsf{Compr}(epk_{\mathcal{T}})}$	
$r_1 \xleftarrow{\$} \{0,1\}^{\lambda}$		
	$\xrightarrow{\quad r_1 \quad}$	
		$s = \mathsf{Sig}(sk_{\mathcal{T}}, (\mathrm{ID}_{\mathcal{C}}, r_1, \mathsf{Compr}(epk_{\mathcal{T}})))$
	$\xleftarrow{\quad s \quad}$	
If $\mathsf{SVf}(pk_{\mathcal{T}}, s, (\mathrm{ID}_{\mathcal{C}}, r_1, \mathsf{Compr}(epk_{\mathcal{T}}))) = 0$, abort		
Chip :	CHIP AUTHENTICATION	**Terminal** :
	$\xrightarrow{cert_{C}, pk_{\mathcal{C}}, \mathrm{D}_{\mathcal{C}}}$	If $\mathsf{CVf}(pk_{\mathcal{CA}}, cert_{C}) = 0$, abort
	$\xleftarrow{\quad epk_{\mathcal{T}} \quad}$	
verify that $epk_{\mathcal{T}}$ matches $\mathsf{Compr}(epk_{\mathcal{T}})$		
$\mathcal{K} = \mathrm{DH}(epk_{\mathcal{T}}, sk_{\mathcal{C}})$		$\mathcal{K} = \mathrm{DH}(pk_{\mathcal{C}}, esk_{\mathcal{T}})$
$r_2 \xleftarrow{\$} \{0,1\}^{\lambda}$		
$\mathcal{K}_{\mathrm{ENC}} = \mathcal{H}_1(\mathcal{K}, r_2)$		
$\mathcal{K}_{\mathrm{MAC}} = \mathcal{H}_2(\mathcal{K}, r_2)$		
$\mathcal{K}'_{\mathrm{MAC}} = \mathcal{H}_3(\mathcal{K}, r_2)$		
$\mathrm{T} = \mathsf{MAC}(\mathcal{K}'_{\mathrm{MAC}}, (epk_{\mathcal{T}}, \mathrm{D}_{\mathcal{C}}))$		
	$\xrightarrow{\quad \mathrm{T}, r_2 \quad}$	
		$\mathcal{K}_{\mathrm{ENC}} = \mathcal{H}_1(\mathcal{K}, r_2)$
		$\mathcal{K}_{\mathrm{MAC}} = \mathcal{H}_2(\mathcal{K}, r_2)$
		$\mathcal{K}'_{\mathrm{MAC}} = \mathcal{H}_3(\mathcal{K}, r_2)$
		If $\mathsf{MVf}(\mathcal{K}'_{\mathrm{MAC}}, \mathrm{T}, (epk_{\mathcal{T}}, \mathrm{D}_{\mathcal{C}})) = 0$, abort
key$=(\mathcal{K}_{\mathrm{ENC}}, \mathcal{K}_{\mathrm{MAC}})$		key$=(\mathcal{K}_{\mathrm{ENC}}, \mathcal{K}_{\mathrm{MAC}})$
$sid = (epk_{\mathcal{T}}, pk_{\mathcal{C}}, \mathrm{D}_{\mathcal{C}}, r_2)$		$sid = (epk_{\mathcal{T}}, pk_{\mathcal{C}}, \mathrm{D}_{\mathcal{C}}, r_2)$
$pid = \emptyset$		$pid = pk_{\mathcal{C}}$

Fig. 2. Extended Access Control (EAC) protocol

The final authentication step. The original scheme uses the output key $\mathcal{K}_{\mathrm{MAC}}$ for the MAC computations (token) in the key-agreement protocol, too. This version, however, may not be provable secure in our security model. The reason is that with the Test query the adversary obtains a random or the genuine secret key, including $\mathcal{K}_{\mathrm{MAC}}$. Then the adversary can possibly test whether this key part $\mathcal{K}_{\mathrm{MAC}}$ together with $epk_{\mathcal{T}}$ matches the transmitted value. For the general analysis, we therefore suggest to derive an ephemeral MAC key $\mathcal{K}'_{\mathrm{MAC}}$ as $\mathcal{K}'_{\mathrm{MAC}} = \mathcal{H}_3(\mathcal{K}, r_2)$ and use this key for authentication. A similar strategy is used in the formal analysis of PACE [3].

Different versions of EAC protocol. In [6] the German Federal Office for Information Security (BSI) proposes two versions of EAC. Both versions give implicit authentication of the data stored on the chip. The second version provides additionally explicit authentication of the chip. This is realized by the authentication token in Step 5 and 6 in the Chip Authentication. We present and analyze the second version of EAC since it allows the composition order executing the Chip Authentication protocol at the end, and this version will be implemented in the electronic German ID card in November 2010.

Encryption by PACE. Basically the whole communication between the chip and the terminal is secured by the session key obtained in the in advance executed PACE protocol instance and presumably SSL/TLS. PACE is responsible for a secure communication between card reader and the chip. The communication security between card reader and the terminal should be assured by SSL/TLS. For our analysis, we do not even use these additional provisions of security means; the EAC protocol alone is already strong enough.

Passive Authentication. In [6] the German Federal Office for Information Security (BSI) recommends to insert a Passive Authentication step between the Terminal Authentication and the Chip Authentication. In this step the chip's certificate (data security object) issued by the document signer is transmitted to the terminal to verify the authenticity of the chip. However, Passive Authentication cannot detect cloning. Therefore, the execution of the Chip Authentication protocol is necessary. To simulate the original workflow of EAC, we substitute the Passive Authentication step by adding a certificate while the chip sends his static public key. This abstracts out the essential part for AKE security.

4 Security of the EAC Protocol

In this section we state the underlying security assumptions and prove EAC to be secure.

4.1 Security Assumptions

As remarked above we carry out our security analysis assuming an ideal hash function (random oracle model). Basically, it says that the three hash functions $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ act like random functions to which all parties have access.

For the compression function Compr we assume either that this function is injective (errorless compression), or at least second-preimage resistant. In any case, it should be hard to find another preimage to a given random image mapping to the same value. For instance, this property is fulfilled by injective and collision-resistant functions, even though second-preimage resistance imposes a weaker condition on the function than collision-resistance. We discuss in the full version that even this requirement can be weakened even further provided that collisions are "appropriately intertwined" with the Diffie-Hellman problem. For example, a projection to the x-coordinate of the public key in case of elliptic curves, suggested in [6], remains secure as well.

Definition 1 (Second-Preimage-Resistance). *For a function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ let denote the probability that algorithm $\mathfrak{A}$ given input $m \leftarrow \mathcal{D}$ (drawn according to distribution $\mathcal{D}$) outputs m' such that $m \neq m'$ and $\mathcal{H}(m) = \mathcal{H}(m')$.*

We usually demand that $\mathbf{Adv}_{\mathcal{H},\mathcal{D}}^{\mathrm{SecPre}}(\mathfrak{A})$ is negligible, and call the function then second-preimage resistant.

For the signature scheme we need unforgeability which says that no outsider should be able to forge the signers signature. More formally, a signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{SVf})$ is (t, Q_s, ϵ)-unforgeable if for any algorithm $\mathfrak{A}$ running in time t the probability that $\mathfrak{A}$ outputs a signature to a fresh message on behalf of an arbitrary public key is $\mathbf{Adv}_{\mathcal{S}}^{\mathrm{forge}}(t, Q_s)$ (which should be negligible small) while $\mathfrak{A}$ has access (at most Q_s times) to a singing oracle. That is, the adversary may ask for signatures of chosen messages.

We define unforgeability for a certification scheme analogously, and denote the advantage bound of outputting a certificate of a new value in time t after seeing Q_c certificates by $\mathbf{Adv}_{\mathcal{CA}}^{\mathrm{forge}}(t, Q_c)$. We assume that the Certification authority only issues unique certificates in the sense that for distinct parties the certificates are also distinct; we also assume that the authority checks that the keys are well-formed group elements. As for MACs, we denote by $\mathbf{Adv}_{\mathcal{M}}^{\mathrm{forge}}(t, Q_m, Q_v)$ the bound on the advantage of finding a valid MAC for a new message after making Q_m queries to MAC and Q_v queries to a verification oracle (which is necessary since verification also uses the secret key).

In addition, we need some known number-theoretic assumptions which we briefly introduce next. To prevent (mainly) passive adversaries, merely eavesdropping the network, we need the common computational Diffie-Hellman assumption. To show security against active adversaries, possibly contributing to the DH steps in the Chip Authentication protocol, we need a stronger assumption, denoted gap Diffie-Hellman problem [4]. This assumption basically says that solving the computational DH problem for (g^a, g^b) 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.

4.2 Security Proof

This section gives a security analysis for the Extended Access Control (EAC) protocol.

Theorem 1. *In the random oracle model we have*

$$\begin{aligned}
&\boldsymbol{Adv}_{EAC}^{AKE}(t, Q) \\
&\quad \le q_e \cdot \boldsymbol{Adv}_{\mathsf{Compr},\mathcal{KG}}^{SecPre}(t) + \binom{q_e}{2} \cdot \left(\tfrac{1}{q} + 2^{-\lambda+1}\right) + q_e \cdot \boldsymbol{Adv}_{\mathcal{M}}^{forge}(t + O(\lambda), 1, q_e) \\
&\qquad + q_e \cdot \boldsymbol{Adv}_{\mathcal{S}}^{forge}(t + O(\lambda \cdot q_e \log q_e), q_e) + \boldsymbol{Adv}_{\mathcal{CA}}^{forge}(t + O(\lambda), q_e) \\
&\qquad + 2q_e^2 \cdot \boldsymbol{Adv}^{GDH}(t + O(\lambda \cdot q_e q_h \log q_e q_h), (2q_e + 1)(q_h + q_e))
\end{aligned}$$

where λ denotes the security parameter (and bit length of nonces), $q = q(k)$ the group order, $Q = (q_e, q_h)$ the number of executions and hash computations, respectively, and $\mathcal{KG}$ the algorithm which generates a ephemeral public key for the terminal.

Proof. The proof of correctness, that untampered executions between honest parties yield the same accepting output, is straightforward from the correctness of the underlying primitives. Therefore, it remains to show the AKE security property.

We show security via the common game based approach, gradually changing the original attack $\textsc{Game}_0$ (with random test bit b) via experiments $\textsc{Game}_1$; $\textsc{Game}_2; \ldots$ to a game where the adversary's success probability to predict b is bounded by the guessing probability of $\frac{1}{2}$. Each transition from $\textsc{Game}_i$ to $\textsc{Game}_{i+1}$ will change the adversary's probability only slightly (depending on cryptographic assumptions), thus showing that the success probability in the original attack cannot be significantly larger than $\frac{1}{2}$. (Formally, we can condition on all "bad" events ruled out in the previous games not to happen.)

Due to space limitations, here we provide a proof sketch given in Figure 3 listing the modifications in each game. For a complete and comprehensive proof, we refer to our full version of this paper.

Games	**Probability loss**	**Description/Restriction**	**Reduction to**
$\textsc{Game}_0$	—	original attack on the EAC protocol	—
$\textsc{Game}_1$	$q_e \cdot \mathbf{Adv}^{\mathrm{SecPre}}_{\mathsf{Compr},\mathcal{KG}}(t)$	no collision in the compression function Compr	Second-Preimage Resistance of Compr
$\textsc{Game}_2$	$\binom{q_e}{2} \cdot \frac{1}{q}$	no collisions among $(epk_{\mathcal{T}}, r_1)$ values chosen (resp. received) by honest terminals	Birthday paradox
$\textsc{Game}_3$	$\binom{q_e}{2} \cdot 2^{-\lambda}$	no collisions among (h, r_1) values received (resp. chosen) by honest chip cards	Birthday paradox
$\textsc{Game}_4$	$q_e \cdot \mathbf{Adv}^{\mathrm{forge}}_{\mathcal{S}}(t + O(\lambda \cdot q_e \log q_e), q_e)$	abort if there exists an adversary $\mathfrak{A}$ who is able to forge a valid signature on behalf of an honest terminal	Unforgeability of the signature scheme
$\textsc{Game}_5$	$\mathbf{Adv}^{\mathrm{forge}}_{\mathcal{CA}}(t + O(\lambda), q_e)$	abort if the adversary in some execution submits a new long-term public key with a valid certificate such that this key has not been registered with the authority before	Unforgeability of the $\mathcal{CA}$ scheme
$\textsc{Game}_6$	$q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}(t + O(\lambda \cdot q_e q_h \log q_e q_h), (2q_e + 1)(q_h + q_e))$	no adversary makes a hash query for a Diffie-Hellman key computed by an honest chip, allegedly in an execution with an honest terminal	GDH problem
$\textsc{Game}_7$	$q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}(t + O(\lambda \cdot q_e q_h \log q_e q_h), (2q_e + 1)(q_e + q_h))$	no adversary makes a hash query for a Diffie-Hellman key computed by an honest terminal, allegedly in an execution with an honest chip	GDH problem
$\textsc{Game}_8$	$\binom{q_e}{2} \cdot 2^{-\lambda}$	no r_2 collisions for honest chip cards	Birthday Paradox
$\textsc{Game}_9$	$q_e \cdot \mathbf{Adv}^{\mathrm{forge}}_{\mathcal{M}}(t + O(\lambda), 1, q_e)$	abort if there are two honest parties with both accepting sessions yielding (K; r2) but which are not partnered	Unforgeability of the MAC scheme

Fig. 3. Overview of the games within the proof

5 Discussion

In this section we put the security guarantees provided by the BR model into perspective of the eCK security model of LaMacchia et al. [14], which in turn extends the model of Canetti and Krawczyk [7]. Among others we analyze the protocol in terms of forward secrecy and key-compromise impersonation. We also briefly discuss questions related to the composition of the security protocols for identity cards. Due to lack of space, the full discussion including the analysis is provided in the full version. However, here we itemize the results.

Forward Secrecy. It is obvious that, whenever the card discloses its long term secret key, or the terminal its ephemeral key, the adversary can deduce easily the session key. On the other hand, assuming that the hardware of the ePA is leakage-resistant, and that the terminal immediately and securely deletes ephemeral keys when no longer required and that these keys cannot be leaked meanwhile, previous sessions cannot be attacked anymore.
We also remark that, if the adversary only knows the terminal's long-term secret key, but lacks knowledge of the terminal's ephemeral key and of the card's long-term secret, then the session key of an execution between honest parties is still secret.

Leakage-Resilience. The card only uses internal randomness r_1, r_2 which is later sent in public. Leaking these information does not harm to the security of the protocol, even if leaked in advance. Leaking the terminal's ephemeral key, however, does breach security. However, it is recommended to hedge terminals against leakage of ephemeral keys, whereby this risk should be minimized.

Leakage of Secrets. Due to the asynchronous structure of the protocol, an adversary who gets hold of the (long-term) secret key of either the terminal or the chip card, is unable to make the respective party accept in an attempt to stand in for the party in the different role. First, the chip's long-term secret is of no avail to sign on behalf of a valid terminal, and, second, by using the long-term secret of a terminal, one is still unable to derive the Diffie-Hellman key (session key).

Acknowledgments

We thank the anonymous reviewers of ISC'10 for helpful comments.

This work was supported by the German Federal Office for Information Security (BSI) and CASED (`http://www.cased.de`). Marc Fischlin is also supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG). 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

1. Michel Abdalla, Pierre alain Fouque, and David Pointcheval. *Password-based authenticated key exchange in the three-party setting.* PKC 2005: 8th International

Workshop on Theory and Practice in Public Key Cryptography, Lecture Notes in Computer Science, pages 65–84. Springer-Verlag, 2005.

2. Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. *Efficient One-Round Key Exchange in the Standard Model*. ACISP '08: Proceedings of the 13th Australasian conference on Information Security and Privacy, pages 69–83. Springer-Verlag, 2008.
3. Jens Bender, Marc Fischlin, and Dennis Kuegler. *Security Analysis of the PACE Key-Agreement Protocol*. Information Security Conference (ISC) 2009, Volume 5735 of Lecture Notes in Computer Science, pages 33–48. Springer-Verlag, 2009.
4. Dan Boneh, Ben Lynn, and Hovav Shacham. *Short Signatures from the Weil Pairing*. Advances in Cryptology — Asiacrypt 2001, Volume 2248 of Lecture Notes in Computer Science, pages 514–532. Springer-Verlag, 2001.
5. Mihir Bellare and Phillip Rogaway. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. Proceedings of the Annual Conference on Computer and Communications Security (CCS). ACM Press, 1993.
6. *Advanced Security Mechanism for Machine Readable Travel Documents Extended Access Control (EAC)*. Technical Report (BSI-TR-03110) Version 2.02 Release Candidate, Bundesamt fuer Sicherheit in der Informationstechnik (BSI), 2008.
7. Ran Canetti and Hugo Krawczyk. *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*. Advances in Cryptology — Eurocrypt 2001, Lecture Notes in Computer Science, pages 453–474. Springer-Verlag, 2001.
8. Cas J.F. Cremers. *Formally and Practically Relating the CK, CK-HMQV, and eCK Security Models for Authenticated Key Exchange*. Number 2009/253 in Cryptology eprint archive. `eprint.iacr.org`, 2009.
9. Cas J.F. Cremers. *Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol*. ACNS '09: Proceedings of the 7th International Conference on Applied Cryptography and Network Security, pages 20–33. Springer-Verlag, 2009.
10. Marc Fischlin and Anja Lehmann. *Delayed-Key Message Authentication for Streams*. Theory of Cryptography Conference (TCC), Volume 5978 of Lecture Notes in Computer Science. Springer-Verlag, 2010.
11. Antoine Joux and Kim Nguyen. *Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups*. *J. Cryptology*, 16(4):239–247, 2003.
12. Mike Just and Serge Vaudenay. *Authenticated Multi-Party Key Agreement*. Advances in Cryptology — Asiacrypt 1996, pages 36–49. Springer-Verlag, 1996.
13. Kim kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. *Examining Indistinguishability-Based Proof Models for Key Establishment Protocols*. Advances in Cryptology — Asiacrypt 2005, pages 585–604. Springer-Verlag, 2005.
14. Brian LaMacchia, Kristin Lauter, and Anton Mityagin. *Stronger Security of Authenticated Key Exchange*. Number 2006/073 in Cryptology eprint archive. `eprint.iacr.org`, 2006.