A preliminary version appears in WISTP 2016, Lecture Notes in Computer Science, Springer-Verlag, 2016. This version is dated July 7, 2016.

Securing Transactions with the eIDAS Protocols

Frank Morgner[1] Paul Bastian[1] Marc Fischlin[2]

[1]Bundesdruckerei GmbH
[2]Technische Universität Darmstadt, Germany

Abstract. The proposed European system for electronic identities, authentication, and trust services (eIDAS) enables remote authentication of an identity card (and selected data of the card) to an eID service. The core system has already been running on the German identity card since 2010. We analyze an extension proposed by Bundesdruckerei that enables the protocol to authenticate further transaction data such as phone numbers or PGP keys. In particular we prove cryptographically that the extension provides strong authenticity guarantees. We also discuss privacy aspects of the solution, preventing the card and the service provider of the eIDAS system to learn the actual transaction data.

1 Introduction

With Regulation EU No 910/2014 about electronic identification, authentication, and trust services for electronic transactions (eIDAS) in 2014, the European parliament has paved the way for a common electronic identity system for Europe. Driven by German and French IT security offices, namely BSI and ANSSI, the first technical proposal for such eIDAS tokens has been put forward in [2] in February 2015. The proposal extends a previous specification for the new German identity cards [1], and as the cards have been issued since November 2010, this means that the basic eIDAS system proposal is already effectively running at this point.

1.1 The basic eIDAS system

The system of the German identity card adopted early the idea to use the identity card securely for internet services. The basic steps of the protocol are outlined in Figure 1. The identity card is securely connected to a local card reader at the user's computer via the password-authenticated connection establishment (PACE) protocol. The reader itself is equipped with a small tamper-proof display and is connected (through an application interface on the user's computer) with a service running the eID server. The connection from the computer to the server may be secured through TLS.

The ID card and the eID server then execute the extended access control (EAC) protocol which consists of a terminal authentication (TA) and a chip authentication (CA). In the TA step the eID server authenticates and transmits the credentials for accessing some fields on the ID card, such as for age verification or the address. The human user confirms access to the fields via the reader, by verifying the request on the reader's display. Then the card authenticates itself and the requested data through the CA step.

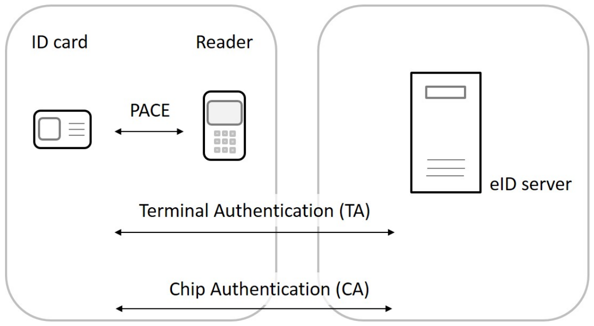


Figure 1: Extended Access Control (EAC) for online services, consisting of the terminal authentication (TA) step and the chip authentication (CA) step.

1.2 Securing transactions through the eID system

Based on a proposal by Bundesdruckerei [3], we discuss how to use the German identity card system (and consequently, the anticipated European eIDAS system) to secure further transaction data. As an example assume that the eID system is used in an online banking service, and that the bank would also like to authenticate the mobile phone number of the user (e.g., for establishing a reliable communication channel for authentication of bank transactions). This phone number is not present on the identity card and thus cannot, per se, be authenticated through the eID system.

The idea of building a transaction system on top of the eID system is to use the auxiliary-data field, originally provisioned for refinements in the data authentication (e.g., the eID server has to commit to the current date for age verification). In order to show that one can securely use this entry for authenticating data like the phone number one needs to show that (a) the solution really validates the transaction, and that (b) it is indeed possible to smoothly extend the existing system to incorporate such checks. The feasibility has already been reported in [3], and we briefly revisit the details at the end of the work, such that it remains to show that the scheme is indeed secure.

We therefore first present a cryptographic description of the transaction system of Bundesdruckerei, called eIDAS transaction system here. The idea is depicted in Figure 2 and consists of the following steps. (1) The card holder and the service provider first agree upon the transaction T. (2) Then the service provider initiates the eID server for a hash value $\mathsf{H}(\mathsf{T})$ of the transaction. (3) The card holder also forwards the transaction in clear to the reader. (4) The original eID components run the TA step, including $\mathsf{H}(\mathsf{T})$ as auxiliary data.

(5) The reader verifies T against H(T), and the user verifies the transaction via the reader's display. (6) To approve the transaction the user initiates chip authentication. (7) The eID server reports back approval of the transaction if chip authentication terminates successfully.

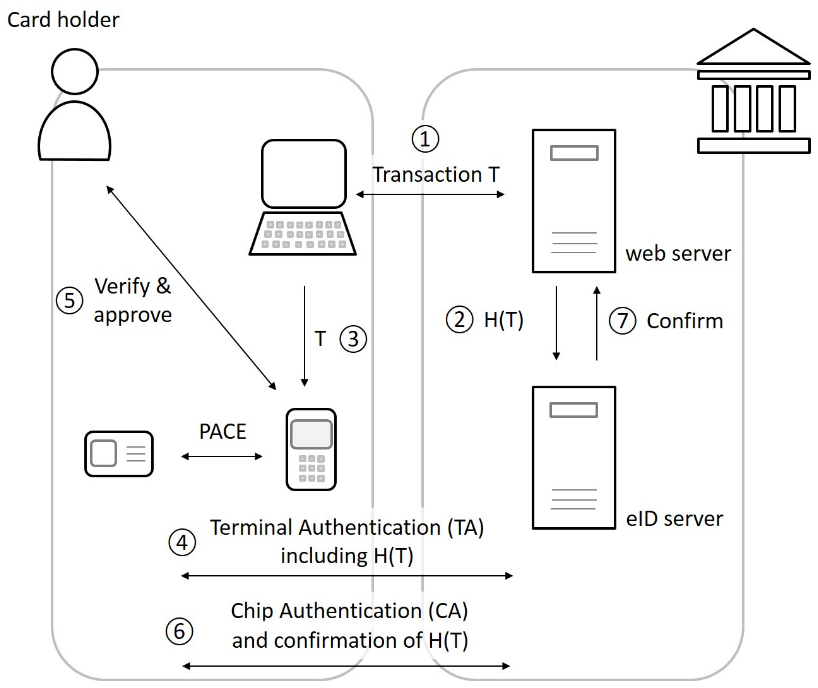

Card holder
① Transaction T
web server
⑤ Verify & approve
T ③
② H(T)
⑦ Confirm
PACE
eID server
④ Terminal Authentication (TA) including H(T)
⑥ Chip Authentication (CA) and confirmation of H(T)

Figure 2: The eIDAS transaction system

Our contribution is to put the security of the eIDAS transaction system on formal grounds. To this end we first discuss a security model which ensures that a transaction system satisfying the requirements in the model provides strong authenticity properties of transactions. The model basically guarantees that both parties, card holder and service provider, can have confidence that they agree on the same transaction with the intended partner in a certain session. By this the security model captures for example replay attacks or cloning of transactions across executions, such that the secure transaction system remains immune against such attacks.

An extra feature of the eIDAS transaction system is that the transaction data can be hidden from the underlying eID service. That is, instead of passing the transaction in clear to the service, one can forward a hashed (or committed) version to the service. Our other contribution is to analyze, too, the privacy guarantees given through such a step formally.

We note again that [3] presents an implementation of the above protocol which requires only minor modifications on the reader's side, but is otherwise based on the existing software and hardware for the German identity card. We report on this briefly in Section 5.

1.3 Related Work

Since we model transactions as digital data, the authenticity problem of transaction systems at the core resembles the authenticity problem in secure channels. Therefore, it is no surprise that previous approaches for modeling secure channels such as [4, 9, 17, 16] can serve as a starting point to model security for the transaction system. There are, nonetheless, significant differences.

On one hand, we face a simpler problem in the sense that transactions are atomic and we do not have to deal for example with the order of channel transmissions. On the other hand, a transaction in our sense should be authenticated *mutually*: both parties should be ensured that the transaction is authentic. In contrast, most channel protocols and models consider unidirectional message transfers, where the message is sent and authenticated by one party only. This in principle allows for mutual authentication by "mirroring" the message back.

In fact, a secure transaction system could be implemented straightforwardly in principle via signatures: have both parties exchange nonces and then have each party sign the transaction and the nonces under its certified key. The nonces would prevent replay attacks. The point of the solution in [3], however, is to build upon an existing system with reliable hardware components such as the identity card and the card reader, *without or with minimal* interference with the underlying system.

The protocols of the German identity card and (some of) the eIDAS extensions have been analyzed in a series of works [7, 11, 15, 10, 5, 6, 13, 8, 12]. In fact, supporting the protocols by security arguments in form of cryptographic proofs has been a key point in the marketing strategy. We do not directly rely on these results, but we draw on the EAC analysis in [11] for showing that the CA ensures transaction authenticity for the service provider.

2 The eIDAS transaction system

We first define transaction systems abstractly, and then present the eIDAS transaction system of Bundesdruckerei in our notation.

2.1 Transaction Systems

A *transaction system* $\mathcal{TS}$ is given by an interactive protocol between one type of parties, called card holders (or clients) and another type of parties, denoted as service providers (or servers). This is formally specified through a pair of (stateful) algorithms $\Pi_{\mathcal{C}}, \Pi_{\mathcal{S}}$ which describe the next message the corresponding party sends upon receiving a message from the other partner. In addition to the interactive protocol the system also comprises key generation algorithms $\mathsf{KG}_{\mathcal{C}}$ and $\mathsf{KG}_{\mathcal{S}}$ for both types of parties. The key generation algorithms generate key pairs (sk, pk) for the corresponding party, together with a certificate *cert*.

We assume uniqueness of the certificates and it is convenient to view some unique identifier in a certificate, such as the serial number, as the party's (administrative) identity id. We often use *cert* interchangeably as the identifier id. The certification process is prepared by a general Setup algorithm which creates public parameters such as the root key pk_{CVCA}

for verification of certificate chains, as well as secret parameters for certificate generation, typically signing keys.

The interactive protocol execution between a card holder and service provider starts with both parties receiving their keys and certificate as input. Both parties also get the public data of the Setup algorithm as additional input. Each party usually also receives a transaction T as input which it tries to authenticate. A party may abort the execution at any time; it may also accept and terminate. We assume the usual completeness requirement that, if running on genuine parameters and the same transaction, both parties accept.

A *preportioned* transaction system does not hand over the transaction T to the parties in clear. Instead, it uses another algorithm H (as a mnemonic for a hash function) to first compute $\mathsf{H}(\mathsf{T})$ before handing the value to the party. The parameters of this algorithm can be specified through the Setup algorithm, and it may be even be a probabilistic algorithm: $\mathsf{H}(\mathsf{T}; r)$ for random string r.[1] We nonetheless often simply refer to $\mathsf{H}(\mathsf{T})$ instead of $\mathsf{H}(\mathsf{T}; r)$, as the difference becomes relevant only for the privacy setting. Note that the actual transaction system can then only ensure validity of $\mathsf{H}(\mathsf{T})$ and that this extends to T must be checked by some other mean, e.g., as in the eIDAS transaction system providing the card reader also with T, r and having it check these values against the hash value used in the underlying transaction system.

2.2 The eIDAS transaction system

The eIDAS transaction system basically consists of the EAC system where we use the auxiliary data field A_T to transport the (hidden) transaction $\mathsf{H}(\mathsf{T})$. Since our goal is to build upon the existing scheme, we do not argue about the specific choices of the EAC system here but instead refer to [1, 2] for information about the design ratio. In more detail, both parties first execute the so-called Terminal Authentication (TA) steps of the extended access control.[2] In this step, the terminal picks a fresh ephemeral key pair $(\mathit{esk}_T, \mathit{epk}_T)$ for the domain parameters D_C describing the elliptic curve, sends over its certificate for the long-term key pk_T (which is also included in cert_T) and a compressed version $\mathrm{Compr}(\mathit{epk}_T)$ of the ephemeral public key. The compression function can be for example the projection onto the x-coordinate of the elliptic curve point. For our analysis we merely assume that Compr is R-regular, meaning that each image has exactly R pre-images.

The chip replies with a random nonce r_C and the terminal then signs this nonce, together with $\mathrm{Compr}(\mathit{epk}_T)$. In this step, the eID server of the service provider augments the signature step by the hash value $\mathsf{H}(\mathsf{T})$ it has received from the web server. This assumes that the forwarding of $\mathsf{H}(\mathsf{T})$ to the eID server is carried out securely; else it is easy to replace the otherwise unprotected transaction T in the communication with the card holder. Note that in the TA protocol both parties also use some chip identifier id_C, not to be confused with our administrative identifier id, for signature generation and verification. Since this value is

[1]In combination with the yet-to-be-specified security properties this makes the algorithm rather a commitment algorithm but which, in turn, can be implemented via a hash function.

[2]We skip the card holder's internal execution of the PACE protocol and assume a trustworthy reader and a secure connection between ID card and reader. This is justified by the fact that PACE has been shown to be a secure password-authenticated key exchange protocol [7] and that we anyhow require a trustworthy display for the user to check the transaction in clear.

irrelevant for our security analysis we do not comment further on this value. The chip finally verifies the signature.

Upon successful completion of the TA step, the reader verifies that the given transaction T matches the augmented data $\mathsf{H}(\mathsf{T})$ from the TA step and then displays the transaction to the user. Upon confirmation of the user, the reader initiates the Chip Authentication (CA) step between the card and the eID server. In this step the chip sends its certificate $cert_C$ and public key pk_C, and the terminal replies with its ephemeral public key epk_T (in clear).

Finally, both parties compute the Diffie-Hellman key of pk_C and epk_T with the corresponding secret key they hold, and derive an encryption key K_{enc} (which is irrelevant here) and the MAC key K_{mac} with the help of the corresponding key derivation functions $\mathsf{KDF}_{\mathsf{Enc}}$ and $\mathsf{KDF}_{\mathcal{M}}$. This step again involves a random nonce r'_C of the chip. The chip computes the MAC over epk_T and sends it to the terminal. If this phase is completed successfully then the eID server reports to the web server that the transaction has been confirmed. This, again, must be done in a secure way.

The protocol details of the TA and CA phase are depicted in Figure 3. For administrative purposes in light of the security model it is convenient to also specify a point in time in the execution in which a session identifier sid is set. This session identifier can be thought of as a quasi unique, purely administrative value, although the parties can determine the identifier easily themselves. Note that both parties output the identifier sid at different points of the execution, due to the sequential order of the authentication steps.

Analogously, we let the protocol specify a partner identifier, pid, which should represent the identity of the intended partner. Since we assume a one-to-one correspondence between certificates and administrative identities, we extract the correct identity out of the certificate. Completeness requires that both session identifier and partner identifier are set upon acceptance, that session identifiers are identical for both parties in genuine executions, and that the partner identifiers correctly point to the corresponding parties.

3 Unforgeability of Transactions

In this section we provide our security analysis of the eIDAS transaction system. For this we first introduce a demanding security model for unforgeability, then we discuss that the eIDAS transaction system achieves this property.

3.1 Defining Unforgeability

Attack Model. We assume that all parties, divided exclusively into card holders from set $\mathcal{C}$ and service providers from a set $\mathcal{S}$, receive their (certified) key pairs as initial input at the outset of the attack. We furthermore presume security of the certification in the sense that parties will only accept certified keys. This follows from the unforgeability of the underlying signature scheme used to create certificate, or even certificate chains, but we omit this here for sake of simplicity.

In the attack model the network is fully controlled by the adversary, implying that the adversary decides when to deliver messages to sessions and if to modify transmissions or even to inject new messages. We even assume that the adversary decides when to start new

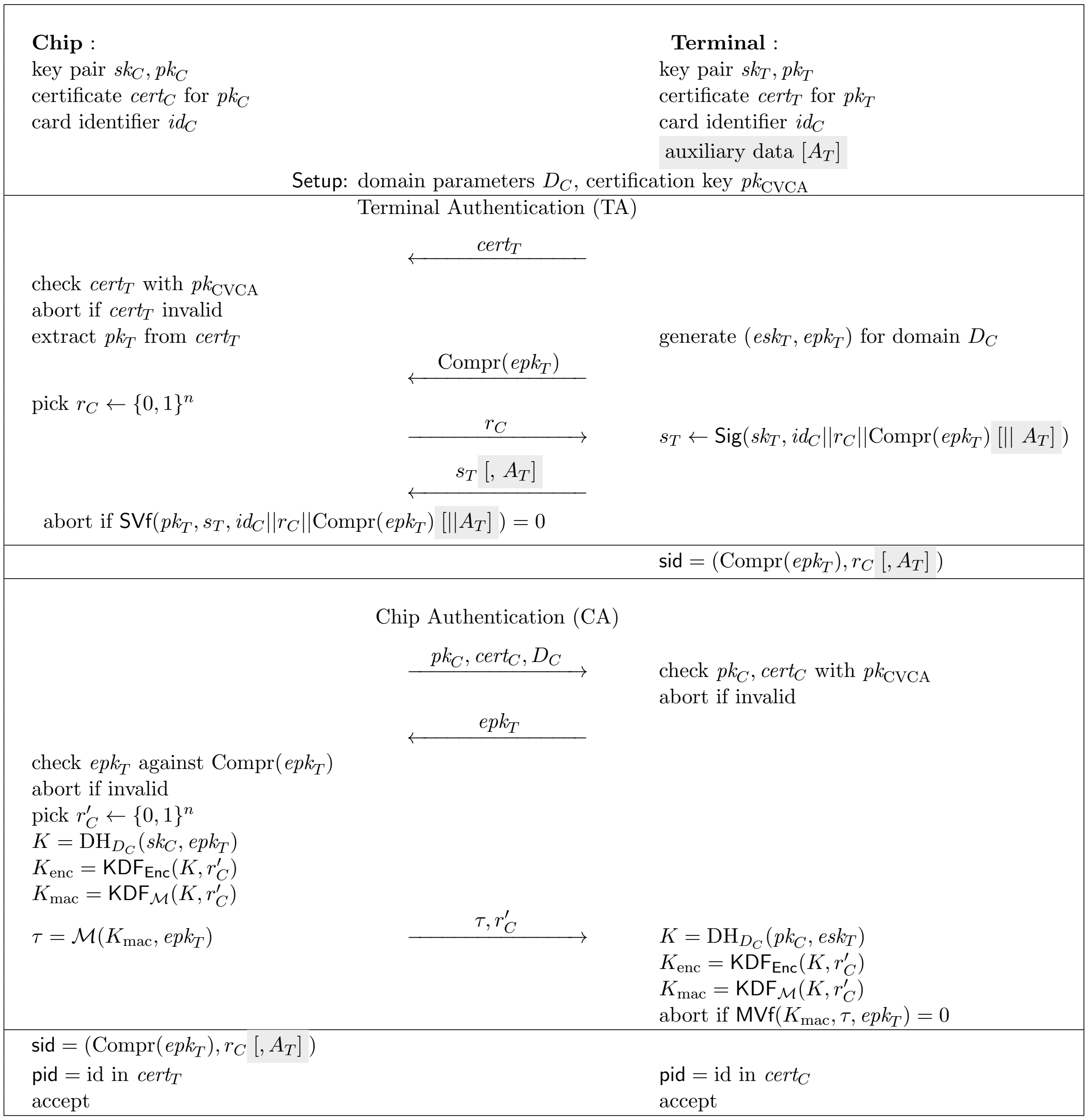

Chip :
key pair sk_C, pk_C
certificate $cert_C$ for pk_C
card identifier id_C
Terminal :
key pair sk_T, pk_T
certificate $cert_T$ for pk_T
card identifier id_C
auxiliary data $[A_T]$
Setup: domain parameters D_C, certification key pk_{CVCA}
Terminal Authentication (TA)
$cert_T$
check $cert_T$ with pk_{CVCA}
abort if $cert_T$ invalid
extract pk_T from $cert_T$
generate (esk_T, epk_T) for domain D_C
$\text{Compr}(epk_T)$
pick $r_C \leftarrow \{0,1\}^n$
r_C
$s_T \leftarrow \mathsf{Sig}(sk_T, id_C||r_C||\text{Compr}(epk_T) \; [|| \; A_T] \;)$
$s_T \; [, \; A_T]$
abort if $\mathsf{SVf}(pk_T, s_T, id_C||r_C||\text{Compr}(epk_T) \; [||A_T] \;) = 0$
$\mathsf{sid} = (\text{Compr}(epk_T), r_C \; [, A_T] \;)$
Chip Authentication (CA)
$pk_C, cert_C, D_C$
check $pk_C, cert_C$ with pk_{CVCA}
abort if invalid
epk_T
check epk_T against $\text{Compr}(epk_T)$
abort if invalid
pick $r'_C \leftarrow \{0,1\}^n$
$K = \text{DH}_{D_C}(sk_C, epk_T)$
$K_{\text{enc}} = \mathsf{KDF}_{\mathsf{Enc}}(K, r'_C)$
$K_{\text{mac}} = \mathsf{KDF}_{\mathcal{M}}(K, r'_C)$
$\tau = \mathcal{M}(K_{\text{mac}}, epk_T)$
τ, r'_C
$K = \text{DH}_{D_C}(pk_C, esk_T)$
$K_{\text{enc}} = \mathsf{KDF}_{\mathsf{Enc}}(K, r'_C)$
$K_{\text{mac}} = \mathsf{KDF}_{\mathcal{M}}(K, r'_C)$
abort if $\mathsf{MVf}(K_{\text{mac}}, \tau, epk_T) = 0$
$\mathsf{sid} = (\text{Compr}(epk_T), r_C \; [, A_T] \;)$
pid = id in $cert_T$
accept
pid = id in $cert_C$
accept

Figure 3: EAC protocol, consisting of Terminal Authentication (TA) and Chip Authentication (CA). All operations are modulo q resp. over the elliptic curve. The gray part shows the (in EAC optional) auxiliary data field, which we deploy here for securing the transaction.

sessions at either party, possibly as a pair of synchronized sessions for the same transaction. Formally, this is captures by giving the adversary access to the following oracles:

- Init: The adversary can initiate new card holder or service provider sessions by calling

INIT(id, T) for some identity id $\in \mathcal{C} \cup \mathcal{S}$ and some chosen transaction T. Upon such a call we spawn a new session of the party for transaction T (and possibly a freshly chosen random string r in case of probabilistic hash functions) and assign it a unique label ℓ for administrative purposes. The label ℓ is returned to the adversary and we write $\ell \leftarrow$ INIT(id, T).

  In the case of a probabilistic hash function we need to give the adversary the ability to initiate sessions with the same random value r. Hence, slightly overloading the notation we allow the adversary to also pass triples $(\mathrm{id}, \mathrm{id}', \mathsf{T})$ to the INIT oracle for id $\in \mathcal{C}$ and id$' \in \mathcal{S}$, upon which the oracle initializes a new session for card holder id as well as a new session for service provider id$'$. This time, however, each protocol party receives the same random string r as additional input. The adversary receives both labels (ℓ, ℓ') of the two sessions.

- SEND: The adversary can send any protocol message m to a session with label ℓ via the SEND(ℓ, m) command. If the session has not been initialized before, then oracle immediately returns $\perp$. Else, it makes the corresponding party compute the next protocol message and this message is returned to the adversary (potentially also returning $\perp$ to express rejection). In case the execution is successfully completed, the adversary is informed that the party has accepted. From then on, we assume that the adversary no longer sends commands to that session.

- CORRUPT: The adversary can corrupt a party with identity id via the command CORRUPT(id). It receives the party's secret key in return, as well as all internal states of running sessions, and we put id in the (initially empty) set $\mathsf{Corrupt}$ of corrupt parties. From now on, we assume that the adversary does not send further commands to that session.

It is often convenient to augment the session identifier sid, output by the execution of some honest party, by the transaction T it has been initialized with. In this case we write $\mathsf{tsid} = (\mathsf{T}, \mathsf{sid})$ for the augmented identifier (but omitting the random string r potentially used for computing the probabilistic hash value).

We also use the following self-explanatory notation. We write $\mathsf{TSID}(\ell)$ for the value tsid of the session with label ℓ, where potentially $\mathsf{TSID}(\ell) = \perp$ if the session identifier has not been set by the session yet. Analogously, we write $\mathsf{ACC}(\ell)$ for the acceptance status of the session (`true` or `false`), $\mathsf{ID}(\ell)$ for the identity id of the session owner, and $\mathsf{PID}(\ell)$ for the intended partner pid, possibly $\mathsf{pid} = \perp$ at this point.

Session-definite unforgeability. We define a very strong notion of unforgeability: if some (honest) card holder accepts a transaction in some execution, then there must be an execution in which a service provider has issued that transaction. Furthermore, this execution of the service provider points to a single card-holder session only. Note that this straightforwardly implies other desirable notions of unforgeability, such as plain unforgeability—if a card holder accepts some transaction then it must have been issued by some service provider—or replay resistance—that no card holder accepts a transaction twice. The latter follows from the fact

Experiment $\mathsf{SessUnf}_{\mathcal{A}}^{\mathcal{TS}}(n)$

```
1 : foreach $i \in \mathcal{C} \cup \mathcal{S}$ do
2 :    if $i \in \mathcal{C}$ then $(sk_i, pk_i, cert_i) \leftarrow \mathsf{KG}_{\mathcal{C}}(1^n)$ fi
3 :    if $i \in \mathcal{S}$ then $(sk_i, pk_i, cert_i) \leftarrow \mathsf{KG}_{\mathcal{S}}(1^n)$ fi
4 : endforeach
5 : $pks \leftarrow \{(pk_i, cert_i) \mid i \in \mathcal{C} \cup \mathcal{S}\}$
6 : $\mathcal{A}^{\textsc{Init}(\cdot,\cdot),\textsc{Send}(\cdot,\cdot),\textsc{Corrupt}(\cdot)}(1^n, pks)$
7 : $b \leftarrow \mathsf{SessUnfPred}$   // evaluate predicate SessUnfPred on execution state
8 : return $\bar{b}$   // where $\bar{b} = \mathrm{NOT}(b)$
```

Figure 4: Session-definite unforgeabilty of transaction system.

that there is a one-to-one correspondence between service-provider sessions and card-holder sessions. We call our notion *session-definite unforgeability.*

More formally, we say that the adversary $\mathcal{A}$ wins the session-definite unforgeability game in Figure 4 if during the execution an honest card-holder party id $\in \mathcal{C} \setminus \mathsf{Corrupt}$ *accepts* a transaction T under some session identifier tsid but such that there the partner id pid is not pointing to a corrupt provider and that provider has not set their local identifier to sid at that point. The latter corresponds to the fact that the service provider has issued the transaction. By symmetry, we demand the same for accepting sessions of honest service providers, i.e., that there must be a corresponding card-holder session.

In addition, we assume that session identifiers only appear in matching pairs on the sides. This implies that there cannot be session-identifier collisions on the card-holder side, nor on the service-provider side. This also means that it suffices to demand for the first property above that every accepting session has a matching partner; together with collision-freeness of session identifiers on one side the matching session must be on the other party's side. The formal predicate capturing these requirements is depicted in Figure 5 and called as a subroutine in the attack game.

Definition 3.1 (Session-definite unforgeability) *A transaction system $\mathcal{TS}$ is session-definite unforgeable if for any efficient adversary $\mathcal{A}$ we have that*

$$\mathrm{Prob}\left[\mathit{SessUnf}_{\mathcal{A}}^{\mathcal{TS}}(n) = 1\right] \approx 0$$

is negligible.

Note that we let the adversary $\mathcal{A}$ decide when to stop the execution and to start evaluating the predicate. Hence, if it is advantageous and the adversary already detects a winning situation, it may end the execution immediately (instead of messing up the winning state by, say, corrupting another party). In our case this is easy to spot since all the data required to evaluate the predicate are known to the adversary.

Predicate $\mathsf{SessUnfPred}$ on execution state

```
1 :  $p \leftarrow \texttt{true}$
2 :  // any accepting party must have honest partner with same tsid (or corrupt partner)
3 :  foreach $\ell \in \{\ell \mid \mathsf{ACC}(\ell) = \texttt{true} \wedge \mathsf{ID}(\ell) \notin \mathsf{Corrupt}\}$ do
4 :      $p \leftarrow p \wedge [\mathsf{PID}(\ell) \in \mathsf{Corrupt} \vee \exists \ell' \neq \ell : (\mathsf{TSID}(\ell') = \mathsf{TSID}(\ell) \neq \bot \wedge \mathsf{PID}(\ell) = \mathsf{ID}(\ell'))]$
5 :  endforeach
6 :  // Collisions among identifiers only between opposite partners
7 :  foreach $(\ell, \ell') \in \{(\ell, \ell') \mid \ell \neq \ell' \wedge \mathsf{ID}(\ell), \mathsf{ID}(\ell') \notin \mathsf{Corrupt} \wedge \mathsf{TSID}(\ell) = \mathsf{TSID}(\ell') \neq \bot\}$ do
8 :      $p \leftarrow p \wedge [(\mathsf{ID}(\ell), \mathsf{ID}(\ell')) \in \mathcal{C} \times \mathcal{S} \cup \mathcal{S} \times \mathcal{C}]$
9 :  endforeach
10 : return $p$   // where we identify true = 1 and false = 0
```

Figure 5: Security predicate $\mathsf{SessUnfPred}$ for session-definite unforgeability

3.2 Security of the eIDAS transaction system

Before proving the eIDAS transaction system unforgeable we need to make some assumptions about the underlying cryptographic primitives. As in [11] we model the key derivation function $\mathsf{KDF}_{\mathcal{M}}$ as a random oracle. We furthermore assume that the H function used to hide the actual transaction is collision-resistance in the sense that the probability $\mathbf{Adv}^{\mathrm{coll}}_{\mathcal{B},\mathsf{H}}(n)$ of $\mathcal{B}$ outputting such a collision is negligible for every efficient adversary $\mathcal{B}$.

We also assume that forging signatures is infeasible, i.e., for any efficient adversary $\mathcal{B}$ the probability $\mathbf{Adv}^{\mathrm{unf}}_{\mathcal{B},\mathcal{SIG}}(n)$ of breaking the terminal's signature scheme given through $\mathcal{SIG} = (\mathsf{KG}_{\mathcal{S}}, \mathsf{Sig}, \mathsf{SVf})$ in an adaptive chosen-message attack (see [14] for a formal definition) is negligible. We analogously demand that forging MACs for the scheme given through $\mathcal{MAC} = (\mathsf{KDF}_{\mathcal{M}}, \mathsf{MAC}, \mathsf{MVf})$ is infeasible, i.e., $\mathbf{Adv}^{\mathrm{unf}}_{\mathcal{B},\mathcal{MAC}}(n)$ is negligible for any efficient adversary $\mathcal{B}$ against the MAC scheme.

Finally, as in [11] we assume that the Gap-Diffie-Hellman problem in the group specified by D_C is hard. That is, consider an efficient adversary $\mathcal{B}$ which receives (in multiplicative notation) g^a, g^b for generator g and hidden random a, b and which gets access to an oracle which for adversarially chosen group elements Y, Z verifies for the adversary whether $Y^a = Z$ or not. The adversary's task it to output $\mathrm{DH}(g^a, g^b) = g^{ab}$. We now require that for no efficient adversary the probability $\mathbf{Adv}^{\mathrm{GapDH}}_{\mathcal{B},D_C}(n)$ of computing that DH value is negligible.

Theorem 3.2 (Session-definite Unforgeability) *The eIDAS transaction system in Section 2 is session-definite unforgeable in the random oracle model, assuming collision resistance of* H*, unforgeability of the signature and MAC scheme, and the GapDH assumption. More precisely, for any efficient adversary $\mathcal{A}$ there exists efficient adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ such that*

$$\begin{aligned}\mathrm{Prob}\left[\mathit{SessUnf}^{\mathcal{TS}}_{\mathcal{A}}(n) = 1\right] \quad \leq \quad & \binom{S}{2} \cdot \left(2^{-n} + \tfrac{R}{q}\right) + \boldsymbol{Adv}^{coll}_{\mathcal{B}_1,\mathsf{H}}(n) + S \cdot \boldsymbol{Adv}^{unf}_{\mathcal{B}_2,\mathcal{SIG}}(n) \\ & + S \cdot \boldsymbol{Adv}^{unf}_{\mathcal{B}_3,\mathcal{MAC}}(n) + C \cdot S \cdot \boldsymbol{Adv}^{GapDH}_{\mathcal{B}_4,D_C}(n)\end{aligned}$$

where we assume that Compr is a R-regular function, q is the group size specified by D_C, the adversary initiates at most s sessions, and there are at most C cards and S service providers.

Moreover, adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_4$ have roughly the same running time $\mathcal{A}$ plus the time to carry out the other steps in the experiment.

Proof. We given the proof only for the case of a deterministic hash $\mathsf{H}(\mathsf{T})$ and single initializations through Init; for probabilistic hashes and synchronous initializations the proof can be adapted easily.

Assume that an adversary $\mathcal{A}$ mounts an attack against session-definite unforgeability. In order to win the adversary must either create a collision in two honest card-holder sessions, or in two honest service-provider sessions, or make one of the two types of parties accept such that there is no matching session (with a party of the other type, either corrupt or not having the same tsid).

The idea of ruling out the above attack possibility is that: sessions among honest card holders or among honest service providers do not collide because each session identifier sid contains fresh random values r_C resp. epk_T of both sides. The other property follows since the adversary, unless it forges signatures, cannot make a card holder accept without having an honest provider sign and output the session identifier sid. Since the session identifier includes the collision-resistant hash of transaction T this argument extends to the fully augmented identifiers tsid, as required by the security definition.

The final step is to note that, analogously to the signature case, an adversary making an honest provider accept without having an honest card-holder partner also requires the adversary to forge MACs. The latter argument is a bit more involved, but can be seen to follow from the proof that TA together CA is a secure key exchange protocol [11]. Here we nonetheless give a self-contained and "stripped-off" version of the proof for our case.

Collision-freeness of session identifiers. We first note that the (augmented) session identifiers contain a fresh random nonce r_C of the card holder, such that a collision among the at most s honest card-holder sessions is bounded by the birthday bound $\binom{s}{2} \cdot 2^{-n}$. Analogously, the same holds for the random ephemeral key epk_T on the service provider side, yielding a collision probability for honest servers of at most $\binom{s}{2} \cdot \frac{R}{q}$ for the size q of the underlying group and the compression regularity factor R. We may therefore from now on assume there can be only collisions between at most one pair of honest card holder and service provider.

For sake of concreteness we can also assume that no collisions in the hashed transactions $\mathsf{H}(\mathsf{T})$ occur during the execution, i.e., there are no $\mathsf{T} \neq \mathsf{T}'$ used in some initialization step such that $\mathsf{H}(\mathsf{T}) = \mathsf{H}(\mathsf{T}')$. We can assume that this holds because of the collision resistance of the underlying hash function such that the probability can be bounded by the advantage to find collisions. Also note that both parties ensure that the hash value is correct, the card holder by having the card reader check correctness, and the provider by computing the hash itself. In particular, we can from now assume that equality $\mathsf{sid} = \mathsf{sid}'$ (including the hashed transactions) on some session identifiers of honest parties also implies that $\mathsf{tsid} = \mathsf{tsid}'$ for these parties, i.e., equality also holds for transactions. Otherwise we would have found a collision for the hash values of these transactions.

Unforgeability on the card holder's side. Next assume that some card-holder session ℓ accepts with identifier tsid. Since the card holder verifies the service provider's certificate *cert* it will only accept a public key under some eligible service provider identity. If the provider behind this identity is corrupt then the requirement of having a corrupt partner is satisfied. Else, if the provider is honest, then the fact that the provider with certificate *cert* has not output tsid yet in any session implies that it has not signed sid in any session yet. (Here we use that sid uniquely determines tsid in the experiment.)

Since the card holder also verifies the signature it must be that the adversary provided a valid forgery to make the card holder accept. If an adversary would trigger this event with non-negligible probability then this can be easily transformed via a reduction into an existential forgery against the service provider's underlying signature scheme, by simply guessing the right key among the S service providers. This shows that honest card holders only accept transactions from honest providers if the providers have output tsid and if the partner identifier matches.

Unforgeability on the provider's side. Finally, assume that an honest service provider session ℓ accepts for (augmented) identifier tsid and partner identifier pid. If this partner identifier points to a corrupt card holder, then we are done. So we may assume that the intended partner is an honest card holder but which has not output tsid.

We first show via a reduction to the GapDH problem that the probability that the adversary asks the random oracle $\mathsf{KDF}_{\mathcal{M}}$ about the Diffie-Hellman key of pk_C and epk_T of that session is negligible. Otherwise we build a DH solver (with a DDH decision oracle) as follows: The solver receives (in multiplicative notation) g^a, g^b in D_C as input. It initially picks one of the at most C card holders and one of the at most S provider sessions at random and starts $\mathcal{A}$'s attack. For the predicted card holder it uses g^a as public key pk_C, and for the predicted provider session it uses g^b as the ephemeral key epk_T. It picks all the other keys according to the protocol description and follows the honest parties steps, exploiting knowledge of all secret keys.

The only exception lies in the way our solver treats queries to the random oracle $\mathsf{KDF}_{\mathcal{M}}$. Our algorithm keeps two lists to emulate the random oracle: one containing entries $((K, r_C), y)$ which represent "regular" queries to and answers from the random oracle, and the other one containing tuples $((A, B, r'_C), y)$ where the solver is not (yet) able to compute the Diffie-Hellman key of A and B. Both lists are initially empty.

If a simulated honest party or the adversary makes a query (K, r'_C) to $\mathsf{KDF}_{\mathcal{M}}$ then our solver checks

- if there already exists an entry $((K, r'_C), y)$ in the first list, then return y;
- else, if there is already an entry $((A, B, r'_C), y)$ in the second list such that $\mathrm{DH}(A, B) = K$—which can be checked via the decision oracle— then store $((K, r'_C), y)$ in the first list and return y;
- else, pick a random y in the range of $\mathsf{KDF}_{\mathcal{M}}$, store $((K, r'_C), y)$ in the first list and return y.

If the user instance for which we have set $pk_C = g^a$ for the unknown a is supposed to compute $\mathrm{DH}(pk_C, epk_T)$ for deriving the MAC in some session with random string r'_C and (potentially maliciously chosen) ephemeral key epk_T —where the solver may not be able to compute the DH key— then our solver instead checks:

- If there exists already an entry $((K, r'_C), y)$ in the first list where $K = \mathrm{DH}(pk_C, epk_T)$, then it returns y;
- else, if there exists already an entry $((pk_C, epk_T, r'_C), y)$ in the second list, it uses y;
- else, it picks a fresh y, stores $((pk_C, epk_T, r'_C), y)$ in the second list and returns y.

In any case the solver uses the returned value y to compute the subsequent MAC. At the end the solver checks if there is an entry $((K, r'_C), y)$ in the first list with $\mathrm{DH}(g^a, g^b) = K$ and, if so, outputs K.

Note that the above simulation of the random oracle is perfectly indistinguishable from the actual random oracle, from $\mathcal{A}$'s point of view. The reason is that we only store pairs (g^a, epk_T) in the second list for different values for epk_T. Each of these entries can only correspond to at most one entry in the first list, and our solver checks for consistency with the first list. There cannot be any inconsistency in the second list itself, e.g., entries $((A, B, r'_C), y)$, $((A', B', r'_C), y')$ with $\mathrm{DH}(A, B) = \mathrm{DH}(A', B')$ but $y \neq y'$, since all pairs (g^a, epk_T) for the identical vale g^a and varying epk_T map to different DH keys.

By assumption, the adversary against transaction unforgeability at some point queries the random oracle $\mathsf{KDF}_{\mathcal{M}}$ in the original game about $\mathrm{DH}(pk_C, epk_T)$ with non-negligible probability for honest parties holding pk_C resp. epk_T. If our solver picks the right indices for $pk_C = g^a$ and $epk_T = g^b$ then the otherwise sound simulation yields an entry for $\mathrm{DH}(g^a, g^b)$ in the first list with non-negligible probability. In this case the solver would break the GapDH problem with non-negligible probability, losing a factor $C \cdot S$ in the reduction for guessing the right sessions.

The final step is to note now that, unless we break the GapDH problem with non-negligible probability, the adversary against transaction unforgeability needs to forge the final MAC in a key only attack. For this assume that there is a session of an honest provider with tsid, including pk_C of an honest card holder and epk_T chosen by the provider itself. Then, if the card holder has also output sid and thus tsid, then this does not violate unforgeability. Else this means that the card's MAC has not been computed yet, implying that the adversary has neither learned the key K_{mac} yet (by the previous consideration), nor seen a MAC under that key. Hence, it needs to produce a MAC forgery in a key-only attack to make the service provider accept. By a similar argument as in case of the provider's signature scheme one can show that can be formally reduced to the unforgeability of the MAC scheme, with a loss of factor S in the security for guessing the right session on the provider's side. □

4 Transaction Privacy

Analogous to the case of unforgeability we first describe the general security requirements and then discuss that the eIDAS transaction system meets these requirements.

Experiment $\mathsf{TPriv}_{\mathcal{A}}^{\mathcal{TS}}(n)$

```
1 : $b \leftarrow \{0,1\}$
2 : **foreach** $i \in \mathcal{C} \cup \mathcal{S}$ **do**
3 :     **if** $i \in \mathcal{C}$ **then** $(sk_i, pk_i, cert_i) \leftarrow \mathsf{KG}_{\mathcal{C}}(1^n)$ **fi**
4 :     **if** $i \in \mathcal{S}$ **then** $(sk_i, pk_i, cert_i) \leftarrow \mathsf{KG}_{\mathcal{S}}(1^n)$ **fi**
5 : **endforeach**
6 : $pks \leftarrow \{(pk_i, cert_i) \mid i \in \mathcal{C} \cup \mathcal{S}\}$
7 : $a \leftarrow \mathcal{A}^{\textsc{Init}(\cdot,\cdot),\textsc{Send}(\cdot,\cdot),\textsc{Corrupt}(\cdot),\textsc{Chall}(b,\cdots)}(1^n, pks)$
8 : **return** $a = b$
```

Figure 6: Transaction privacy experiment

4.1 Defining Privacy

Transaction privacy refers to the inability for the eID service to determine the transaction, even though it operates on the hash value. We use an indistinguishability-based approach here in which a privacy-adversary can initiate (multiple) executions on a random choice of one of two adversarially-chosen transactions $\mathsf{T}_0, \mathsf{T}_1$. This of course assumes that transactions are augmented by sufficient entropy, or else the adversary can easily determine the choice of the transaction used in the hash value.

The attack model is as in case of unforgeability. The only difference is now that the adversary now also gets a challenge oracle $\textsc{Chall}$, which is initialized with a secret bit $b \leftarrow \{0,1\}$. If called about identities $\mathrm{id} \in \mathcal{C}$, $\mathrm{id}' \in \mathcal{S}$, as well as two transactions $\mathsf{T}_0, \mathsf{T}_1$, then the challenge oracle executes $\ell \leftarrow \textsc{Init}(\mathrm{id}, \mathrm{id}', \mathsf{T}_b)$ to initialize both parties, and it returns the session lables (ℓ, ℓ') to the adversary. From then on the adversary can communicate with the card-holder and service provider sessions via the $\textsc{Send}$ oracle for the corresponding label (and in particular learn the hashed transaction). The adversary eventually should predict the bit b.

We next define privacy with the experiment in Figure 6.

Definition 4.1 (Transaction Privacy) *A transaction system $\mathcal{TS}$ is transaction private if for any efficient adversary $\mathcal{A}$ we have that*

$$\mathrm{Prob}\left[\mathsf{TPriv}_{\mathcal{A}}^{\mathcal{TS}}(n) = 1\right] \le \tfrac{1}{2} + \mathsf{negl}(n)$$

is negligibly close to $\frac{1}{2}$.

Conceivably, any transaction-private system needs to be preportioned.

4.2 Privacy of the eIDAS transaction system

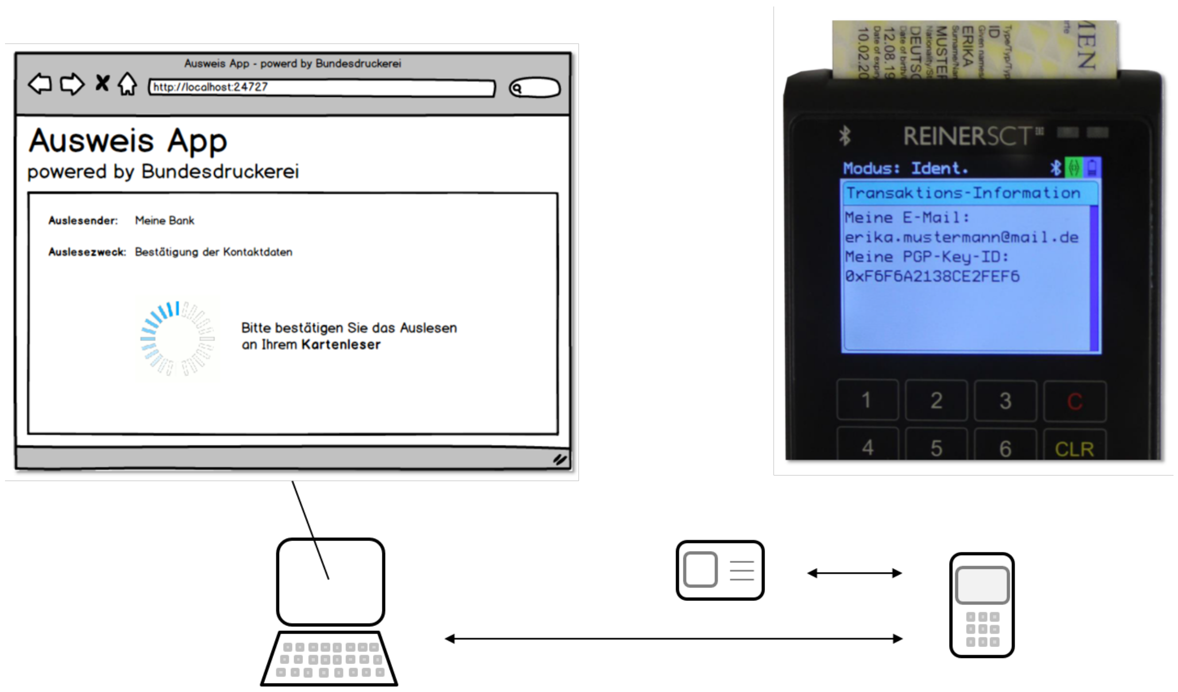
The privacy of the eIDAS transaction system now relies on the hiding property of the hash function, namely, that the probability of being able to distinguish $\mathsf{H}(\mathsf{T}_0, r_0)$ from $\mathsf{H}(\mathsf{T}_1, r_1)$ for chosen $\mathsf{T}_0, \mathsf{T}_1$ and random r_0, r_1 is infeasible for any efficient adversary $\mathcal{B}$. We note that this is straightforward to formalize in terms of the hiding property of commitment schemes (see again [14]), such that for the advantage $\mathbf{Adv}_{\mathcal{B},\mathsf{H}}^{\mathrm{hide}}(n)$ distinguishing the two cases it holds:

Theorem 4.2 (Transaction Privacy of the eIDAS transaction system) *The eIDAS transaction system provides transaction privacy. More precisely, for any efficient adversary $\mathcal{A}$ making at most Q challenge queries there exists an efficient adversary $\mathcal{B}$ such that*

$$\text{Prob}\left[\mathsf{TPriv}_{\mathcal{A}}^{\mathcal{TS}}(n) = 1\right] \leq \tfrac{1}{2} + Q \cdot \boldsymbol{Adv}_{\mathcal{B},\mathsf{H}}^{hide}(n).$$

Moreover, the running time of $\mathcal{B}$ is essentially the one of $\mathcal{A}$, plus the time to execute the other steps of the privacy experiment.

Proof. Each call of $\mathcal{A}$ to the Chall oracle creates another fresh "commitment" of either the left or the right transaction. Then all parties work on the hashed value only. It follows that $\mathcal{A}$'s success probability is bounded by the probability of distinguishing at most Q left commitments from Q right commitments. Since a standard hybrid argument for commitments shows that such Q commitments (for adaptively chosen pairs) can only increase the distinguishing advantage compared to a single commitment by a factor Q, the claim now follows. ☐


Ausweis App - powerd by Bundesdruckerei

http://localhost:24727

Ausweis App

powered by Bundesdruckerei

Auslesender: Meine Bank

Auslesezweck: Bestätigung der Kontaktdaten

Bitte bestätigen Sie das Auslesen an Ihrem **Kartenleser**

Modus: Ident.

Transaktions-Information

Meine E-Mail:
erika.mustermann@mail.de
Meine PGP-Key-ID:
0xF6F6A2138CE2FEF6

Figure 7: Prototype implementation on a card reader. In this example the identity card has already been connected to the card reader, and the user is now asked via the (mock-up) software on the computer to authenticate the e-mail address and PGP fingerprint on the reader.

5 Implementation

As mentioned before, the eIDAS transaction system has been successfully integrated in a test bed for the German identity card, involving the Reiner SCT cyberJack reader. Recall that the user places the card in (or near) the reader and initiates a service through the software on its computer. The software application then requests the user to verify the transaction data (see Figure 7). The transaction data is encoded into an admissible format for the auxiliary data of the eID service under an unused object identifier such that the card, upon receiving the data, authenticates them through the protocol but ignores its semantics. In contrast, the reader has been modified to interpret these data accordingly.

From the user's perspective the steps to authenticate transaction data integrate smoothly with the other eID steps (see Figure 8). In the first step, where the regular eID system and the transaction system differ slightly, the user now checks the transaction data on the card reader, e.g., the mobile phone number in Figure 8. Then the user proceeds as for the regular eID service, confirming the card data the eID service can access (step 2 in Figure 8), and finally entering the card's PIN (step 3).

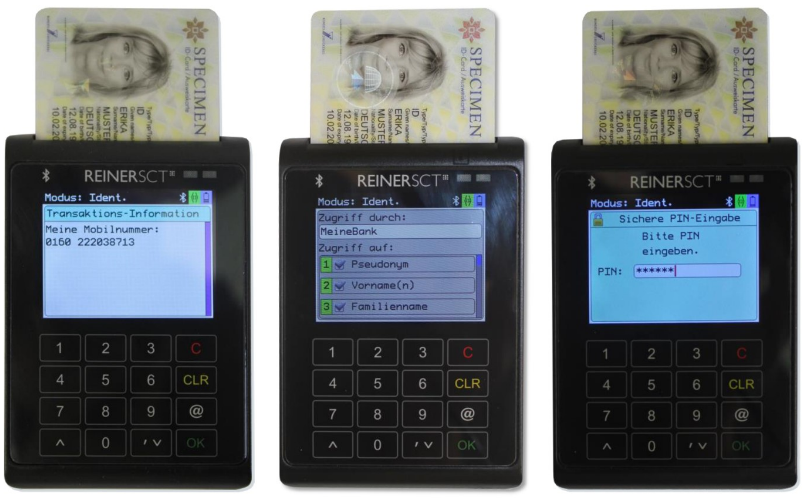


Figure 8: Immigration of transaction verification (left) into data access confirmation (middle) and PIN entering (right) of eID service.

Overall, implementing the eIDAS transaction system requires minimal changes to the underlying system —only the reader's software needs to be adapted slightly— and to provide corresponding software applications.

6 Conclusion

The eIDAS transaction system is an easy and practical method to authenticate transactions via the eIDAS infrastructure. The work here demonstrates that, cryptographically, it also provides strong unforgeability guarantees, thwarting for example replay attacks. Furthermore, it also allows to hide the actual transaction data from the underyling infrastructure, again in a very strong sense.

Acknowledgments

We thank the anonymous reviewers of WISTP 2016 for valuable comments.

References

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI, Federal Office for Information Security): *Advanced Security Mechanism for Machine Readable Travel Documents – Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI)*, BSI-TR-03110, Version 2.0, 2008.

[2] Bundesamt für Sicherheit in der Informationstechnik (BSI, Federal Office for Information Security): *Technical Guideline TR-03110-2: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token*, Part 2, Protocols for electronic IDentification, Authentication and trust Services (eIDAS). BSI-TR-03110, Version 2.2, 2015.

[3] Frank Morgner: Transaktionsabsicherung mit der Online-Ausweisfunktion. Kryptographische Bindung von Transaktionsdaten an den Personalausweis. Presentation, CeBit 2014, March 2014.

[4] Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. ACM Trans. Inf. Syst. Secur. 7(2), 206–241 (2004)

[5] Jens Bender, Özgür Dagdelen, Marc Fischlin, Dennis Kügler: The PACE|AA Protocol for Machine Readable Travel Documents, and Its Security. Financial Cryptography, Lecture Notes in Computer Science, Volume 7397, pp. 344-358, Springer-Verlag, 2012.

[6] Jens Bender, Özgür Dagdelen, Marc Fischlin, Dennis Kügler: Domain-Specific Pseudonymous Signatures for the German Identity Card. Information Security Conference (ISC) 2012, Lecture Notes in Computer Science, Volume 7483, pp. 104-119, Springer-Verlag, 2012.

[7] Jens Bender, Marc Fischlin, Dennis Kügler: Security Analysis of the PACE Key-Agreement Protocol Information Security Conference (ISC) 2009, Lecture Notes in Computer Science, Volume 5735, pp. 33-48, Springer-Verlag, 2009.

[8] Jens Bender, Marc Fischlin, Dennis Kügler: The PACE|CA Protocol for Machine Readable Travel Documents INTRUST 2013, Lecture Notes in Computer Science, Volume 8292, pp. 17-35, Springer-Verlag, 2013

[9] Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)

[10] Jean-Sebastien Coron, Aline Gouget, Thomas Icart, Pascal Paillier: Supplemental Access Control (PACE v2): Security Analysis of PACE Integrated Mapping. In: Cryptography and Security: From Theory to Applications, Volume 6805 of the series Lecture Notes in Computer Science, pp. 207-232, 2012.

[11] Özgür Dagdelen, Marc Fischlin: Security Analysis of the Extended Access Control Protocol for Machine Readable Travel Documents. ISC 2010: 54-68

[12] Lucjan Hanzlik, Miroslaw Kutylowski: Restricted Identification Secure in the Extended Canetti-Krawczyk Model. J. UCS 21(3): 419-439 (2015)

[13] Lucjan Hanzlik, Lukasz Krzywiecki, Miroslaw Kutylowski: Simplified PACE—AA Protocol. ISPEC 2013: 218-232

[14] Jonathan Jatz, Yehuda Lindell: Introduction to Modern Cryptography. Second Edition, Chapman & Hall/CRC Cryptography and Network Security Series, 2015.

[15] Miroslaw Kutylowski, Lukasz Krzywiecki, Przemyslaw Kubiak, Michal Koza: Restricted Identification Scheme and Diffie-Hellman Linking Problem. INTRUST 2011: 221-238

[16] Maurer, U., Tackmann, B.: On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 505–515. ACM Press, October 2010

[17] Namprempre, C.: Secure channels based on authenticated encryption schemes: a simple characterization. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 515–532. Springer, Heidelberg (2002)