# Privacy-Preserving Auditing for Attribute-Based Credentials

Jan Camenisch, Anja Lehmann, Gregory Neven, and Alfredo Rial
IBM Research – Zurich, 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 audited 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 efficient instantiations based on the strong RSA and Decision Composite Residuosity (DCR) assumptions in the random-oracle model.

## 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 can't tell whether they were performed by the same or by different users. PABCs offer important privacy advantages over other attribute certification schemes, that 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 further than 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 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 audited token verifiable under the issuer's public key?

**Trivial constructions.** Current PABC schemes don't 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 doesn't 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 as well as audited 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 signature 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 signature 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, doesn't work.

Another related line of work is that 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, $\mathcal{U}$ 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 $\varphi$ is analyzed by comparing the view of an environment $\mathcal{Z}$ in a real execution of $\varphi$ 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. We assume static corruptions, meaning that the adversary chooses which parties to corrupt at the beginning of the game. 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 protocol $\varphi$ 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 $\varphi$ or with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}_\varphi$.

More formally, let $k \in \mathbb{N}$ denote the security parameter and $a \in \{0,1\}^*$ denote an input. Two binary distribution ensembles $X = \{X(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ are indistinguishable ($X \approx Y$) if for any $c, d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all $a \in \cup_{\kappa\leq k^d}\{0,1\}^\kappa$, $|\Pr[X(k,a) = 1] - \Pr[Y(k,a) = 1]| < k^{-c}$. Let $\mathrm{REAL}_{\varphi,\mathcal{A},\mathcal{Z}}(k,a)$ denote the random variable given by the output of $\mathcal{Z}$ when executed on input $a$ with $\mathcal{A}$ and parties running $\varphi$, and let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,a)$ denote the output distribution of $\mathcal{Z}$ when executed on $a$ with $\mathcal{S}$ and dummy parties relaying to $\mathcal{F}$. We say that protocol $\varphi$ securely realizes $\mathcal{F}$ if, for all polynomial-time $\mathcal{A}$, there exists a polynomial-time $\mathcal{S}$ such that, for all polynomial-time $\mathcal{Z}$, $\mathrm{REAL}_{\varphi,\mathcal{A},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

When describing ideal functionalities, we use the following conventions:

**Network vs. local communication.** The identity of an ITM instance (ITI) consists of a party identifier *pid* and a session identifier *sid*. A set of parties in an execution of a system of ITMs are a protocol instance if they have the same session identifier *sid*. ITIs can pass direct inputs to and outputs from "local" ITIs that have the same *pid*. An ideal functionality $\mathcal{F}$ has $pid = \perp$ and is considered local to all parties. 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 received from party $\mathcal{P}$ only. Communication between ITIs with different party identifiers must take place over the network. The network is controlled by the adversary, meaning that he can arbitrarily delay, modify, drop, or insert messages.

**Waiting for the simulator.** 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 $m'$.

A protocol $\varphi^{\mathcal{G}}$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model when $\varphi$ is allowed to invoke the ideal functionality $\mathcal{G}$. Therefore, for any protocol $\psi$ that securely realizes functionality $\mathcal{G}$, the composed protocol $\varphi^\psi$, which is obtained by replacing each invocation of an instance of $\mathcal{G}$ with an invocation of an instance of $\psi$, securely realizes $\mathcal{F}$.

Our protocol makes use of the standard functionalities $\mathcal{F}_{\mathrm{REG}}$ [23] for key registration, $\mathcal{F}_{\mathrm{SMT}}$ for secure message transmission, and $\mathcal{F}^D_{\mathrm{CRS}}$ [23] for common

reference strings with distribution $D$. 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]. These functionalities are quite complex, as they model the various imperfection of the protocols, in particular, what routing information an adversary learns. These information depend heavily on how messages are routed, how many other users currently use the chanel, how many nodes are controlled by the adversary, etc. Indeed, the modelling and realizations of anonymous communcation is an active field of research. Now, if we used one of these functionalities for our protocols, we would have had to model all these imperfactions 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 not not learn the identities of the sender and the receiver.

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

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

1. On input $(\mathsf{send}, sid, m)$ 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 $(\mathsf{send}, sid')$ from $\mathcal{S}$.
   - Send $(\mathsf{sent}, sid, m)$ to $\mathcal{R}$.

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

### 3.2 Ideal Functionality of Privacy-Preserving Audits

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

**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 $(\mathsf{issue}, sid)$ from $\mathcal{S}$.
   - If $\mathcal{U}_n$ is honest then store $[\mathcal{U}_n, \langle a_l \rangle_{l=1}^{L}]$ in $\mathsf{Tbl}_1$, 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. $a'_l$

To save on notation we further introduce the following conventions, which basically enforce that the functionality only proceeds if the incoming messages are well-formed. That is, for the presentation, audit and verify messages, $\mathcal{F}_{\text{AUD}}$ 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, $\mathcal{F}_{\text{AUD}}$ further checks that $T \subseteq D$.

The functionality assumes certified identities of the users, verifiers, and the signer. See the discussion on public keys below.

The issue interface is called by the issuer with the user identity and the attributes in the 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, then 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 presentation phase in Figure 2 lets a user present a subset of attributes to a verifier $\mathcal{V}_j$. Honest users can only show combinations of attributes that appear in 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 (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.

The audit phase depicted in Figure 2 has two interfaces: the auditgen interface to create audited tokens, and the auditvf to verify audited tokens. Honest verifiers can only create audited 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 audited tokens that he wants. Unlike credentials and presentations, audited 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 audited token. Note that the simulator is only given the values of the transferred attributes $T$, which guarantees that audited 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 audited token through the auditvf interface. The functionality enforces consistency through Table 3, guaranteeing that verification of the same audited 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 audited 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 audited 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, audited 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 audited 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 audited 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 Technical Preliminaries

### 4.1 Trapdoor Commitment Schemes

A non-interactive commitment scheme consists of algorithms ComSetup, Commit and VfCom. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme $par_c$. $\mathsf{Commit}(par_c, x)$ 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 has a hiding property and a binding property. Informally speaking, 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 $x'$.

A trapdoor commitment scheme [26, 27] is a commitment scheme where there exists trapdoor information that allows to open commitments to any value. More formally, there exist polynomial-time algorithms ComSimSetup and ComOpen, where ComSimSetup on input $1^k$ outputs parameters $par_c$ with trapdoor $td_c$ such that $par_c$ are indistinguishable from those produced by ComSetup. Given a commitment $com$ for message $x_1$ and opening information $open_1$, a second message $x_2$ and trapdoor $td_c$, ComOpen produces the opening information $open_2$ such that $\mathsf{VfCom}(par_c, com, x_2, open_2) = 1$.

### 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 $\mathcal{L}$ 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 $\sigma$ on the message $m$ with respect to statement $x$, else output $\perp$.

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

A signature of knowledge scheme needs to fulfil the following three properties.

**Definition 1 (Correctness).** *Correctness ensures that the algorithm* $\mathsf{SKVerify}$ *accepts the signatures of knowledge that are output by the algorithm* $\mathsf{SKSign}$*. More formally, for any* $(x, w) \in R$ *and any* $m \in \mathcal{M}$*, we require*

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

**Definition 2 (Simulatability).** *Simulatability requires the existence of a simulator defined by algorithms* $(\mathsf{SKSimSetup}, \mathsf{SKSimSign})$ *that can compute signatures without having a witness for the statement. Let* $\mathsf{SKSim}$ *be an algorithm that, on input* $(par_s, R, x, w, m)$*, if* $(x, w) \in R$*, outputs* $\mathsf{SKSimSign}(par_s, td_s, R, x, m)$*, else outputs* $\perp$*. More formally, simulatability is defined as follows.*

$$\Big|Pr\left[par_s \leftarrow \mathsf{SKSetup}(1^k);\ b \leftarrow \mathcal{A}(par_s)^{\leftrightarrow \mathsf{SKSign}(par_s, \cdot, \cdot, \cdot, \cdot)} :\ b = 1\right] - Pr\left[(par_s, td_s) \leftarrow \mathsf{SKSimSetup}(1^k); b \leftarrow \mathcal{A}(par_s)^{\leftrightarrow \mathsf{SKSim}(par_s, td_s, \cdot, \cdot, \cdot, \cdot)} :\ b = 1\right]\Big| \leq \epsilon(k)$$

**Definition 3 (Extraction).** *Extraction requires the existence of an algorithm* $\mathsf{SKExt}$ *that, given* $(R, x, \sigma, m)$ *such that* $\mathsf{SKVerify}(par_s, R, x, m, \sigma)$ *outputs* 1*, extracts a witness* $w$ *such that* $(x, w) \in R$*. Let* $S_{sk}$ *contain the tuples* $(R, x, w, m)$ *submitted to* $\mathsf{SKSim}$*. More formally, extraction is defined as follows.*

$$Pr\left[\begin{array}{l}(par_s, td_s) \leftarrow \mathsf{SKSimSetup}(1^k);\ (R, x, m, \sigma) \leftarrow \mathcal{A}(par_s)^{\leftrightarrow \mathsf{SKSim}(par_s, \cdot, \cdot, \cdot, \cdot)}; \\ w \leftarrow \mathsf{SKExt}(par_s, td_s, x, m, \sigma) : \\ 1 \leftarrow \mathsf{SKVerify}(par_s, R, x, m, \sigma)\ \wedge\ (R, x, \cdot, m) \notin S_{sk}\ \wedge\ (x, w) \notin R\end{array}\right] \leq \epsilon(k)$$

## 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 $\sigma$, proving that he has a valid credential for all the committed values. To further ensure that the signature can not be used in a different context, e.g., by a malicious party trying to impersonate an honest user, the proof signs a message which contains the public key of the verifier and a fresh *nonce* chosen by the user. The entire presentation token then consists of the signature of knowledge $\sigma$, 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 the *nonce* contained in those proofs 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 where he wishes to disclose 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 can not 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 $\sigma$, the correct opening of all commitments for the disclosed attributes and the verifiers signature $s$.

### 5.1 Our protocol

Our protocol uses a trapdoor commitment scheme $(\mathsf{ComSetup}, \mathsf{Commit}, \mathsf{VfCom})$, and two signature schemes $(\mathsf{KeyGen}_{\mathcal{I}}, \mathsf{Sign}_{\mathcal{I}}, \mathsf{VfSig}_{\mathcal{I}})$ (for the issuer) and $(\mathsf{KeyGen}_{\mathcal{V}}, \mathsf{Sign}_{\mathcal{V}}, \mathsf{VfSig}_{\mathcal{V}})$ (for the verifier). 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 issuers signature scheme must allow signing blocks of messages, whereas for the verifiers 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 generates its key as $(ipk, isk) \leftarrow \mathsf{KeyGen}_{\mathcal{I}}(1^k)$, stores $(ipk, isk)$ and sends $(\mathsf{register}, \mathcal{I}, ipk)$ to $\mathcal{F}_{\mathrm{REG}}$. Similarly, each verifier $\mathcal{V}_j$ runs $(vpk_j, vsk_j) \leftarrow \mathsf{KeyGen}_{\mathcal{V}}(1^k)$, stores $(vpk_j, vsk_j)$ and sends $(\mathsf{register}, \mathcal{V}_j, vpk_j)$ to $\mathcal{F}_{\mathrm{REG}}$.

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 $par_c \leftarrow \mathsf{ComSetup}(1^k)$ are obtain via $\mathcal{F}_{\mathrm{CRS}}^{D}$, 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. The verifier also maintains a list of of nonces $\mathsf{L}_{nonce}$ 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 $\mathcal{F}_{\text{SMT}}$ and $\mathcal{F}_{\text{ASMT}}$ respectively.

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

**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 randomly chosen $sid''$ and send $(\mathsf{send}, sid_{\text{SMT}}, (sid, \langle a_l \rangle_{l=1}^{L}, cred))$ to $\mathcal{F}_{\text{SMT}}$.

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

a) Upon receiving $(\mathsf{sent}, sid_{\text{SMT}}, (sid, \langle a_l \rangle_{l=1}^{L}, cred))$ 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 Protocol.** 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 S1. 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 exist.
b) Create a signature of knowledge of a valid credential w.r.t. committed attributes and bound to a nonce:
   - Compute $(com_l, open_l) \leftarrow \mathsf{Commit}(par_c, a_l)\ \forall l \in D \cup F$.
   - 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 $R :=$

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

     and set the statement and witness to $x := (ipk, \langle com_l \rangle_{l \in D \cup F}, par_c, D, F)$, $w := (cred, \langle a_l \rangle_{l=1}^{L}, \langle open_l \rangle_{\in D \cup F})$.
   - Generate the signature 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 randomly chosen $sid''$.

- 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 S2. Verifier $\mathcal{V}_j$ verifies the presentation token:**

a) Upon receiving a message $(\mathsf{sent}, sid_{\text{ASMT}}, (\langle a_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma))$ from $\mathcal{F}_{\text{ASMT}}$ check that $nonce \notin \mathsf{L}_{nonce}$ and abort otherwise.

b) Verify signature of knowledge and commitments:
- Set $(R, x, m)$ similarly as in Step S1(b) and verify that $1 \leftarrow \mathsf{SKVerify}(par_s, R, x, m, \sigma)$.
- Verify that $1 \leftarrow \mathsf{VfCom}(par_c, a_l, com_l, open_l)$ for all $l \in D \cup F$. Abort if a verification fails.

c) Store token & 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 exist.

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

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 $1 \leftarrow \mathsf{VfSig}_{\mathcal{V}}(vpk_j, s, (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, T))$.

c) Set $(R, x, m)$ as in Step S1(b) and verify that $1 \leftarrow \mathsf{SKVerify}(par_s, R, x, m, \sigma)$.

d) Verify that $1 \leftarrow \mathsf{VfCom}(par_c, a_l, com_l, open_l)$ for all $l \in T$.

e) If all checks succeeded, output $(\mathsf{audvf}, sid, \texttt{valid})$ and end.

### 5.2 Security

We prove our protocol secure in the UC model based on the security properties of the underlying building blocks. A sketch of the simulator is given in Appendix A.

**Theorem 1.** *The above construction securely implements $\mathcal{F}_{\text{AUD}}$ in the $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{CRS}}^{D}$, 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.*

## 6 Instantiation of Privacy-Preserving Audits

We recall the Damgård-Fujisaki commitment scheme, which under the strong RSA assumption securely instantiates algorithms ($\mathsf{ComSetup}, \mathsf{Commit}, \mathsf{VfCom}, \mathsf{ComSimSetup}, \mathsf{ComOpen}$) described in Section 4.1. 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 $l_r = 80$.

$\mathsf{ComSetup}(1^k)$**.** 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 $par_c = (g, h, \tilde{n})$.

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

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

$\mathsf{ComSimSetup}(1^k)$**.** Same as $\mathsf{ComSetup}$, but outputting $\alpha$ 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 [30] 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 $\ell_m$, $\ell_e$, $\ell_n$, and $\ell_r$ be system parameters determined by a function of $k$, where $\ell_r$ 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 $\ell_m$ plus an additional bit carrying the sign, i.e., $\ell_m + 1$ bits in total.

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

$\mathsf{Sign}(sk, \langle m_1, \ldots, m_L \rangle)$**.** The message space is the set $\{(m_1, \ldots, m_L) : m_i \in \pm\{0,1\}^{\ell_m}\}$. On input $m_0, \ldots, m_L$, 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 $(e, A, v)$.

$\mathsf{VfSig}(pk, s, \langle m_1, \ldots, m_L \rangle)$**.** 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 the CPA secure version of Camenisch-Shoup encryption scheme [31] as well as generalized Schnorr proof protocols [32, 29, 33]. We describe how to instantiate the signature of knowledge scheme for the relation we require in our protocol, i.e., for $R := \{(x := (ipk, \langle com_l \rangle_{l \in D \cup F}, par_c, D, F), w := (cred, \langle a_l \rangle_{l=1}^{L}, \langle open_l \rangle_{\in D \cup F})$: s.t. $1 \leftarrow \mathsf{VfSig}_{\mathcal{I}}(ipk, cred, \langle a_l \rangle_{l=1}^{L}) \ \wedge \ 1 \leftarrow \mathsf{VfCom}(par_c, a_l, com_l, open_l) \ \forall l \in D \cup F\}$. It is a secure

signature of knowledge in the random oracle model under the strong RSA assumption and the DCR assumption (the proof is straightforward and is given in the full version of this paper).

$\mathsf{SKSetup}(1^k)$**.** On input $1^k$, choose an $\ell_n = k$-bit safe RSA modulus $\mathsf{n}$. Choose random $\mathsf{x}_{(1,1)}, \ldots, \mathsf{x}_{(1,2L)}, \in_R [\mathsf{n}^2/4]$, choose a random $\mathsf{g}' \in_R \mathbb{Z}^*_{\mathsf{n}^2}$, and compute $\mathsf{g} \leftarrow (\mathsf{g}')^{2n}$, and $\mathsf{y}_{(1,i)} \leftarrow \mathsf{g}^{\mathsf{x}_{(1,i)}}$ for $i = 1, \ldots, 2L{+}2$. Output $(\mathsf{n}, \mathsf{g}, \{\mathsf{y}_{(1,i)}\})$.

$\mathsf{SKSign}(par_s, R, x, w, m)$**.** Let $\mathsf{h} = (1 + \mathsf{n})$. Compute a randomized credential: choose random $v' \in_R \{0,1\}^{\ell_v}$ and compute $A' \leftarrow AS^{v'}$ and $v^* \leftarrow v - v'e$. Choose random $r \in_R [\mathsf{n}/4]$ and compute $\mathsf{u} \leftarrow \mathsf{g}^r$, $\mathsf{e}_i \leftarrow \mathsf{y}_{(1,i)}^r \mathsf{h}^{a_i}$ for $i = 1, \ldots, L$, $\mathsf{e}_{L+l} \leftarrow \mathsf{y}_{(1,L+l)}^r \mathsf{h}^{open_l}$ for all $l \in D \cup F$, $\mathsf{e}_{2L+1} \leftarrow \mathsf{y}_{(1,i)}^r \mathsf{h}^{e}$, and $\mathsf{e}_{2L+1} \leftarrow \mathsf{y}_{(1,i)}^r \mathsf{h}^{v^*}$. Compute

$$\pi \leftarrow SPK\{(r, A', v^*, \langle a_l \rangle_{l=1}^{L}, \{o_l\}_{l \in D \cup F}) : \mathsf{u} = \mathsf{g}^r \wedge \bigwedge_{\forall l \in D \cup F} \big(com_l = g^{a_l} h^{o_l} \pmod{\tilde{n}} \wedge \mathsf{e}_{L+l} = \mathsf{y}_{(1,L+l)}^r \mathsf{h}^{o_l}\big) \wedge$$
$$\mathsf{e}_1 = \mathsf{y}_{(1,1)}^r \mathsf{h}^{a_1} \wedge \ldots \wedge \mathsf{e}_1 = \mathsf{y}_{(1,L)}^r \mathsf{h}^{a_L} \wedge Z = A^e R_1^{a_1} \ldots R_L^{a_L} S^v \pmod{n}\big) \wedge$$
$$\mathsf{e}_{2L+1} = \mathsf{y}_{(1,i)}^r \mathsf{h}^{e} \wedge \mathsf{e}_{2L+1} = \mathsf{y}_{(1,i)}^r \mathsf{h}^{v^*} \wedge a_i \in \pm\{0,1\}^{\ell_m} \wedge 2^{\ell_e} > e > 2^{\ell_e - 1}\}(m)$$

and output $(\mathsf{u}, \{\mathsf{e}_i\}, A', \pi)$. For the realization of the non-interactive proof of knowledge $\pi$ we refer Camenisch et al. [29, 33].

$\mathsf{SKVerify}(par_s, R, x, m, (\mathsf{u}, \{\mathsf{e}_i\}, A', \pi))$**.** This algorithm will verify whether $\pi$ 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 [34]. The $\mathsf{SKExt}$ works by decryption of $(\mathsf{u}, \{\mathsf{e}_i\})$, providing all attributes, opening information of the commitments, and the credential (CL-signature).

## 7 Conclusion

Data minimization is a basic privacy principle in authentication mechanisms. In this paper, we show that data minimization doesn't 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 audited token.

In our construction, presentations and audited 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 the construction of a scheme satisfying a stronger privacy notion with fully unlinkable audited tokens as an open problem.

## 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) Available from `https://abc4trust.eu`.
2. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM **28**(10) (1985) 1030–1044
3. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In SCN 02. Volume 2576 of LNCS., Springer (September 2002) 268–289
4. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge, MA, USA (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) (1991) 691–729
7. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In Dwork, C., ed.: CRYPTO 2006. Volume 4117 of LNCS., Springer (August 2006) 78–96
8. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, IEEE Computer Society Press (October 2001) 136–145
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 ISC 2004. Volume 3225 of LNCS, Springer, pp. 61–72.
11. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003)
12. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In CT-RSA 2002. Volume 2271 of LNCS, Springer, pp. 244–262.
13. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In ESORICS 2005. Volume 3679 of LNCS., Springer , pp.159–177.
14. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In PKC 2010. Volume 6056 of LNCS, Springer, pp., 444–461.
15. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In PKC 2009. Volume 5443 of LNCS, Springer, pp. 317–336.
16. Brzuska, C., Busch, H., Dagdelen, Ö., 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 ACNS 10.
17. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In Kim, K., ed.: ICISC 01. Volume 2288 of LNCS., Springer (December 2001) 285–304
18. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. In TCC 2012. Vol. 7194 of LNCS.
19. Bellare, M., Neven, G.: Transitive signatures based on factoring and RSA. In ASIACRYPT 2002. Volume 2501 of LNCS, Springer pp. 397–414.
20. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In CRYPTO 2009. Volume 5677 of LNCS. Springer, pp. 108–125.
21. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In CRYPTO 2010. Volume 6223 of LNCS, Springer, pp. 209–236.

22. Fuchsbauer, G.: Commuting signatures and verifiable encryption. In EUROCRYPT 2011. Volume 6632 of LNCS, Springer, pp. 224–245.
23. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, IEEE Computer Society (2001) 136–145
24. Backes, M., Goldberg, I., Kate, A., Mohammadi, E.: Provably secure and practical onion routing. In IEEE CSF-25, IEEE press, pp. 369–385.
25. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: CRYPTO 2005, Springer, pp. 169–187.
26. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. **37**(2) (1988), pp. 156–189.
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) (1988) 281–308
29. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In CRYPTO 97. Vol. 1294 of LNCS, Springer, pp. 410–424.
30. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In SCN 02, Volume 2576 of LNCS, Springer, pp. 268–289.
31. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In CRYPTO 2003. Volume 2729 of LNCS, Springer, pp. 126–144.
32. Schnorr, C.P.: Efficient identification and signatures for smart cards (abstract) (rump session). In EUROCRYPT '89. Vol. 434 of LNCS, Springer, pp. 688–689.
33. Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized schnorr proofs. In EUROCRYPT 2009. Volume 5479 of LNCS, Springer, pp. 425–442.
34. Damgård, I.: Efficient concurrent zero-knowledge in the auxiliary string model. In EUROCRYPT 2000, Volume 1807 of LNCS, Springer, pp. 418–430.

# A Security

To prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{\text{AUD}}$, we have to show that for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ there exist a simulator $\mathcal{S}$, such that $\mathcal{Z}$ can not distinguish whether it's interacting with $\mathcal{A}$ and the protocol in the real world or with $\mathcal{S}$ and $\mathcal{F}_{\text{AUD}}$.

The simulator thereby plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{\text{AUD}}$ for all corrupt parties in the ideal world. We denote by $''\mathcal{P}''$ that the simulator plays the role of the honest party $\mathcal{P}$ in the real world.

The simulation is described along two different main cases, depending on whether the issuer is honest or corrupt. Regarding the other parties, the adversary can corrupt an arbitrary subset of verifiers, users, and auditors. Depending on the different combinations of honest and corrupt parties, the simulation then branches further in the different subprotocols for issuance, presentation, audit token generation and verification. We now provide a sketch of our simulator, the full description and the games appear in the full version of the paper.

## A.1 Sketch of Simulator

For our simulation, we first replace the parameters of the signature of knowledge and of the trapdoor commitment scheme by simulated parameters, where

the trapdoors are known to the simulator. That is, the CRS is now replaced by $(par_s, par_c)$ where $(par_s, td_s) \leftarrow \mathsf{SKSimSetup}(1^k)$ and $(par_c, td_c) \leftarrow \mathsf{ComSimSetup}(1^k)$, and $td_s$ and $td_c$ are known to the simulator. This has no noticeable impact on the view of the environment, due to the simulatability property of the signature of knowledge scheme and the trapdoor commitments.

**Case 1 – Honest Issuer**
When the issuer is honest, the simulated issuer "$\mathcal{I}$" creates his signing key $(ipk, isk) \leftarrow \mathsf{KeyGen}_{\mathcal{I}}(1^k)$ as in the real protocol. Thus, we can create valid credentials in the simulation.

*Issuance:* The simulation is triggered when $\mathcal{S}$ receives $(\mathsf{issue}, sid, \mathcal{U}_n)$ from $\mathcal{F}_{\mathrm{AUD}}$, and then branches depending on whether $\mathcal{U}_n$ is corrupt or honest as follows.

**honest user.** If $\mathcal{U}_n$ is honest, $\mathcal{S}$ starts the simulation of "$\mathcal{U}_n$" and "$\mathcal{I}$" in the real world, merely simulating the traffic between both parties. When "$\mathcal{U}_n$" receives the dummy message that "$\mathcal{I}$" had sent, $\mathcal{S}$ sends $(\mathsf{issue}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$, ensuring that a credential entry in the ideal functionality will be created.

**corrupt user.** If $\mathcal{U}_n$ is corrupt, $\mathcal{S}$ sends $(\mathsf{issue}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$, receiving $(\mathsf{issue}, sid, \langle a_l \rangle_{l=1}^{L})$ from $\mathcal{F}_{\mathrm{AUD}}$. With the knowledge of all attributes, "$\mathcal{I}$" then runs the normal issuance protocol with $\mathcal{U}_n$ in the real world and stores $[\mathcal{S}, \langle a_l \rangle_{l=1}^{L}]$ in a locally maintained table $\mathsf{Tbl}_1$.

*Presentation:* The simulation for the presentation protocol again depends on the different combinations of user and verifier. Note that the case of a corrupt user and corrupt verifier requires no simulation, as all communication is internal to the adversary.

**honest user & honest verifier.** The simulation is triggered when $\mathcal{S}$ receives $(\mathsf{present}, sid, \mathcal{V}_j)$ from $\mathcal{F}_{\mathrm{AUD}}$. Let "$\mathcal{U}_n$" send a dummy message of the correct length to "$\mathcal{V}_j$". When "$\mathcal{V}_j$" receives the message, $\mathcal{S}$ sends $(\mathsf{present}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$, increases the token identifier $tid$ for "$\mathcal{V}_j$" and stores an empty token presentation token $\perp$ and dummy attributes $\langle a_l \rangle_{l=1}^{L}$ together with $tid$ in $\mathsf{Tbl}_2$.

**honest user & corrupt verifier.** Again, the simulation starts when $\mathcal{S}$ receives $(\mathsf{present}, sid, \mathcal{V}_j)$ from $\mathcal{F}_{\mathrm{AUD}}$. The simulator then sends $(\mathsf{present}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$ and receives $(\mathsf{tokrec}, sid, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid(\mathcal{V}_j))$ in return. Knowing all attributes that will be shown to the verifier, $\mathcal{S}$ now creates real commitments and openings for all attributes in $D \cup F$ and produces a simulated $\sigma$ using $\mathsf{SKSimSign}$ and the trapdoor $td_s$. The honest user "$\mathcal{U}_n$" then sends the composed presentation token to $\mathcal{V}_j$.

**corrupt user & honest verifier** Here the simulation starts when an honest verifier "$\mathcal{V}_j$" in the real world receives a presentation token $(\mathsf{sent}, sid_{\mathrm{ASMT}}, (\langle a_l \rangle_{l \in D \cup F}, D, F, msg, nonce, \langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in D \cup F}, \sigma))$ from a dishonest user. If the token is valid, but there is *no* matching credential entry in $\mathsf{Tbl}_1$ maintained by $\mathcal{S}$, we can use the signature of knowledge extractor $\mathsf{SKExt}$

to obtain the underlying credential (and all attributes) from $\sigma$ and derive a forgery against the issuers signature scheme.
If the token is valid and there is a matching entry in $\mathsf{Tbl}_1$, $\mathcal{S}$ sends ($\mathsf{present}$, $sid$, $\mathcal{V}_j, D, F, \langle a_l \rangle_{l \in D \cup F}, msg$) to $\mathcal{F}_{\mathrm{AUD}}$ and subsequently triggers with ($\mathsf{present}$, $sid$) the delivery to the honest verifier in the ideal world. The simulator increases $tid$ for $\mathcal{V}_j$ and stores the received presentation token ($nonce, \langle com_l \rangle_{l \in D \cup F}$, $\langle open_l \rangle_{l \in D \cup F}, \sigma$) together with all disclosed attributes $\langle a_l \rangle_{l \in D \cup F}$and $tid$ in $\mathsf{Tbl}_2$.

*Audit Token Generation:* Here we only have to consider the case where the verifier is honest. The simulation is triggered when $\mathcal{S}$ receives ($\mathsf{auditgen}$, $sid$, $\langle a_l \rangle_{l \in T}$, $D, F, T, tid$) from $\mathcal{F}_{\mathrm{AUD}}$. $\mathcal{S}$ then retrieves the presentation token stored with $tid$ in $\mathsf{Tbl}_2$, which can come in three different flavours to which we have adapt our simulation accordingly. Those are: the simulator retrieves (i) an empty presentation token for $tid$, (ii) an incomplete (simulated) presentation token with dummy openings for attributes he has to reveal now, or (iii) a full presentation token.

**honest verifier & case (i).** In this case the retrieved token equals $\perp$, which occurs when the presentation happened between an honest user and honest verifier. $\mathcal{S}$ then computes a (simulated) audit token from scratch. That is, the simulator first computes real commitments and openings for all attributes in $T$, whereas for all attributes in $D \setminus T$ and $F$ he commits to zeroes obtaining a "dummy" commitment and opening. He then chooses a fresh $nonce$ and simulates $\sigma$ using $\mathsf{SKSimSign}$ and the trapdoor $td_s$. The signature $s$ is computed normally and the audit token is set to $audtok := (\langle com_l \rangle_{l \in D \cup F}, \langle open_l \rangle_{l \in T}, \sigma, nonce, s)$ which is input to $\mathcal{F}_{\mathrm{AUD}}$ as ($\mathsf{auditgen}$, $sid$, $audtok$).
Internally, $\mathcal{S}$ now stores the simulated presentation token, together with $\langle a'_l \rangle_{l=1}^{L}$ and $tid$ in $\mathsf{Tbl}_2$, where $a'_l := a_l$ for all $l \in T$ and $a'_l := \perp$ otherwise. Note that the audit token already reveals commitments for all attributes in $D \setminus T$ and $F$, for which the simulator does not know the attribute value or real opening yet. However, we can open them in a correct way whenever $\mathcal{S}$ learns the real attributes in a subsequent audit token request. This is handled in the next case.

**honest verifier & case (ii).** Here, a simulated and incomplete presentation token retrieved from $\mathsf{Tbl}_2$ and $\mathcal{S}$ received an audit request that should reveal attributes $a_l \in T$ where the stored presentation token contains (for some of them) $a'_l := \perp$. That is, the simulator has to open some of the dummy commitments in the new audit token. However, as $\mathcal{S}$ now learned the corresponding attributes from $\mathcal{F}_{\mathrm{AUD}}$, he uses $\mathsf{ComOpen}$ on input the trapdoor $td_c$, the dummy commitment and the dummy opening (stored in the presentation token) to produces openings that will open the dummy commitment to the correct attribute. The presentation token in $\mathsf{Tbl}_2$ is then updated accordingly, replacing the dummy openings and dummy attributes with the correct ones. The rest of the audit token is computed according to the real protocol,

and $\mathcal{S}$ finally sends the produced audit token as $(\mathsf{auditgen}, sid, audtok)$ to $\mathcal{F}_{\mathrm{AUD}}$.

**honest verifier & case (iii).** In this case a full fledged presentation token is stored (received from a corrupt user, or generated by the simulator), and thus "$\mathcal{V}_j$" simply derives an audit token *audtok* according to the real protocol and sends it to $\mathcal{F}_{\mathrm{AUD}}$.

*Audit Token Verification:* Whenever $\mathcal{S}$ receives $(\mathsf{auditvf}, sid, audtok, \mathcal{V}_j, D, F, T, \langle a_l \rangle_{l \in T}, msg)$ from $\mathcal{F}_{\mathrm{AUD}}$, he verifies *audtok* according to our protocol and sends the result to $\mathcal{F}_{\mathrm{AUD}}$.

When an auditor receives a valid audit token in the *real* world which would not be valid according to the functionality, we can either break the unforgeability of the signature scheme of the verifier or of the issuer. The information for the latter case will be extracted from the received audit token using $\mathsf{SKExt}$. The reduction is given in the full version of the proof.

**Case 2 – Corrupt Issuer**

When the issuer is corrupt, the simulator no longer controls the set of "valid" credentials, which is reflected in the proof. The main changes occur in the simulation of the issuance with an honest user and the presentation between an honest user and corrupt verifier. Thus, we omit the other cases here.

*Issuance:* Here we only have to consider issuance with an honest user, as with a corrupt user the communication is internal to the adversary.

**with honest user.** If an honest user "$\mathcal{U}_n$" receives a message $(\mathsf{sent}, sid_{\mathrm{SMT}}, (sid, \langle a_l \rangle_{l=1}^{L}, cred))$ where *cred* is a valid credential from a dishonest issuer $\mathcal{I}$ (specified in *sid*), the simulator $\mathcal{S}$ sends $(\mathsf{issue}, sid, \mathcal{U}_n, \langle a_l \rangle_{l=1}^{L})$ to $\mathcal{F}_{\mathrm{AUD}}$, and triggers the output to the honest user by sending $(\mathsf{issue}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$. Thus, a credential entry will be registered for the same attributes in the ideal functionality as well. The simulator also stores the credential and attributes in $\mathsf{Tbl}_1$.

*Presentation:* The simulation of the presentation protocol between an honest user & honest verifier is exactly the same as in Case 1. For the setting of corrupt user & honest verifier, the only difference to Case 1 is that we no longer control the issued credentials and thus do not reduce a forged presentation token to a forged signature.

**honest user & corrupt verifier.** The simulation starts when $\mathcal{S}$ receives $(\mathsf{present}, sid, \mathcal{V}_j)$ from $\mathcal{F}_{\mathrm{AUD}}$. The simulator then sends $(\mathsf{present}, sid)$ to $\mathcal{F}_{\mathrm{AUD}}$ and receives $(\mathsf{tokrec}, sid, D, F, \langle a_l \rangle_{l \in D \cup F}, msg, tid(\mathcal{V}_j))$ in return. Having learned the verified attributes $\langle a_l \rangle_{l \in D \cup F}$, $\mathcal{S}$ then retrieves a matching credential (and additional unrevealed attributes) from $\mathsf{Tbl}_1$ and derives the presentation token according to the protocol. The simulator stores the derived token together with the attributes and $tid(\mathcal{V}_j)$ in $\mathsf{Tbl}_2$ and sends the presentation token to $\mathcal{V}_j$.