Privacy-Preserving Auditing for Attribute-Based Credentials

Jan Camenisch, Anja Lehmann, Gregory Neven, and Alfredo Rial

IBM Research – Zurich, Rüschlikon, Switzerland
{jca,anj,nev,lia}@zurich.ibm.com

Abstract. Privacy-enhancing attribute-based credentials (PABCs) allow users to authenticate to verifiers in a data-minimizing way, in the sense that users are unlinkable between authentications and only disclose those attributes from their credentials that are relevant to the verifier. We propose a practical scheme to apply the same data minimization principle when the verifiers' authentication logs are subjected to external audits. Namely, we propose an extended PABC scheme where the verifier can further remove attributes from presentation tokens before handing them to an auditor, while preserving the verifiability of the audit tokens. We present a generic construction based on a signature, a signature of knowledge and a trapdoor commitment scheme, prove it secure in the universal composability framework, and give an efficient instantiation based on the strong RSA assumption in the random-oracle model.

Keywords: Attribute-based credentials, audits, universal composability, privacy-enhancing technologies.

1 Introduction

Privacy-enhancing attribute-based credentials (PABC) [1], also known as anonymous credentials [2, 3] or minimal-disclosure tokens [4], are cryptographic mechanisms to perform data-minimizing authentication. They allow users to obtain credentials from an issuer, by which the issuer assigns a list of certified attribute values to the user. Users can then use these credentials to authenticate to verifiers, but have the option to disclose only a subset of the attributes; all non-disclosed attributes remain hidden from the verifier. Moreover, different authentications are unlinkable, in the sense that a verifier cannot tell whether they were performed by the same or by different users. PABCs offer important privacy advantages over other attribute certification schemes, which usually either employ a central authority that is involved in every authentication and therefore forms a privacy bottleneck (e.g., SAML, OpenID, or Facebook Connect), or force users to disclose all of their attributes (e.g., X.509 certificates [5]).

But sometimes, attributes travel beyond the verifier. Verifiers may be subjected to external audits to check that access was only granted to entitled users. For example, government authorities may require a video streaming service to

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 109–127, 2014.

prove that age-restricted movies were streamed exclusively to viewers of the required age, or film distributors may require it to prove that films were only streamed to residents of geographic areas for which it bought the rights. It makes perfect sense to extend the data minimization principle to auditors as well: why should auditors be handed any user attributes that are not relevant to the audit? Can one design a scheme where verifiers can further "maul" authentication tokens so that some of the disclosed attributes are blinded, yet keeping the audit token verifiable under the issuer's public key?

Trivial Constructions. Current PABC schemes do not allow for such functionality, or at least not efficiently. Presentation tokens usually consist of non-malleable non-interactive zero-knowledge proofs. In theory, one can always rely on generic zero-knowledge techniques [6] to prove knowledge of a valid presentation token for a subset of the disclosed attributes, but such proofs will be prohibitively expensive in practice. If the number of disclosed attributes is small, or the combination of attributes required by the auditor is known upfront, the user can generate multiple separate presentation tokens, one for the verifier and one for each of the auditors. This solution does not scale, however: if there are m disclosed attributes and the audited combination is not known upfront, the user would have to prepare 2^m presentation tokens.

Our Contributions. We present an efficiently auditable PABC scheme, meaning the size of authentication tokens as well as audit tokens stays linear in the number of attributes. Just like many PABC schemes, credentials in our construction are signatures on blocks of messages, where each message block encodes an attribute value. A presentation token is computed with an efficient signature of knowledge [7] of a valid credential that reveals only part of the message blocks. The basic idea of our construction is that, rather than simply revealing the disclosed attribute values, the user commits to them and creates a signature of knowledge of a valid credential for the same values as those that he committed to. The opening information of all commitments is handed to the verifier, who can check that they contain the claimed attribute values, but in the auditing phase, the verifier only forwards the opening information of the transferred attributes to the auditor, together with the user's original signature of knowledge.

We prove our construction secure in the universal composability (UC) framework [8], which guarantees that our protocol can be securely composed with itself as well as with other protocols in arbitrary environments. Given the several iterations that it took to define the security of basic signatures in this framework [9–11], defining security for a complicated primitive like ours is a delicate balancing act. We elaborately motivate our design choices for our ideal functionality in Section 3, in the hope that it can be of independent interest as a source of inspiration for future signature variants with privacy features.

Related Work. There are several proposals for dedicated signature schemes that allow the receiver of a signed message to reduce the amount of information in the message while retaining the ability to verify the corresponding signature.

Those are known as homomorphic [12], sanitizable [13–15], redactable [16], or content extracting signatures [17]. Other constructions, described e.g. in [18, 19] even allow more advanced operations on the signed data.

Those mechanisms do not yield straightforward constructions of our primitive as they only consider modifications of signed *messages*, whereas our scheme has to work with *presentation tokens* which itself are already derived from signed credentials. The crucial difference between signed messages and presentation tokens is that the latter should not be usable by a cheating verifier to impersonate the user at other verifiers. Therefore, the simple scheme where the credential and presentation token are redactable signatures on the list of attributes and where the presentation token can be further redacted by the verifier does not work.

Another related line of work conducts research on delegatable anonymous credentials [20], structure-preserving signatures [21], and commuting signatures [22]. The former allow credentials to be repetitively delegated while hiding the identity of the delegators. The latter two are more general signature schemes where the public key, the signed message, and the signature are all in the same mathematical group, and that among other things can be used to build delegatable credentials. Even though verifiable auditing is a sort of delegation, none of these primitives achieves the goals that we set out, as they cannot bind attributes to a delegatable credential.

2 System Overview

A privacy-preserving audit protocol consists of four parties: an auditor $\mathcal{R}$, an issuer $\mathcal{I}$, verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_J$, and users $\mathcal{U}_1, \ldots, \mathcal{U}_N$. The interaction between the parties is as follows. First, in the *issuing phase*, a user $\mathcal{U}_n$ gets credentials that certify her attributes from the issuer $\mathcal{I}$. A credential consists of L attributes $(a_1, \ldots, a_L)$. In the *presentation phase*, $\mathcal{U}_n$ sends a presentation token to a verifier $\mathcal{V}_j$. In each presentation token, $\mathcal{U}_n$ chooses which attributes are revealed to $\mathcal{V}_j$ and, moreover, which of those attributes can further be revealed to the auditor $\mathcal{R}$. The indexes of the attributes that are only revealed to $\mathcal{V}_j$ are included in a set F, and the indexes of the attributes that are revealed to $\mathcal{V}_j$ and that can also be revealed to $\mathcal{R}$ are included in a set D. We call the attributes given by D *transferable*, while the ones given by F are *non-transferable*. In the *audit phase*, $\mathcal{V}_j$ reveals to $\mathcal{R}$ (a subset of) the transferable attributes, whose indexes are included in a subset T such that $T \subseteq D$.

3 Security Definition of Privacy-Preserving Audits

3.1 Universally Composable Security

The universal composability framework [23] is a general framework for analyzing the security of cryptographic protocols in arbitrary composition with other protocols. The security of a protocol φ is analyzed by comparing the view of an environment $\mathcal{Z}$ in a real execution of φ against that of $\mathcal{Z}$ when interacting with

an ideal functionality $\mathcal{F}$ that carries out the desired task. The environment $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. In the real world, $\mathcal{Z}$ can communicate freely with an adversary $\mathcal{A}$ who controls the network as well as any corrupt parties. In the ideal world, $\mathcal{Z}$ interacts with dummy parties, who simply relay inputs and outputs between $\mathcal{Z}$ and $\mathcal{F}$, and a simulator $\mathcal{S}$. We say that a protocol φ securely realizes $\mathcal{F}$ if $\mathcal{Z}$ cannot distinguish the real world from the ideal world, i.e., $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and parties running protocol φ or with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}_\varphi$.

When describing ideal functionalities, we use the following conventions: The identity of an ITM instance (ITI) consists of a party identifier *pid* and a session identifier *sid*. An instance of $\mathcal{F}$ with session identifier *sid* only accepts inputs from and passes outputs to machines with the same session identifier *sid*. When describing functionalities, the expressions "output to $\mathcal{P}$" and "on input from $\mathcal{P}$", where $\mathcal{P}$ is a party identity *pid*, mean that the output is passed to and the input is received from party $\mathcal{P}$ only. Communication between ITIs with different party identifiers takes place over the network which is controlled by the adversary, meaning that he can arbitrarily delay, modify, drop, or insert messages.

When we say that $\mathcal{F}$ *sends* m *to* $\mathcal{S}$ *and waits for* m' *from* $\mathcal{S}$, we mean that $\mathcal{F}$ chooses a unique execution identifier, saves its current state, and sends m together with the identifier to $\mathcal{S}$. When $\mathcal{S}$ invokes a dedicated resume interface with a message m' and an execution identifier, $\mathcal{F}$ looks up the execution state associated to the identifier and continues running its program where it left off using m'.

Our protocol makes use of the standard functionalities $\mathcal{F}_{\text{REG}}$ [23] for key registration, $\mathcal{F}_{\text{SMT}}$ for secure message transmission [23], and $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ [23] for common reference strings with distribution $\mathcal{D}$. Descriptions and realizations of all these functionalities can be found in the literature.

We also use the non-standard anonymous secure message transmission functionality $\mathcal{F}_{\text{ASMT}}$ given in Figure 1. The literature provides a fair number of protocols that provide some form of anonymous communication. These include some onion routing protocols for which ideal functionalities have been defined [24, 25]. Their functionalities are quite complex, as they model the various imperfections of the protocols, in particular, what routing information an adversary learns. This information depends heavily on how messages are routed, how many other users currently use the channel, how many nodes are controlled by the adversary, etc. Indeed, the modeling and realizations of anonymous communication is an active field of research. Now, if we had used one of these functionalities for our protocols, we would have had to model all these imperfections in our ideal functionality $\mathcal{F}_{\text{AUD}}$ as well. We consider such modeling orthogonal to our protocol and our goals and therefore choose to assume ideal anonymous communication where the adversary only learns that some message is sent (and is allowed to deny its delivery) but does not learn the identities of the sender and the receiver.

Functionality $\mathcal{F}_{\text{ASMT}}$

Parameterized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{F}_{\text{ASMT}}$ works as follows:

1. On input (send, sid, m) from a party $\mathcal{T}$, execute the following program:
   - If $sid \neq (\mathcal{R}, sid')$, exit the program.
   - Send $(\mathsf{send}, sid', l(m))$ to $\mathcal{S}$.
   - Wait for a message (send, sid') from $\mathcal{S}$.
   - Send (sent, sid, m) to $\mathcal{R}$.

Fig. 1. The ideal functionality of anonymous secure message transmission

3.2 Ideal Functionality of Privacy-Preserving Audits

We describe our ideal functionality $\mathcal{F}_{\text{AUD}}$ of privacy-preserving audits in Figure 2. We assume static corruptions, meaning that the adversary decides which parties to corrupt at the beginning of the game but cannot corrupt additional parties once the protocol is running. $\mathcal{F}_{\text{AUD}}$ employs the following tables:

Table 1. Tbl_1 stores entries of the form $[\mathcal{U}_n, \langle a_l \rangle_{l=1}^{L}]$ associating a user $\mathcal{U}_n$ to her attributes $\langle a_l \rangle_{l=1}^{L}$, or of the form $[\mathcal{S}, \langle a_l \rangle_{l=1}^{L}]$ if the credential was issued to a corrupt user.

Table 2. Tbl_2 stores entries of the form $[\mathcal{V}_j, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid]$ associating verifiers $\mathcal{V}_j$ to the information of previous presentation phases.

Table 3. Tbl_3 stores entries of the form $[audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, v]$, associating audit tokens to the data used to compute or verify the token, plus $v \in \{\mathtt{valid}, \mathtt{invalid}\}$ indicating whether the token is valid.

To shorten the description, we assume that the functionality proceeds only if the incoming messages are well-formed. That is, for presentation, audit, and verify messages, $\mathcal{F}_{\text{AUD}}$ continues only if $D \cap F = \emptyset$, $D \subseteq [1, L]$, $F \subseteq [1, L]$, and $sid = (\mathcal{I}, sid')$ holds. For the audit and verify messages, $\mathcal{F}_{\text{AUD}}$ additionally checks that $T \subseteq D$.

The functionality assumes certified identities of all users, verifiers, and the signer (cf. the discussion on public keys below). We now discuss the four interfaces of the ideal functionality $\mathcal{F}_{\text{AUD}}$ given in Figure 2.

The issue interface is called by the issuer with the user identity and the attributes in the to-be-issued credential, meaning that the issuer is aware of which attributes he issues to which user. The simulator indicates when the issuance is to be finalized by sending a (issue, sid) message. At this point, the issuance is recorded in Table 1. If the user is honest, the issuance is recorded under the correct user's identity; any instantiating protocol will have to set up an authenticated channel to the user to ensure this in the real world. If the user is corrupt, the credential is recorded as belonging to the simulator, modeling that corrupt users may pool their credentials. Note that the simulator is not given the issued attribute values, so the real-world protocol must hide these from the adversary.

The $\mathsf{present}$ interface lets a user $\mathcal{U}_n$ present a subset of attributes to a verifier $\mathcal{V}_j$. Honest users can only show combinations of attributes that appear in

Functionality $\mathcal{F}_{\text{AUD}}$

1. On input $(\mathsf{issue}, sid, \mathcal{U}_n, \langle a_l \rangle_{l=1}^{L})$ from $\mathcal{I}$:
   - If $sid \neq (\mathcal{I}, sid')$ then exit the program.
   - Send $(\mathsf{issue}, sid, \mathcal{U}_n)$ to $\mathcal{S}$ and wait for (issue, sid) from $\mathcal{S}$.
   - If $\mathcal{U}_n$ is honest then store $[\mathcal{U}_n, \langle a_l \rangle_{l=1}^{L}]$ in Tbl_1, else store $[\mathcal{S}, \langle a_l \rangle_{l=1}^{L}]$.
   - Output $(\mathsf{issue}, sid, \langle a_l \rangle_{l=1}^{L})$ to $\mathcal{U}_n$.

2. On input $(\mathsf{present}, sid, \mathcal{V}_j, D, F, \langle a_l \rangle_{l \in D \cup F}, msg)$ from $\mathcal{U}_n$:
   - Continue only if one of the following conditions is satisfied:
     - there exist $[\mathcal{U}'_n, \langle a'_l \rangle_{l=1}^{L}] \in \mathsf{Tbl}_1$ s.t. $a'_l = a_l \ \forall l \in D \cup F$ where $\mathcal{U}'_n = \mathcal{U}_n$ if $\mathcal{U}_n$ is an honest user, or $\mathcal{U}'_n = \mathcal{S}$ if $\mathcal{U}_n$ is corrupt,
     - $\mathcal{U}_n$ and $\mathcal{I}$ are both corrupt.
   - Send $(\mathsf{present}, sid, \mathcal{V}_j)$ to $\mathcal{S}$ and wait for $(\mathsf{present}, sid)$ from $\mathcal{S}$.
   - Increment the token identifier $tid(\mathcal{V}_j) = tid(\mathcal{V}_j) + 1$.
   - Store $[\mathcal{V}_j, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid(\mathcal{V}_j)]$ in Tbl_2.
   - Output $(\mathsf{tokrec}, sid, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid(\mathcal{V}_j))$ to $\mathcal{V}_j$.

3. On input $(\mathsf{auditgen}, sid, D, F, T, \langle a_l \rangle_{l \in T}, msg, tid)$ from $\mathcal{V}_j$:
   - Continue only if one of the following conditions is satisfied:
     - $\mathcal{V}_j$ and $\mathcal{I}$ are both corrupt,
     - $\mathcal{V}_j$ is corrupt and there exists $[\mathcal{S}, \langle a'_l \rangle_{l=1}^{L}] \in \mathsf{Tbl}_1$ s.t. $a'_l = a_l \ \forall l \in T$,
     - there exist $[\mathcal{V}_j, D, F, \langle a''_l \rangle_{l \in D \cup F}, msg, tid] \in \mathsf{Tbl}_2$ s.t. $a''_l = a_l \ \forall l \in T$.
   - Send the message $(\mathsf{auditgen}, sid, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, tid)$ to $\mathcal{S}$ and wait for $(\mathsf{auditgen}, sid, audtok)$ from $\mathcal{S}$.
   - Store $[audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, \mathtt{valid}]$ in Tbl_3.
   - Output $(\mathsf{audrec}, sid, audtok)$ to $\mathcal{V}_j$.

4. On input $(\mathsf{auditvf}, sid, audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg)$ from a party $\mathcal{P}$:
   - Send $(\mathsf{auditvf}, sid, audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg)$ to $\mathcal{S}$ and wait for $(\mathsf{auditvf}, sid, w)$ from $\mathcal{S}$.
   - If there exists $[audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, u] \in \mathsf{Tbl}_3$ then set $v = u$. *(Completeness/Consistency)*
   - Else, set $v = w$ only if one of the following conditions is satisfied *(Unforgeability)*:
     - $\mathcal{V}_j$ and $\mathcal{I}$ are both corrupt,
     - $\mathcal{V}_j$ is corrupt and there exists $[\mathcal{S}, \langle a'_l \rangle_{l=1}^{L}] \in \mathsf{Tbl}_1$ or $[\mathcal{V}_j, D, F, \langle a'_l \rangle_{l \in D \cup F}, msg, tid] \in \mathsf{Tbl}_2$ such that $a'_l = a_l \ \forall l \in T$,
     - there exists $[audtok', \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, \mathtt{valid}] \in \mathsf{Tbl}_3$.
   - Otherwise, set $v = \mathtt{invalid}$.
   - Store $[audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg, v]$ in Tbl_3.
   - Output (audvf, sid, v) to $\mathcal{P}$.

Fig. 2. The ideal functionality $\mathcal{F}_{\text{AUD}}$ and its four interfaces

a credential issued to that user. If the issuer is honest, but the user is corrupt, then the presented attributes must be part of a credential issued to some corrupt user, not necessarily $\mathcal{U}_n$ itself. Upon receiving a message $(\mathsf{present}, sid)$ from $\mathcal{S}$, the presented attributes and the associated message are recorded in Table 2. The table also contains the identity of the verifier to whom the attributes were presented. Finally, the verifier is informed about the revealed attributes and the message. Note that neither the verifier nor the simulator learns the identity of the user who initiated the presentation protocol, which guarantees that presentation protocols are anonymous. Of course, one requires some form of anonymous communication between the user and the verifier to achieve this.

To generate and verify audit tokens, $\mathcal{F}_{\text{AUD}}$ offers two interfaces: the auditgen interface to create audit tokens, and the auditvf to verify audit tokens. Honest verifiers can only create audit tokens that can be derived from presentations that they have seen, as recorded in Table 2. If the verifier is corrupt, but the issuer is honest, the verifier can additionally create tokens that can be derived from any credentials issued to a corrupt user, as recorded in Table 1. If the verifier and the issuer are both corrupt, then the adversary can generate any audit tokens that he wants. Unlike credentials and presentations, audit tokens have an actual bit string representations in our functionality that can be verified by anyone, not just by a dedicated auditor. We follow Canetti's signature functionality [11] by letting the simulator determine the value of the audit token. Note that the simulator is only given the values of the transferred attributes T, which guarantees that audit tokens do not reveal any information about the non-transferred attributes. The functionality registers the token as valid in Table 3 and returns it to the verifier.

Any party can verify an audit token through the auditvf interface. The functionality enforces consistency through Table 3, guaranteeing that verification of the same audit token for the same parameters always returns the same result. Note that this also enforces completeness, i.e., that honestly generated tokens verify correctly, because honestly generated tokens are recorded in Table 3 as valid. When the issuer is honest, the functionality enforces unforgeability by rejecting all audit tokens that the adversary should not be able to create. If in the real world the adversary manages to come up with a valid forgery, then the environment will be able to notice a difference between the real and the ideal world by verifying the token. Tokens are considered forgeries when they could not have been derived from any credentials issued to corrupt users in Table 1, from any presentation to a corrupt verifier $\mathcal{V}_j$ in Table 2, or from any honestly generated audit tokens in Table 3. Note that in the latter condition, the honestly generated token $audtok'$ may be different from the verified token $audtok$. This models conventional (i.e., non-strong) unforgeability: if the environment previously obtained any token that is valid for the same parameters, then the current token is no longer considered a forgery.

Public Keys. We define our functionality $\mathcal{F}_{\text{AUD}}$ so that, rather than providing a binding to the public key of the issuer, audit tokens provide a direct binding to the issuer's identity, much like Canetti's certified signature functionality $\mathcal{F}_{\text{CERT}}$

provides a direct binding to the signer's identity [11]. Similarly, presentation protocols are bound directly to verifiers' identities rather than their public keys. This greatly simplifies the description of our functionality because we do not have to model public keys of issuers and verifiers, and we do not have to specify how the various interfaces behave when called with incorrect public keys. Indeed, when tokens are bound directly to party identities, public keys become an implementation detail of the protocol. This of course comes at a price: in order to satisfy the functionality, our protocol must rely on an underlying public-key infrastructure to bind public keys to party identities.

Session Identifiers. The restriction that the issuer's identity $\mathcal{I}$ must be included in the session identifier $sid = (\mathcal{I}, sid')$ guarantees that each issuer can initialize its own instance of the functionality. In applications where the issuer is to remain anonymous, the issuer identity could be replaced with a unique pseudonym.

Representations of Credentials, Presentations, and Audit Tokens. The issuing phase depicted in Figure 2 does not expose any bit string representation for credentials to the environment, but merely records which attributes are issued to which user. Just like public keys, credentials are thereby reduced to implementation details that remain internal to the state of honest parties. Unlike public keys, however, this is not just an easy way to simplify our functionality, but is actually crucial to the unforgeability guarantee. Namely, our functionality imposes unforgeability of audit tokens by letting the verification interface reject tokens that the environment should not have been able to produce, including tokens that could have been derived from honest users' credentials, but not from corrupt users' credentials. However, if the functionality were to output actual credentials to honest users, the environment could itself derive valid audit tokens from these credentials, which the functionality would have to accept. Similarly, the presentation phase in Figure 2 merely records which combinations of attributes were shown to which verifier, without exposing a cryptographic token of that presentation to the environment.

Linkability of Audit Tokens. An audit token can be linked to the presentation token from which it was computed. For each verifier $\mathcal{V}_j$, each presentation phase is given a unique identifier $tid(\mathcal{V}_j)$, and this identifier is passed to the functionality when creating an audit token through the auditgen interface. The functionality also passes $tid(\mathcal{V}_j)$ to the simulator when it is asked to create the actual token, so that the simulator can create an audit token that respects the linkability to the corresponding presentation token. The simulator still does not get any information about the non-transferred attributes, however.

4 Preliminaries

This section describes the cryptographic primitives our realization of $\mathcal{F}_{\text{AUD}}$ uses.

4.1 Trapdoor Commitment Schemes

A non-interactive commitment scheme consists of algorithms CSetup, Com and VfCom. $\mathsf{CSetup}(1^k)$ generates the parameters of the commitment scheme par_c, which include a description of the message space $\mathcal{M}$. $\mathsf{Com}(par_c, x)$ outputs a commitment com to x and auxiliary information $open$. A commitment is opened by revealing $(x, open)$ and checking whether $\mathsf{VfCom}(par_c, com, x, open)$ outputs 1 or 0. A commitment scheme should fulfill the correctness, hiding and binding properties. Correctness requires that VfCom accepts all commitments created by algorithm Com, i.e., for all $x \in \mathcal{M}$

$$\Pr\left[\begin{array}{c} par_c \leftarrow \mathsf{CSetup}(1^k);\ (com, open) \leftarrow \mathsf{Com}(par_c, x) : \\ 1 = \mathsf{VfCom}(par_c, com, x, open) \end{array}\right] = 1\ .$$

The hiding property ensures that a commitment com to x does not reveal any information about x, whereas the binding property ensures that com cannot be opened to another value x'.

Definition 1 (Hiding Property). *For any PPT adversary $\mathcal{A}$, the hiding property is defined as follows:*

$$Pr\left[\begin{array}{l} par_c \leftarrow \mathsf{CSetup}(1^k); \\ (x_0, x_1, st) \leftarrow \mathcal{A}(par_c); \\ b \leftarrow \{0,1\};\ (com, open) \leftarrow \mathsf{Com}(par_c, x_b); \\ b' \leftarrow \mathcal{A}(st, com) : \\ x_0 \in \mathcal{M}\ \wedge\ x_1 \in \mathcal{M}\ \wedge\ b = b' \end{array}\right] \leq \frac{1}{2} + \epsilon(k)\ .$$

Definition 2 (Binding Property). *For any PPT adversary $\mathcal{A}$, the binding property is defined as follows:*

$$Pr\left[\begin{array}{l} par_c \leftarrow \mathsf{CSetup}(1^k);\ (com, x, open, x', open') \leftarrow \mathcal{A}(par_c) : \\ x \in \mathcal{M}\ \wedge\ x' \in \mathcal{M}\ \wedge x \neq x' \wedge\ 1 = \mathsf{VfCom}(par_c, com, x, open) \\ \wedge\ 1 = \mathsf{VfCom}(par_c, com, x', open') \end{array}\right] \leq \epsilon(k)\ .$$

A trapdoor commitment scheme [26, 27] is a commitment scheme where there exists trapdoor information that allows to open commitments to any value.

Definition 3 (Trapdoor Property). *There exist polynomial-time algorithms* $\mathsf{CSimSetup}$ *and* $\mathsf{ComOpen}$*, where* $\mathsf{CSimSetup}$ *on input* 1^k *outputs parameters* par_c *with trapdoor* td_c *such that: (1)* par_c *are indistinguishable from those produced by* CSetup*, and (2) for any* $x, x' \in \mathcal{M}$

$$\left|Pr\left[\begin{array}{c} (par_c, td_c) \leftarrow \mathsf{CSimSetup}(1^k);\ (com, open') \leftarrow \mathsf{Com}(par_c, x'); \\ open \leftarrow \mathsf{ComOpen}(par_c, td_c, x, x', open') :\ 1 = \mathcal{A}(par_c, td_c, com, open) \end{array}\right]\right.$$
$$\left. -\ Pr\left[\begin{array}{c} (par_c, td_c) \leftarrow \mathsf{CSimSetup}(1^k); (com, open) \leftarrow \mathsf{Com}(par_c, x) : \\ 1 = \mathcal{A}(par_c, td_c, com, open) \end{array}\right]\right| \leq \epsilon(k)\,.$$

4.2 Signature Schemes

A signature scheme consists of the algorithms KeyGen, Sign, and VfSig. Algorithm $\mathsf{KeyGen}(1^k)$ outputs a secret key sk and a public key pk, which include a description of the message space $\mathcal{M}$. $\mathsf{Sign}(sk, m)$ outputs a signature s on message $m \in \mathcal{M}$. $\mathsf{VfSig}(pk, s, m)$ outputs 1 if s is a valid signature on m and 0 otherwise. This definition can be extended to blocks of messages $\bar{m} = (m_1, \ldots, m_n)$. A signature scheme must fulfill the correctness and existential unforgeability properties [28].

4.3 Signatures of Knowledge

Let $\mathcal{L}$ be an NP language defined by a polynomial-time computable relation R as $\mathcal{L} = \{x | \exists w : (x, w) \in R\}$. We call x a statement in the language $\mathcal{L}$ and w with $(x, w) \in R$ a witness for x. A signature of knowledge (SK) [29, 7] for $\mathcal{L}$ consists of the following algorithms:

$\mathsf{SKSetup}(1^k)$**.** Output parameters par_s, which include a description of the message space $\mathcal{M}$.

$\mathsf{SKSign}(par_s, R, x, w, m)$**.** If $(x, w) \in R$, output a signature of knowledge σ on the message m with respect to statement x, else output $\perp$.

$\mathsf{SKVerify}(par_s, R, x, m, \sigma)$**.** If σ is a valid signature of knowledge on the message m with respect to statement x, output 1, else output 0.

Correctness ensures that $\mathsf{SKVerify}$ accepts the signatures of knowledge generated by SKSign. More formally, for any $(x, w) \in R$ and any $m \in \mathcal{M}$, we require

$$\Pr\left[\begin{array}{c} par_s \leftarrow \mathsf{SKSetup}(1^k);\ \sigma \leftarrow \mathsf{SKSign}(par_s, R, x, w, m) : \\ 1 = \mathsf{SKVerify}(par_s, R, x, m, \sigma) \end{array}\right] = 1\ .$$

To obtain efficient instantiations in the random-oracle model, we adopt the signature of knowledge definitions of Benhamouda et al. [36], which are a random-oracle adaptation of those of Chase and Lysyanskaya [7].

Definition 4 (Simulatability and Extractibility). *There exists a stateful simulation algorithm* SKSim *that can be called in three modes. When called as* $(par_s, st) \leftarrow \mathsf{SKSim}(\mathtt{setup}, 1^k)$*, it produces simulated parameters* par_s*, possibly keeping a trapdoor in its internal state* st*. When run as* $(h, st') \leftarrow \mathsf{SKSim}(\mathtt{ro}, q, st)$*, it produces a response* h *for a random oracle query* q*. When run as* $(\sigma, st') \leftarrow \mathsf{SKSim}(\mathtt{sign}, R, x, m, st)$*, it produces a simulated signature of knowledge* σ *without using a witness.*

For ease of notation, let $\mathsf{SKSimRO}(q)$ *be an oracle that returns the first part of* $\mathsf{SKSim}(\mathtt{ro}, q, st)$ *and let* $\mathsf{SKSimSign}(R, x, w, m)$ *be an oracle that returns the first part of* $\mathsf{SKSim}(\mathtt{sign}, R, x, m, st)$ *if* $(x, w) \in R$ *and returns* $\perp$ *otherwise, while a synchronized state is kept for* SKSim *across invocations.* H *denotes a hash function, which is modeled as a random oracle. The algorithms satisfy the following properties:*

- Simulatability*: No adversary can distinguish whether it is interacting with a real random oracle and signing oracle, or with their simulated versions. Formally, for all PPT $\mathcal{A}$ there exists a negligible function ϵ such that*

$$\big|\Pr[(par_s, st) \leftarrow \mathsf{SKSim}(\mathtt{setup}, 1^k), b \leftarrow \mathcal{A}^{\mathsf{SKSimRO},\mathsf{SKSimSign}}(par_s) : b = 1] - \Pr[par_s \leftarrow \mathsf{SKSetup}(1^k), b \leftarrow \mathcal{A}^{H,\mathsf{SKSign}}(par_s) : b = 1]\big| \leq \epsilon(k) \ .$$

- Extractability*: The only way to produce a valid signature of knowledge is by knowing a witness. Formally, for all PPT $\mathcal{A}$ there exists an extractor $\mathsf{SKExt}_{\mathcal{A}}$ and a non-negligible function p such that*

$$\Pr\left[\begin{array}{c}(par_s, st) \leftarrow \mathsf{SKSim}(\mathtt{setup}, 1^k),\\ (R, x, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{SKSimRO},\mathsf{SKSimSign}}(par_s; \rho_{\mathcal{A}}),\\ w \leftarrow \mathsf{SKExt}_{\mathcal{A}}(par_s, R, x, m, \sigma, \rho_{\mathsf{S}}, \rho_{\mathcal{A}}) :\\ \mathsf{SKVerify}(par_s, R, x, m, \sigma) = 1 \wedge (R, x, m) \notin Q \wedge (x, w) \notin R\end{array}\right] \geq \epsilon(k) \ ,$$

where Q is the set of queries (R, x, m) that $\mathcal{A}$ submitted to its $\mathsf{SKSimSign}$ oracle, and where ρ_{S} and $\rho_{\mathcal{A}}$ are the random tapes of the simulator and the adversary, respectively.

5 Construction of Privacy-Preserving Audits

The high-level idea of our protocol is as follows: a user $\mathcal{U}_n$ can obtain credentials from an issuer $\mathcal{I}$, where credentials are signed sets of attributes. From a credential the user can subsequently derive a presentation token which discloses attributes a_l for $l \in D$ in a transferable way to the verifier, and attributes a_l for $l \in F$ in a non-transferable way. To this end, the user first creates a commitment and opening $(com_l, open_l)$ for each disclosed attribute a_l with $l \in D \cup F$. He then generates a signature of knowledge σ, proving that he has a valid credential for all the committed values. To further ensure that the signature cannot be used in a different context, e.g., by a malicious party trying to impersonate an honest user, the signature signs a message which contains the verifier identifier and a fresh *nonce* chosen by the user. The entire presentation token then consists of the signature of knowledge σ, the commitments $\langle com_l \rangle_{l \in D \cup F}$ and openings $\langle open_l \rangle_{l \in D \cup F}$ for all disclosed attributes, and the random *nonce*.

The verifier $\mathcal{V}_j$ can check the correctness of such a token by verifying the signature of knowledge and verifying whether the commitments com_l open to the correct values a_l for all $l \in D \cup F$. If that is the case, the verifier stores the token and will not accept any further token that signs the same *nonce*.

When the verifier wants to derive an audit token from the presentation token and to transfer attributes $T \subseteq D$ to the auditor, he simply reuses the presentation token with the modification that he only includes the openings for the subset of transferred attributes into the audit token. The verifier further adds a signature s, where he signs the redacted presentation token with his own signing key. This ensures that a malicious user cannot bypass an honest verifier and directly create an audit token by himself.

An auditor can verify an audit token by verifying the correctness of the forwarded signature of knowledge σ, the correct opening of all commitments for the transferred attributes and the verifier's signature s.

5.1 Our Realization of $\mathcal{F}_{\text{AUD}}$

Our protocol uses a trapdoor commitment scheme $(\mathsf{CSetup}, \mathsf{Com}, \mathsf{VfCom})$, and two signature schemes, one for the issuer $(\mathsf{KeyGen}_{\mathcal{I}}, \mathsf{Sign}_{\mathcal{I}}, \mathsf{VfSig}_{\mathcal{I}})$ and one for the verifier $(\mathsf{KeyGen}_{\mathcal{V}}, \mathsf{Sign}_{\mathcal{V}}, \mathsf{VfSig}_{\mathcal{V}})$. Both signature schemes follow the standard signature definition given in Section 4.2 and can be instantiated with the same construction. However, as the issuer's signature also serves as witness in a signature of knowledge scheme $(\mathsf{SKSetup}, \mathsf{SKSign}, \mathsf{SKVerify})$ it might be beneficial to choose a signature scheme for $(\mathsf{KeyGen}_{\mathcal{I}}, \mathsf{Sign}_{\mathcal{I}}, \mathsf{VfSig}_{\mathcal{I}})$ that already comes with efficient protocols for such proofs. Furthermore, the issuer's signature scheme must allow signing blocks of messages, whereas for the verifier's scheme only a single message needs to be signed.

For simplicity, it is assumed that all issuers and verifiers in the scheme have registered public keys. That is, the issuer and verifiers generate their signing keys as $(ipk, isk) \leftarrow \mathsf{KeyGen}_{\mathcal{I}}(1^k)$ and $(vpk_j, vsk_j) \leftarrow \mathsf{KeyGen}_{\mathcal{V}}(1^k)$ respectively and register their public keys with $\mathcal{F}_{\text{REG}}$. We further assume that all parties fetch the necessary parameters and public keys by invoking the corresponding functionalities. That is, the system parameters (par_s, par_c) with $par_s \leftarrow \mathsf{SKSetup}(1^k)$ and $par_c \leftarrow \mathsf{CSetup}(1^k)$ are obtained via $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, and the public keys of the verifiers and issuer can be retrieved via the $\mathcal{F}_{\text{REG}}$ functionality. Note that the issuer identity $\mathcal{I}$ is part of the session identifier $sid = (\mathcal{I}, sid')$ that is contained in every message. Each verifier maintains a list of nonces L_{nonce} which is initially set to $\mathsf{L}_{nonce} := \emptyset$ and will be filled with nonces of verified presentation tokens, which is used to guarantee a one-time showing for each token. The communication between the different parties is done over ideal functionalities $\mathcal{F}_{\text{SMT}}$ in the issuing phase and $\mathcal{F}_{\text{ASMT}}$ in the presentation phase.

As in the ideal functionality $\mathcal{F}_{\text{AUD}}$, the parties in our protocol only proceed if the incoming messages are well-formed, i.e., for the presentation, audit and verify messages the respective party only continues if $D \cap F = \emptyset$, $D \subseteq [1, L]$, $F \subseteq [1, L]$ and $sid = (\mathcal{I}, sid')$. For the audit and verify messages, the verifier and auditor further check that $T \subseteq D$.

Issuance. On input $(\mathsf{issue}, sid, \mathcal{U}_n, \langle a_l \rangle_{l=1}^{L})$ where $sid = (\mathcal{I}, sid')$, the issuer $\mathcal{I}$ executes the following program with the user $\mathcal{U}_n$:

Step I1 – Issuer $\mathcal{I}$ generates and sends credential:

a) Generate credential as $cred \leftarrow \mathsf{Sign}_{\mathcal{I}}(isk, \langle a_l \rangle_{l=1}^{L})$.
b) Set $sid_{\text{SMT}} := (\mathcal{U}_n, sid, sid'')$ for a fresh sid'' and send $(\mathsf{send}, sid_{\text{SMT}}, (\langle a_l \rangle_{l=1}^{L}, cred))$ to $\mathcal{F}_{\text{SMT}}$.

Step I2 – User $\mathcal{U}_n$ verifies and stores credential:

a) Upon receiving $(\mathsf{sent}, sid_{\text{SMT}}, (\langle a_l \rangle_{l=1}^{L}, cred))$ from $\mathcal{F}_{\text{SMT}}$, verify that $1 \leftarrow \mathsf{VfSig}_{\mathcal{I}}(ipk, cred, \langle a_l \rangle_{l=1}^{L})$ and abort if the verification fails.

b) Store $(\langle a_l \rangle_{l=1}^{L}, cred)$ and output $(\mathsf{issue}, sid, \langle a_l \rangle_{l=1}^{L})$.

Presentation. On input $(\mathsf{present}, sid, \mathcal{V}_j, D, F, \langle a_l \rangle_{l \in D \cup F}, msg)$, the user $\mathcal{U}_n$ executes the following program with verifier $\mathcal{V}_j$.

Step P1 – User $\mathcal{U}_n$ creates a presentation token:

a) Retrieve the credential $(\langle a'_l \rangle_{l=1}^{L}, cred)$ where $a'_l = a_l$ for all $l \in D \cup F$. Abort if no such credential exists.
b) Create a signature of knowledge of a valid credential w.r.t. committed attributes and bound to a nonce:
   - Compute $(com_l, open_l) \leftarrow \mathsf{Com}(par_c, a_l)\ \forall l \in D \cup F$.
   - Choose a random nonce $nonce \in \{0,1\}^k$ and set $m := (msg, \mathcal{V}_j, nonce)$.
   - Prepare a signature of knowledge for the statement that a valid credential is known which contains the same attribute values as the commitments. That is, set the relation to $R :=$
   $$(1 = \mathsf{VfSig}_{\mathcal{I}}(ipk, cred, \langle a_l \rangle_{l=1}^{L}) \quad \wedge \quad 1 = \mathsf{VfCom}(par_c, a_l, com_l, open_l)\ \forall l \in D \cup F),$$
   and set the statement and witness to $x := (ipk, \langle com_l \rangle_{l \in D \cup F}, par_c, D, F)$, $w := (cred, \langle a_l \rangle_{l=1}^{L}, \langle open_l \rangle_{l \in D \cup F})$.
   - Generate the signature of knowledge as $\sigma \leftarrow \mathsf{SKSign}(par_s, R, x, w, m)$.
c) Compose and send the presentation token:
   - Set $sid_{\text{ASMT}} := (\mathcal{V}_j, sid, sid'')$ for a fresh sid'' and send $(\mathsf{send}, sid_{\text{ASMT}}, (\langle a_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma))$ to $\mathcal{F}_{\text{ASMT}}$.

Step P2 – Verifier $\mathcal{V}_j$ verifies the presentation token:

a) Upon receiving a message given by $(\mathsf{sent}, sid_{\text{ASMT}}, (\langle a_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma))$ from $\mathcal{F}_{\text{ASMT}}$, verify that $nonce \notin \mathsf{L}_{nonce}$ and abort otherwise.
b) Verify signature of knowledge and commitments:
   - Set the tuple (R, x, m) similarly as in Step P1(b) and check that $1 = \mathsf{SKVerify}(par_s, R, x, m, \sigma)$.
   - Verify that $1 = \mathsf{VfCom}(par_c, a_l, com_l, open_l)$ for all $l \in D \cup F$. Abort if a verification fails.
c) Store token and nonce and end:
   - Set the token-identifier to $tid := tid + 1$ and $\mathsf{L}_{nonce} := \mathsf{L}_{nonce} \cup nonce$.
   - Store $(\langle a_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma, tid)$.
   - Output $(\mathsf{tokrec}, sid, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid)$.

Audit Token Generation. On input $(\mathsf{auditgen}, sid, D, F, T, \langle a_l \rangle_{l \in T}, msg, tid)$, the verifier $\mathcal{V}_j$ executes the following program.

a) Retrieve the tuple $(\langle a'_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma, tid)$ such that $a'_l = a_l$ for all $l \in T$. Abort if no such tuple exists.
b) Sign the redacted token information as
$$s \leftarrow \mathsf{Sign}_{\mathcal{V}}(vsk_j, (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, T)).$$
c) Set the audit token to $audtok := (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, nonce, s)$ and end with output $(\mathsf{audrec}, sid, audtok)$.

Audit Token Verification. On input the message $(\mathsf{auditvf}, sid, audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg)$, the auditor $\mathcal{R}$ executes the following program. Whenever a verification step fails, the auditor ends with output $(\mathsf{audvf}, sid, \mathtt{invalid})$.

a) Parse token as $audtok = (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, nonce, s)$.
b) Verify that $1 = \mathsf{VfSig}_{\mathcal{V}}(vpk_j, s, (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, T))$.
c) Set (R, x, m) as in Step P1(b) and verify that $1 = \mathsf{SKVerify}(par_s, R, x, m, \sigma)$.
d) Verify that $1 = \mathsf{VfCom}(par_c, a_l, com_l, open_l)$ for all $l \in T$.
e) If all checks succeeded, output $(\mathsf{audvf}, sid, \mathtt{valid})$.

5.2 Security Analysis

Our protocol is secure in the UC model based on the security properties of the underlying building blocks.

Theorem 1. *The above construction securely implements $\mathcal{F}_{\text{AUD}}$ in the $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, and $\mathcal{F}_{\text{ASMT}}$-hybrid model if the underlying trapdoor commitment scheme is hiding and binding, the underlying signature schemes are existentially unforgeable, and the signature of knowledge scheme is simulatable and extractable.*

A full proof of the above theorem is given in the full version of this paper [30]. Below, we summarize the sequence of games that leads to an indistinguishable simulation of our protocol.

Game 0**:** This is the original game where the simulator lets all honest parties run the real protocol based on the inputs from the environment and the network communication controlled by the adversary. The simulator also executes the code of the ideal functionalities $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, and $\mathcal{F}_{\text{ASMT}}$.

Game 1**:** When an honest verifier receives a presentation token from an honest user, the verifier skips the check that the nonce is fresh. This game deviates from the previous one whenever an honestly generated nonce collides with any previously seen nonce, which for a total of N received presentations happens with probability at most $N(N+1)/2^k$.

Game 2**:** This game maintains tables Tbl_1 with the credentials that were issued to corrupt users, Tbl_{2a} with the presentation tokens sent by honest users to a corrupt verifier, Tbl_{2b} with the presentation tokens received by honest verifiers from corrupt users, and Tbl_3 with the audit tokens created by honest verifiers. When an honest user computes a presentation token for an honest verifier $\mathcal{V}_j$, the simulator adds an entry of the form $[\bot, \ldots, \bot, tid(\mathcal{V}_j)]$ to Tbl_{2b}. These changes do not affect the environment's view at all.

Game 3**:** This game replaces the common parameters for the commitment and the signature of knowledge schemes with simulated parameters, so that the simulator knows the corresponding trapdoors. The change is indistinguishable by the simulatability of the signature of knowledge scheme and by the trapdoor property of the commitment scheme.

Game 4**:** Whenever an honest issuer sends a credential to an honest user, the simulated user submits a string of zeroes of the correct length to $\mathcal{F}_{\text{SMT}}$ instead

of a real credential. Likewise, an honest user submits a string of zeroes to $\mathcal{F}_{\text{ASMT}}$ when it performs a presentation to an honest verifier. Since $\mathcal{F}_{\text{SMT}}$ and $\mathcal{F}_{\text{ASMT}}$ only leak the length of the transmitted message, these changes do not affect the environment's view.

Game 5**:** When an honest user sends a presentation token to a corrupt verifier, the simulator uses SKSim to generate the signature of knowledge σ. Similarly, when an honest verifier is asked to compute an audit token from a presentation token that originates from an honest user and where Tbl_{2b} stores $[\bot, \ldots, \bot, tid]$, this game builds the audit token based on a simulated presentation token. That is, it simulates the signature of knowledge, computes the rest according to the protocol, and then updates the entry in Tbl_{2b} to contain σ as well as all commitments and openings. The indistinguishability follows from the simulatability of the signature of knowledge scheme.

Game 6**:** When an honest verifier computes an audit token from a presentation token that originates from an honest user but where a complete entry in Tbl_{2b} exists (i.e., another audit token based on the same presentation token was already produced), it uses the trapdoor of the commitment scheme to compute the opening, rather than using the *open* value stored in Tbl_{2b}. This game hop is indistinguishable by the trapdoor property of the commitment scheme.

Game 7**:** If an honest verifier is instructed to compute an audit token for a presentation token from an honest user and for which Tbl_{2b} stores $[\bot, \ldots, \bot, tid]$, this game replaces all commitments for values that are *not* disclosed with commitments to random values. Note that, by the previous game, commitments which are opened only in a subsequent audit token are opened to the required value using the trapdoor. This game is indistinguishable from the previous one by the hiding property of the commitment scheme.

Note that at this point, all presentations and audit tokens are simulated based only on information that is given to the adversary by $\mathcal{F}_{\text{AUD}}$.

Game 8**:** When a corrupt user sends a valid presentation token to an honest verifier for attribute values that were never issued to a corrupt user as recorded in Tbl_1, the simulator aborts. This game hop is indistinguishable if the issuer's signature scheme is unforgeable, the signature of knowledge is extractable, and the commitment scheme is binding.

Game 9**:** When any honest party is instructed to verify a valid audit token for an honest verifier $\mathcal{V}_j$ that the verifier never created, then abort. By the unforgeability of the verifier's signature scheme, this only happens with negligible probability.

Game 10**:** When any honest party is instructed to verify a valid audit token for a corrupt verifier $\mathcal{V}_j$ that cannot have been derived from credentials issued to honest users (as recorded in Tbl_1), nor from presentations by honest users to $\mathcal{V}_j$ (as recorded in Tbl_{2a}), then abort. This happens with negligible probability by the extractability of the signature of knowledge, the binding property of the commitment scheme, and the unforgeability of the issuer's signature scheme.

6 Instantiation of Privacy-Preserving Audits

We recall the Damgård-Fujisaki commitment scheme, which securely instantiates the algorithms $(\mathsf{CSetup}, \mathsf{Com}, \mathsf{VfCom}, \mathsf{CSimSetup}, \mathsf{ComOpen})$ described in Section 4.1 under the strong RSA assumption. Let l_n be the bit-length of the RSA modulus n and l_r be the bit-length of a further security parameter, both are functions of k. Typical values are $l_n = 2048$ and $l_r = 80$.

$\mathsf{CSetup}(1^k)$. Compute a safe RSA modulus $\tilde{n}$ of length l_n, i.e., such that $\tilde{n} = pq$, $p = 2p' + 1$, $q = 2q' + 1$, where p, q, p', and q' are primes. Pick a random generator $h \in QR_{\tilde{n}}$ and random $\alpha \leftarrow \{0,1\}^{l_n + l_r}$ and compute $g \leftarrow h^\alpha$. Output the commitment parameters $par_c = (g, h, \tilde{n})$.

$\mathsf{Com}(par_c, x)$. Pick random $open \leftarrow \{0,1\}^{l_n + l_r}$, set $com \leftarrow g^x h^{open} \pmod{\tilde{n}}$, and output the commitment com and the auxiliary information $open$.

$\mathsf{VfCom}(par_c, com, x, open)$. On inputs x and $open$, compute $com' \leftarrow g^x h^{open} \bmod \tilde{n}$ and output 1 if $com = com'$ and 0 otherwise.

$\mathsf{CSimSetup}(1^k)$. Run CSetup and output par_c and α as trapdoor.

$\mathsf{ComOpen}(com, x_1, open_1, x_2, td_c)$. Compute $open_2 = open_1 + \alpha(x_1 - x_2)$.

We employ the Camenisch-Lysyanskaya signature scheme [31] to implement the issuer signature scheme $(\mathsf{KeyGen}_\mathcal{I}, \mathsf{Sign}_\mathcal{I}, \mathsf{VfSig}_\mathcal{I})$. This signature scheme is existentially unforgeable against adaptive chosen message attacks [28] under the strong RSA assumption.

Let ℓ_m, ℓ_e, ℓ_n, and ℓ_r be system parameters determined by a function of k, where ℓ_r is a security parameter and the meaning of the others will become clear soon. We denote the set of integers $\{-(2^{\ell_m} - 1), ..., (2^{\ell_m} - 1)\}$ by $\pm\{0,1\}^{\ell_m}$. Elements of this set can thus be encoded as binary strings of length ℓ_m plus an additional bit carrying the sign, i.e., $\ell_m + 1$ bits in total.

$\mathsf{KeyGen}_\mathcal{I}(1^k)$. On input 1^k, choose an ℓ_n-bit safe RSA modulus $n = pq$. Choose, uniformly at random, $R_1, \ldots, R_L, S, Z \in \mathrm{QR}_n$. Output the public key $(n, R_1, \ldots, R_L, S, Z)$ and the secret key $sk \leftarrow p$.

$\mathsf{Sign}_\mathcal{I}(sk, \langle m_1, \ldots, m_L \rangle)$. The message space is the set $\{(m_1, \ldots, m_L) : m_i \in \pm\{0,1\}^{\ell_m}\}$. On input $m_1, \ldots, m_L$, choose a random prime number e of length $\ell_e > \ell_m + 2$, and a random number v of length $\ell_v = \ell_n + \ell_m + \ell_r$. Compute $A \leftarrow (Z/(R_1^{m_1} \ldots R_L^{m_L} S^v))^{1/e} \bmod n$. Output the signature (e, A, v).

$\mathsf{VfSig}_\mathcal{I}(pk, s, \langle m_1, \ldots, m_L \rangle)$. To verify that the tuple (e, A, v) is a signature on message $\langle m_1, \ldots, m_L \rangle$, check that the statements $Z \equiv A^e R_1^{m_1} \ldots R_L^{m_L} S^v \pmod{n}$, $m_i \in \pm\{0,1\}^{\ell_m}$, and $2^{\ell_e} > e > 2^{\ell_e - 1}$ hold.

For the realization of signatures of knowledge we use generalized Schnorr proof protocols [33, 29, 34]. We describe how to instantiate the signature of knowledge scheme for the relation we require in our protocol, i.e., for

$$R := \{1 = \mathsf{VfSig}_\mathcal{I}(ipk, cred, \langle a_l \rangle_{l=1}^L) \quad \wedge \quad 1 = \mathsf{VfCom}(par_c, a_l, com_l, open_l)\ \forall l \in D \cup F\}.$$

with $x := (ipk, \langle com_l \rangle_{l \in D \cup F}, par_c, D, F)$ and $w := (cred, \langle a_l \rangle_{l=1}^L, \langle open_l \rangle_{\in D \cup F})$

where *cred* is a CL-signature (e, A, v) as defined above. It is a secure signature of knowledge in the random-oracle model under the strong RSA assumption (the proof is straightforward and is given in the full version of this paper [30]).

$\mathsf{SKSetup}(1^k)$**.** There are no separate system parameters for the signature of knowledge scheme.

$\mathsf{SKSign}(par_s, R, x, w, m)$**.** Randomize the credential (contained in w): choose random $v' \in_R \{0,1\}^{\ell_v}$ and compute $A' \leftarrow AS^{v'}$ and $v^* \leftarrow v - v'e$. Compute

$$\pi \leftarrow SPK\big\{(e, v^*, \langle a_l\rangle_{l=1}^{L}, \rangle o_l \langle_{l\in D\cup F}) : \bigwedge_{\forall l\in D\cup F} com_l = g^{a_l}h^{o_l} \pmod{\tilde{n}}$$

$$\wedge\ Z = A'^e R_1^{a_1} \ldots R_L^{a_L} S^{v^*} \pmod{n} \wedge a_i \in \pm\{0,1\}^{\ell_m} \wedge 2^{\ell_e} > e > 2^{\ell_e - 1}\big\}(m)$$

and output $\sigma \leftarrow (A', \pi)$. For the realization of the non-interactive proof of knowledge π we refer to Camenisch et al. [29, 34].

$\mathsf{SKVerify}(par_s, R, x, m, (A', \pi))$**.** This algorithm verifies if π is correct.

The signature of knowledge simulator $\mathsf{SKSimSign}$ will make use of the random oracle and the honest-verifier zero-knowledge property of the generalized Schnorr proofs. One can get rid of the random oracle with alternative techniques [35]. SKExt works by rewinding the adversary to obtain, with non-negligible probability, all attributes, the opening information of all commitments, and the credential (CL-signature).

Efficiency. We obtain a rough estimate of the efficiency of our protocol by counting only multi-exponentiations modulo n or $\tilde{n}$. Computation of a presentation for our protocol takes each of the user and the verifier $2(\#D + \#F) + L + 2$ multi-exponentiations modulo $\tilde{n}$. Auditing a token costs only a single standard signature, while verifying an audit token involves one standard signature verification and $\#D + \#F + \#T + L + 2$ multi-exponentiations modulo $\tilde{n}$. In terms of bandwidth, a presentation consumes, apart from the length of the revealed attributes, $2\ell_n + \ell_m + \ell_e + 4k + 3\ell_r + L(\ell_m + k + \ell_r) + (\#D + \#F)(3\ell_n + k + 3\ell_r)$ bits. An audited token takes $2\ell_n + \ell_m + \ell_e + 4k + 3\ell_r + L(\ell_m + k + \ell_r) + (\#D + \#F)(2\ell_n + k + 2\ell_r) + \#T(\ell_n + \ell_r)$ bits plus the length of a standard signature.

7 Conclusion

Data minimization is a basic privacy principle in authentication mechanisms. In this paper, we show that data minimization does not need to stop at the verifier: using our auditable PABC scheme, the information revealed in a presentation token can be further reduced when forwarding it to an auditor, all while preserving the verifiability of the audit token. In our construction, presentations and audit tokens are anonymous in the sense that neither of them can be linked to the user or credential from which they originated. Audited tokens can be linked to the presentation from which they were derived. This can be used as a feature when the verifier must be unable to inflate the number of presentations that it performed, but it may also be a privacy drawback. We leave open the construction of a scheme satisfying a stronger privacy notion with fully unlinkable audit tokens.

Acknowledgement. This work was supported by the European Community through the Seventh Framework Programme (FP7), under grant agreements n°257782 for the project ABC4Trust and n°318424 for the project FutureID.

References

1. Camenisch, J., Krontiris, I., Lehmann, A., Neven, G., Paquin, C., Rannenberg, K., Zwingelberg, H.: H2.1 – abc4trust architecture for developers. ABC4Trust Heartbeat H2.1 (2011), `https://abc4trust.eu`
2. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM 28(10), 1030–1044 (1985)
3. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
4. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge (2000)
5. Adams, C., Farrell, S.: Rfc 2510, x. 509 internet public key infrastructure certificate management protocols. Internet Engineering Task Force (1999)
6. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM 38(3), 691–729 (1991)
7. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Heidelberg (2006)
8. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (October 2001)
9. Canetti, R.: Universally composable signature, certification, and authentication. In: IEEE CSFW-17. IEEE Computer Society (2004)
10. Backes, M., Hofheinz, D.: How to break and repair a universally composable signature functionality. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 61–72. Springer, Heidelberg (2004)
11. Canetti, R.: Universally composable signatures, certification and authentication. IACR Cryptology ePrint Archive, Report 2003/239 (2003)
12. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
13. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: de Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005)
14. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 444–461. Springer, Heidelberg (2010)
15. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009)
16. Brzuska, C., Busch, H., Dagdelen, O., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., Schröder, D.: Redactable signatures for tree-structured data: Definitions and constructions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 87–104. Springer, Heidelberg (2010)

17. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)
18. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 1–20. Springer, Heidelberg (2012)
19. Bellare, M., Neven, G.: Transitive signatures based on factoring and RSA. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 397–414. Springer, Heidelberg (2002)
20. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 108–125. Springer, Heidelberg (2009)
21. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010)
22. Fuchsbauer, G.: Commuting signatures and verifiable encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 224–245. Springer, Heidelberg (2011)
23. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145. IEEE Computer Society (2001)
24. Backes, M., Goldberg, I., Kate, A., Mohammadi, E.: Provably secure and practical onion routing. In: IEEE CSF-25, pp. 369–385. IEEE Press
25. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 169–187. Springer, Heidelberg (2005)
26. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
27. Fischlin, M.: Trapdoor Commitment Schemes and Their Applications. PhD thesis, Goethe Universität Frankfurt (2001)
28. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
29. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
30. Camenisch, J., Lehmann, A., Neven, G., Rial, A.: Privacy-preserving auditing for attribute-based credentials. IACR Cryptology ePrint Archive, Report 2014/468
31. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
32. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
33. Schnorr, C.-P.: Efficient identification and signatures for smart cards (abstract) (rump session). In: Quisquater, J.-J., Vandewalle, J. (eds.) Advances in Cryptology - EUROCRYPT 1989. LNCS, vol. 434, pp. 688–689. Springer, Heidelberg (1990)
34. Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (2009)
35. Damgård, I.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
36. Benhamouda, F., Camenisch, J., Krenn, S., Lyubashevsky, V., Neven, G.: Improved Zero-Knowledge Proofs for Lattice Encryption Schemes, Linking Classical and Lattice Primitives, and Applications (2014) (unpublished manuscript)