Recovering Lost Device-Bound Credentials

Foteini Baldimtsi[1], Jan Camenisch[2], Lucjan Hanzlik[3](✉), Stephan Krenn[4], Anja Lehmann[2], and Gregory Neven[2]

[1] Boston University, Boston, USA
foteini@bu.edu
[2] IBM Research – Zurich, Rüschlikon, Switzerland
{jca,anj,nev}@zurich.ibm.com
[3] Wrocław University of Technology, Wrocław, Poland
lucjan.hanzlik@pwr.wroc.pl
[4] AIT Austrian Institute of Technology GmbH, Vienna, Austria
stephan.krenn@ait.ac.at

Abstract. Anonymous credential systems allow users to authenticate in a secure and private fashion. To protect credentials from theft as well as from being shared among multiple users, credentials can be bound to physical devices such as smart cards or tablets. However, device-bound credentials cannot be exported and backed up for the case that the device breaks down or is stolen. Restoring the credentials one by one and re-enabling the legitimate owner to use them may require significant efforts from the user. We present a mechanism that allows users to store some partial backup information of their credentials that will allow them to restore them through a single interaction with a device registration authority, while security and privacy are maintained. We therefore define anonymous credentials *with backup* and provide a generic construction that can be built on top of many existing credential systems.

Keywords: Anonymous credentials · Backup · Restore credentials

1 Introduction

Digital credentials are used to certify a set of attributes for a user (i.e., birth date, sex, clearances, access rights, or qualifications), similar to traditional paper-based credentials such as identity cards or driving licenses. However, their electronic nature makes them easy to duplicate and share. This is particularly problematic when users have an incentive to share their credentials, e.g., when they give access to payed subscription services such as music or video streaming. The problem becomes even worse when *anonymous* credentials are used, since a service provider cannot determine whether two *presentations* (i.e., authentications) were performed using the same or different credentials.

This work was supported by the European Community through the Seventh Framework Programme (FP7) under grant agreement number 318424 (*FutureID*) and by project S40012/K1102 at Wrocław University of Technology.

Service providers therefore often protect credentials by "binding" them to an uncloneable hardware device that can perform authentications, but from which the credentials are not easily extracted. The main idea is that physical access to the device is required to be able to present the credential. A high-security way of doing so is by embedding the credentials on tamperproof smart cards or secure elements; while for lower-security use cases, storing the credentials in obfuscated form on the user's phone or tablet PC may suffice.

Unfortunately, both those techniques do not allow users to make backups of their credentials in order to recover if the device breaks down, or is lost or stolen. Instead, they will have to re-issue all the credentials and possibly also revoke the old ones. However, a single device may store many such credentials and replacing all of them is often costly and impracticable since it might require off-line authentication steps such as appearing in person at an office, receiving a letter by paper mail, or answering secondary security questions. Although efficient backup mechanisms for credentials—and in particular, anonymous credentials—seem essential, no such construction has been proposed so far in the literature.

Our Contributions. In this paper, we describe a scheme for efficient backup and restoration of device-bound credentials. Rather than binding the credentials to the device directly, we propose binding credentials to the user, while devices are registered to users as well. To perform a correct authentication, the user must prove that the credential is bound to the same user to which the device is registered. Credentials can then be exported and backed up in the traditional way, while a device registration authority prevents credential sharing and theft by ensuring that users can only register a limited number of devices and cannot register devices in other users' names.

We consider very strong security features for users as well as service providers. We assume that users store their backups on untrusted media that could fall into the wrong hands of malicious users, or even of malicious device manufacturers. In spite of having access to the backup and being able to register new devices to any user, the attacker should not be able to impersonate the user. We do so by requiring the user to keep a strong secret in an offline vault, e.g., on a piece of paper stored in a safe place. To maintain an acceptable level of usability, however, the vault secret is solely needed for device registration but not for everyday use.

We first give a high level description of an anonymous credential system with backup (BPABC) in Sect. 2, where we also define the syntax of BPABC and give an overview of the related security requirements. Besides the basic functionalities and backup, our framework covers advanced issuance (i.e., attributes can be carried over into new credentials without revealing them to the issuer), scope-exclusive pseudonyms (i.e., pseudonyms that are linkable for a fixed scope string, but unlinkable across scopes), revocation, and equality predicates (i.e., users can prove equality of hidden attributes, potentially across multiple credentials). In Sect. 3 we give a high-level description of the generic construction of a BPABC scheme together with a sketch of its security proof.

Related Work. Anonymous credentials were originally envisioned by Chaum [Cha81,Cha85], and subsequently a large number of schemes have been proposed, e.g., [BL13,BCC+09,CH02,CL01,CL02,CL04,Bra99,PZ13,CMZ14, GGM14]. Various formalizations of basic credential schemes have been proposed in the literature, typically only considering a limited set of features, e.g., Camenisch and Lysyanskaya [CL01] or Garman et al. [GGM14]. Recently, Camenisch et al. [CKL+14] presented the so far most holistic definitional framework for attribute-based credential systems, covering the same features as our framework, except that theirs was limited to software credentials only and thus there was no need for backup. Following their approach of a unified definitional framework, we extend their syntax, definitions, and generic construction to additionally support device-bound credentials.

2 Device-Bound Credentials with Backup

A privacy-enhancing attribute-based credential system (PABC) consists of *users* $\mathcal{U}$ that can request credentials on their attributes from *issuers*, $\mathcal{I}$, and *verifiers*, $\mathcal{V}$, to whom users can present (i.e., prove possession of) an arbitrary set of their credentials. Additionally, in a PABC system *with backup* (BPABC), *device manufacturers* $\mathcal{DM}$ generate hardware tokens, and *device credential issuers* $\mathcal{DI}$ can issue device credentials that can only be used if the device is physically present. The idea is that software credentials certify the users' attributes, whereas device credentials only guarantee that a user has physical access to a valid device. Then, upon presentation, the user shows that he possesses a valid device, and an arbitrary set of software credentials that also belong to the same user.

When joining the system, every user computes a user secret key *usk*, which is used to bind credentials to the user and allows him to derive unique *pseudonyms* for different *scopes*, where a scope may be an arbitrary bit string. Pseudonyms are linkable if computed twice for the same scope, but are completely unlinkable across scopes. Furthermore, a user computes a *vault user secret/public key* pair (*vusk*, *vupk*). Upon presentation, the user needs to know *vupk*, whereas the vault user secret key *vusk* can be stored in a secure vault (e.g., it could be written on paper and stored in a safe), and is only needed to "authenticate" the user every time he obtains a new device or in order to restore a lost device.

Note here that any method to legitimately re-obtaining credentials (anonymous or not) requires some secret data to be stored outside the used hardware tokens, e.g., on paper or an offline data stick. This is, because otherwise the honest owner of a credential could not prove legitimate ownership of a credential once the adversary got access to his device, as the adversary would know exactly the same information and could thus perfectly imitate the user. In traditional settings, this data may be the correct response to a security question for a given service. For a backup mechanism to be practical, it is important that the amount of secret data, as well as the number of modifications and look-ups of this data are kept small. In our construction, the secret data consists of only a single long-term signing key, independent of the number of credentials, and needs to only be accessed when setting up a new device.

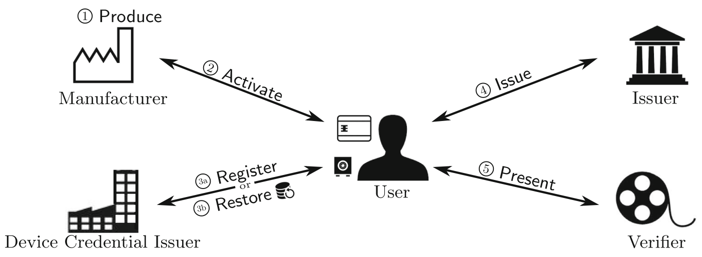


Fig. 1. All steps of a BPABC system are user-centric, and no two actions taken by the user can be linked unintentionally.

In Fig. 1 we describe the main steps of BPABC. Every user may possess several devices, and store an arbitrary set of his software credentials on any of these devices. Users can obtain software credentials from issuers as in a traditional credential system using ④ Issue, i.e., no device is required to obtain a credential. However, we assume that presenting credentials to verifiers using ⑤ Present always requires possession of a device.[1] Furthermore, certain credentials may be bound to specific devices, i.e., they only can be used with this specific device, by binding it to a device binding identifier *dbid* that is unique for every device.

The device manufacturer first generates a device containing a certificate of genuineness using the algorithm ① Produce. When buying a device, the user first has to activate it in interaction with the manufacturer by running the protocol ② Activate, at the end of which the device contains an initial device credential. Now, the user has two possibilities to personalize his device. If the user wants to register a new device, he runs ⓐ Register in interaction with the device credential issuer. If, on the other hand, the user wants to restore a lost device, he runs ⓑ Restore with $\mathcal{DI}$. In both cases, the user uses his vault secret key *vusk* for this personalization; in the latter case, he further uses some backup token that was computed prior to losing the device. Restoring a device can be seen as a special way of registering a device: while for a plain registration, the device receives a fresh *dbid*, restoring allows the user to register the new device with the *dbid of the lost device*. Doing so allows the user to use all his software credentials (including those that were bound to the lost device) with the new device. However, the security requirements of the system guarantee that no user can abuse this restoring procedure to clone or duplicate devices, i.e., it is ensured that at any point in time at most one device with a certain *dbid* is valid in the system.

We chose a multi-step approach for personalizing the device to ensure maximum security to all involved parties, and to model reality more accurately. For instance, requiring to first activate the device with the manufacturer gives him the chance to deny this activation, e.g., if the vendor reported the given device to be stolen. However, as no personal information is involved in the Activate

[1] Note that this is without loss of generality, as the system parameters could simply contain a dummy issuer key for which a user can compute a dummy device credential whenever the verifier's policy does not require possession of a physical device.

protocol on the user side, the manufacturer does not learn any information about the user, but only that a given device is now activated. Splitting the activation and personalization steps allows us to distinguish the device credential issuer (e.g., a public authority) and the manufacturer (e.g., a smart card producer).

2.1 Syntax of Anonymous Credentials with Backup

In the following we formally specify the syntax and interfaces of an anonymous credential system with backup (BPABC). We kept the syntax as close as possible to that of PABC schemes without backup [CKL+14].

We denote algorithms by sans serif font, e.g., A, B. Drawing s uniformly at random from a set $\mathcal{S}$ is denoted by $s \overset{\$}{\leftarrow} \mathcal{S}$. Similarly, $a \overset{\$}{\leftarrow} \mathsf{A}$ denotes that a is the output of a randomized algorithm A. For a two party protocol (A, B), we write $(out_{\mathsf{A}}; out_{\mathsf{B}}) \overset{\$}{\leftarrow} \langle (\mathsf{A}(in_{\mathsf{A}}); \mathsf{B}(in_{\mathsf{B}}) \rangle$ to denote that A obtained output out_{A} on input in_{A} (accordingly for B). For sets $\mathcal{P} \subseteq \mathcal{S}$, we write $\mathcal{P}^c$ for the complement of $\mathcal{P}$, i.e., $\mathcal{P}^c = \mathcal{S} \setminus \mathcal{P}$. We write $(x_i)_{i=1}^{n}$ to denote the vector $(x_1, \ldots, x_n)$. Finally, for $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$.

A BPABC scheme consists of a specification of an attribute space $\mathcal{AS} \subseteq \pm\{0,1\}^{\ell}$, algorithms SPGen, UKGen, VKGen, IKGen, DMKGen, $\mathsf{Produce}$, ITGen, ITVf, $\mathsf{Present}$, Verify, Revoke, BTGen, and protocols $\langle \mathcal{U}.\mathsf{Issue}, \mathcal{I}.\mathsf{Issue} \rangle$, $\langle \mathcal{U}.\mathsf{Register}, \mathcal{DI}.\mathsf{Register} \rangle$, $\langle \mathcal{U}.\mathsf{Restore}, \mathcal{DI}.\mathsf{Restore} \rangle$, $\langle \mathcal{U}.\mathsf{Activate}, \mathcal{DM}.\mathsf{Activate} \rangle$ defined as:

$\mathsf{SPGen} \xrightarrow{\$} \boldsymbol{spar}$**.** On input 1^{κ}, this *system parameter generation* algorithm generates system parameters *spar*.

$\mathsf{UKGen} \xrightarrow{\$} \boldsymbol{usk}$**.** On input system parameters *spar*, the *user key generation* algorithm outputs a user secret key *usk*.

$\mathsf{VKGen} \xrightarrow{\$} (\boldsymbol{vusk}, \boldsymbol{vupk})$**.** On input system parameters *spar*, the *vault user key generation* algorithm outputs a vault user secret/public key pair.

$\mathsf{IKGen} \xrightarrow{\$} (\boldsymbol{isk}, \boldsymbol{ipk}, \boldsymbol{RI})$**.** On input *spar*, the *(device) issuer key generation* algorithm outputs a public/private issuer key pair and some initial public revocation information, *RI*. Formally, our construction does not require to distinguish software and device credential issuers. However, to ease presentation, we will write $(disk, dipk, RI_{\mathcal{DI}})$ whenever an issuer is currently in the role of a device credential issuer.

$\mathsf{DMKGen} \xrightarrow{\$} (\boldsymbol{dmsk}, \boldsymbol{dmpk})$**.** On input *spar*, the *device manufacturer key generation* algorithm outputs a public/secret manufacturer key pair.

$\mathsf{Produce} \xrightarrow{\$} \mathtt{cert}$**.** On input a secret manufacturer key *dmsk*, the *device production* algorithm outputs a genuineness certificate.

$\langle \mathcal{U}.\mathsf{Activate}; \mathcal{DM}.\mathsf{Activate} \rangle \xrightarrow{\$} ((\boldsymbol{dsk}, \boldsymbol{dbid}, \boldsymbol{dcred}_{init}), \varepsilon)$**.** In this interactive *device activation* protocol, the user takes inputs $(dmpk, \mathtt{cert})$, whereas the device manufacturer takes input *dmsk*. At the end of the protocol, the user obtains a device secret key *dsk*, a device binding identifier *dbid* and the initial device credential $dcred_{init}$.

$\langle \mathcal{U}.\mathsf{Register}; \mathcal{DI}.\mathsf{Register} \rangle \xrightarrow{\$} (\boldsymbol{dcred}, \boldsymbol{RI'}_{\mathcal{DI}})$**.** In this *device registration* protocol, the user takes inputs $(dipk, dmpk, vusk, vupk, dsk, dbid, dcred_{init}, drh)$, while the device credential issuer takes inputs $(\mathit{disk}, \mathit{dmpk}, RI_{\mathcal{DI}}, \mathit{drh})$, where the inputs and outputs are defined as before. Moreover, drh is the revocation handle for the new device. As the result of this protocol, the user obtains the device credential dcred, while the issuer receives an updated revocation information $RI'_{\mathcal{DI}}$.

$\mathsf{ITGen} \xrightarrow{\$} (\boldsymbol{nym}, \boldsymbol{pit}, \boldsymbol{sit})$**.** To issue a software credential, a user needs to generate an *issuance token* that defines the attributes of the credentials to be issued, where (some of) the attributes and the secret key can be hidden from the issuer and can be blindly "carried over" from credentials that the user already possesses (so that the issuer is guaranteed that hidden attributes were vouched for by another issuer).

Taking inputs $\big(\mathit{usk}, \mathit{scope}_{\mathcal{U}}, \mathit{rh}, \big(\mathit{ipk}_i, RI_i, \mathit{cred}_i, (a_{i,j})_{j=1}^{n_i}, R_i\big)_{i=1}^{k+1}, E, M, \mathit{vupk}, \mathit{dbid}\big)$, the *issuance token generation* algorithm outputs a user pseudonym nym and a public/secret issuance token $(\mathit{pit}, \mathit{sit})$. The inputs are defined as follows:

- usk is the user's secret key;
- $\mathit{scope}_{\mathcal{U}}$ is the scope of the generated user pseudonym nym, where $\mathit{scope} = \varepsilon$ if no pseudonym is to be generated (in which case $\mathit{nym} = \varepsilon$);
- rh is the revocation handle for cred_{k+1} (e.g., chosen by the issuer before);
- $(\mathit{ipk}_i, RI_i)_{i=1}^{k}$ are the issuers' public keys and current revocation information for $(\mathit{cred}_i)_{i=1}^{k}$; $(\mathit{ipk}_{k+1}, RI_{k+1})$ correspond to the issuer of the new credential;
- $(\mathit{cred}_i)_{i=1}^{k}$ are credentials owned by the user and involved in this issuance and $\mathit{cred}_{k+1} = \varepsilon$ is a placeholder for the new credential to be issued;
- $R_i \subseteq [n_i]$ is the set of attribute indices for which the value is revealed;
- for $i \in [k]$, $(a_{i,j})_{j=1}^{n_i}$ is the list of attribute values certified in cred_i; $(a_{k+1,j})_{j \in R_{k+1}}$ are the attributes of cred_{k+1} that are revealed to the issuer;
- $(a_{k+1,j})_{j \notin R_{k+1}}$ are the attributes of cred_{k+1} that remain hidden;
- $((k+1, j), (i', j')) \in E$ means that the jth attribute of the new credential will have the same value as the j'th attribute of the i'th credential;
- $M \in \{0,1\}^*$ is a message to which the issuance token is to be bound;
- vupk is the user's vault public key;
- dbid is the device's binding identifier which can be set to ε if the new credential should not be device bound.

$\mathsf{ITVf} \xrightarrow{\$} \mathsf{accept/reject}$**.** On inputs $\big(\mathit{nym}, \mathit{pit}, \mathit{scope}_{\mathcal{U}}, \mathit{rh}, (\mathit{ipk}_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k+1}, E, M\big)$, this *issuance token verification* algorithm outputs whether to accept or to reject the issuance token. For $j = 1, \ldots, k$ all inputs are as before, but for $k+1$ they are for the new credential to be issued based on pit.

$\langle \mathcal{U}.\mathsf{Issue}; \mathcal{I}.\mathsf{Issue} \rangle \xrightarrow{\$} (\boldsymbol{cred}, \boldsymbol{RI'})$**.** In the interactive *issuance* protocol, the user takes input sit, whereas the issuer takes inputs $(\mathit{isk}, \mathit{pit}, RI)$, where pit has been verified by the issuer before. At the end of the protocol, the user obtains a credential cred as an output, while the issuer receives an updated revocation information RI'.

$\mathsf{Present} \xrightarrow{\$} (\boldsymbol{pt}, \boldsymbol{nym}, \boldsymbol{dnym})$**.** On input $\big(\mathit{usk}, \mathit{scope}_{\mathcal{U}}, \big(\mathit{ipk}_i, RI_i, \mathit{cred}_i, (a_{i,j})_{j=1}^{n_i}, R_i\big)_{i=1}^{k}, E, M, \mathit{dsk}, \mathit{scope}_{\mathcal{D}}, \big(\mathit{dipk}, RI_{\mathcal{DI}}, \mathit{dcred}\big), \mathcal{BD}, \mathit{vupk}\big)$, the *presentation* algorithm outputs a presentation token pt, a user pseudonym nym, and a device pseudonym $dnym$. Most of the inputs are as before, but:

- $E \subseteq \{(i,j) : i \in [k], j \in [n_i]\}$, where $((i,j),(i',j')) \in E$ means that the presentation token proves that $a_{i,j} = a_{i',j'}$ without revealing the actual attribute values. That is, E enables one to prove equality predicates;
- M is a message to which the presentation is bound. This might, e.g., be a nonce chosen by $\mathcal{V}$ to prevent replay attacks, where $\mathcal{V}$ uses a presentation token to impersonate a user towards another verifier;
- dsk is the user's device secret key stored in the secure element of the device;
- $scope_{\mathcal{D}}$ is the scope of the generated device pseudonym $dnym$;
- $dipk$ and $RI_{\mathcal{DI}}$ are the public key and current revocation information of the issuer of the device credential $dcred$;
- $\mathcal{BD} \subseteq [k]$ is a set of indices specifying which credentials are device-bound;
- $vupk$ is the user's vault public key;

$\mathsf{Verify} \xrightarrow{\$} \mathsf{accept/reject}$. The *presentation verification* algorithm takes $\big(pt, nym, scope_{\mathcal{U}}, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k}, E, M, dnym, scope_{\mathcal{D}}, dipk, RI_{\mathcal{DI}}, \mathcal{BD}\big)$ defined as before as inputs, and outputs whether to accept or to reject a presentation token. For notational convenience, we assume that a term like $(a_{i,j})_{j \in R_i}$ implicitly also describes the set R_i.

$\mathsf{Revoke} \xrightarrow{\$} \boldsymbol{RI'}$. The *revocation* algorithm takes as inputs (isk, RI, rh), where isk is the issuer's secret key, RI is the current revocation information, and rh is the revocation handle to be revoked. It outputs an updated revocation information RI'.

$\mathsf{BTGen} \xrightarrow{\$} (\boldsymbol{dnym}, \boldsymbol{pbt}, \boldsymbol{sbt})$. The *backup token generation* algorithm takes as input $\big(dsk, vupk, dipk, RI_{\mathcal{DI}}, dcred, dbid\big)$, where all the values are as before. It outputs a device pseudonym $dnym$ and a public/secret backup token (pbt, sbt), which will allow the user to carry over the current $dbid$ into a new device upon restoring. In theory, the entire backup token may be stored in a public cloud, as no adversary would be able to use it to get a new device credential re-issued. However, as sbt may contain personally identifying information, it is recommended to store sbt privately or only in an encrypted form.

$\langle \mathcal{U}.\mathsf{Restore}; \mathcal{DI}.\mathsf{Restore} \rangle \xrightarrow{\$} (\boldsymbol{dcred}, \boldsymbol{RI'_{\mathcal{DI}}})$. In the *device restoring* protocol, the user takes as input $(sbt, dipk, dmpk, vusk, dsk, dcred_{init}, drh)$, whereas the device issuer takes as input $(dnym, pbt, disk, dmpk, RI_{\mathcal{DI}}, drh)$. At the end of the protocol, the user outputs a fresh device credential $dcred$, while the issuer receives an updated revocation information $RI'_{\mathcal{DI}}$.

2.2 Security Definitions (Informal)

We now next describe the security properties of an attribute-based credential system with backup. Our definitions extend that of Camenisch et al. [CKL+14], who gave a comprehensive formal definitional framework for traditional credential schemes. As our system involves more types of parties and offers more interfaces, the formal definitions are quite complex, thus we only sketch the intuition here and refer to the full version for the formal definitions.

Oracles. In our definitions, the adversary is granted access to three oracles: a device manufacturer oracle $\mathcal{O}^{\texttt{producer}}$, an issuer oracle $\mathcal{O}^{\texttt{issuer}}$, and a user oracle $\mathcal{O}^{\texttt{user}}$, allowing the adversary to interact with honest device manufacturers, issuers, and users, respectively. While most of the interfaces of the oracles are natural, $\mathcal{O}^{\texttt{user}}$ has additional interfaces that allow the adversary to "steal" a device with a specific set of credentials from an honest user. Furthermore, the adversary is given interfaces to use such a device. His available options depend on the assumed security guarantees of the devices. We explicitly distinguish the following three types of devices, but our definitions are flexible enough to be easily adopted for other settings as well. First, if the devices are assumed to have secure memory and are protected by strong PINs, then the adversary essentially cannot profit from the stolen device at all. Second, if the memory is assumed to be secure but no strong PINs are used, then the adversary can use (i.e., backup, present, etc.) the stolen device and the credentials on them, but he does not learn the precise values of the user secret key or the credentials. Finally, if no assumption on the device are made, the adversary learns all the information stored on the device, including the user secret key and the credentials.

We believe that parameterizing our security definitions by the assumed security of the devices is useful to realistically model a broad range of real world scenarios, as, for instance, the security guarantees of eID cards, smart phones, or public transport tickets might drastically differ in practice. Clearly, making no assumptions on the devices results in the strongest definitions; however, as the computational capacity of embedded devices is often limited, our approach of considering additional properties is essential to obtain practical protocols.

Completeness. If all parties behave honestly, all protocols can be run successfully. In particular, honest users interacting with an honest counterpart can always activate, register, and restore devices, as well as obtain and present credentials.

Unforgeability. We define unforgeability as a game between an adversary and the $\mathcal{O}^{\texttt{producer}}$, $\mathcal{O}^{\texttt{issuer}}$, $\mathcal{O}^{\texttt{user}}$ oracles. The adversary can produce new devices, obtain and revoke credentials from honest issuers, instruct honest users to obtain credentials on inputs of his choice, request presentation tokens and receive backup tokens for given device credentials. Moreover, as mentioned above, he can "steal" devices and receive the device credential together with the software credentials and secret keys. At the end of the game the adversary outputs a number of presentation tokens and pseudonyms and wins if at least one of the presentation tokens is a forgery, or an issuance token successfully submitted to the honest issuer oracle was a forgery. A token is a forgery if it is *inconsistent* with the world the adversary interacts with using oracles $\mathcal{O}^{\texttt{producer}}$, $\mathcal{O}^{\texttt{issuer}}$, and $\mathcal{O}^{\texttt{user}}$. Informally, being consistent here means that for each token returned by the adversary, all software credentials are bound to the same use secret key, the pseudonym *nym* is sound for the given scope $scope_{\mathcal{U}}$, values of revealed attributes are correct, the equality relation E is satisfied, all credentials presented in the token are either bound to the same device binding identifier *dbid* or to ε and bound to the same vault public key. Moreover, presentation tokens must have been created using a valid device credential (issued for a genuine device using restore or register), the device pseudonym *dnym* is sound for the given scope $scope_{\mathcal{D}}$ and the device credential must be signed such that the signature is verifiable with the vault public key.

Privacy. Similar to Camenisch et al. [CKL+14], we define privacy via a simulation based approach. We consider a set of honest users and let the adversary control all other parties in the system (device manufacturers, issuers, and verifiers). It should be computational infeasible for the adversary to distinguish whether he is communicating with the real set of honest users, or with a simulator, S, that only has access to the public information of the respective protocol (e.g., revealed attributes, scopes, public keys). For this, we define a filter $\mathcal{F}$ that has the same interfaces as the user oracle $\mathcal{O}^{\mathtt{user}}$, but sanitizes the inputs beforeforwarding them to the S. For instance, unlinkability of presentation is ensured by not forwarding the credential identifiers *cid* (that the adversary gives to $\mathcal{O}^{\mathtt{user}}$) to S. Furthermore, the filter performs comprehensive book keeping to exclude trivial distinguishing attacks that would result from requesting presentation tokens from invalid credentials which would be answered by the S but not by $\mathcal{O}^{\mathtt{user}}$.

3 A Generic Construction and Its Security

One possible way to realize recoverable device-bound credentials would be to choose a unique "recovery secret" a for each device-bound credential and embed its image through a one-way function $\mathsf{f}(a)$ as an additional attribute in the credential. This attribute is not revealed during presentation, but to backup a credential *cred*, he stores a presentation token *pt* revealing $\mathsf{f}(a)$ on insecure backup media, and stores a in a secure offline vault. To restore *cred* upon loss, the user sends *pt* and a to the issuer, who checks if *pt* and $\mathsf{f}(a)$ are correct, and then participates in an advanced issuance protocol that allows the user to blindly carry over all attributes from *cred* into a new credential that will be bound to freshly chosen recovery secret a'.[2]

One drawback of this construction is that a malicious device issuer getting access to a user's device carrying some software credentials could simply reissue himself a new device credential, as he would then just omit the proof of knowledge for a. The device issuer could then use the user's software credentials with the new device, and the user would have no option to revoke the malicious device credential as it would contain a fresh revocation handle.

This problem could be mitigated by encrypting credentials before loading them onto the device. The decryption key could be stored inside the secure element of the device, and the credentials would only be decrypted in this secure environment. So a malicious device issuer finding a user's device would not learn

[2] Alternatively, one could also use the same a for all device-bound credentials, and then only prove in zero-knowledge that one knows the trapdoor a to the attribute $\mathsf{f}(a)$.

the user's software credentials and thus could not impersonate him. However, reality shows that virtually any tamperproof device can be broken by a sufficiently powerful adversary. In this case a user could again be impersonated.

One solution to this problem is to let the verifier not only check that the user owns a valid device credential, but also that the user "accepted" this credential. On a very high level, this can be done by letting the user sign his device credential, and embed the verification key of this signature into all his credentials as an attribute. Then, at presentation, the user shows that he owns a device credential and a signature thereon, and that the public verification key is also contained in all the other credentials. As a malicious device issuer may never learn the signing key of this user (as it is stored in a secure vault), he may not impersonate the user with a fresh device credential any more, as this would require forging a signature on this credential.

A bit more precisely, each user computes a signature key pair $(\mathit{vusk}, \mathit{vupk})$ and stores *vusk* in a bank vault as his *vault user secret key*, and only needs to access *vusk* when (re-)obtaining device credentials. After buying and activating a device, the user requests a device credential *dcred*, that certifies the validity of the device and a unique device binding identifier *dbid*. The device credential *dcred* is bound to *vupk*. All software credentials also get bound to *vupk*, and optionally to *dbid* if the credential is to be bound to a specific device; if no *dbid* is given, the credential can be used with any device. Upon loss of a device, the user now only needs to get *dcred* re-issued, but all the software credentials can be left unchanged. To ensure that only the legitimate user can re-obtain and later prove possession of a device credential, we let the user sign the (unique) revocation handle *rh* of *dcred* using *vusk*. Upon presentation, the user now not only shows that he possesses a device credential, but also that he knows a signature under *vupk* on *rh*. This signature protects against malicious credential issuers, which cannot create such proof of signature knowledge on a non-revoked token *rh*. Unfortunately, binding the credentials to *vupk* by adding it as an attribute does not work here generically, as this would require that the attribute space of the credential scheme subsumes the key (message) space of the signature scheme. Furthermore, standard signatures would require the verifier to learn the signature verification key *vupk*, which must not be revealed to maintain unlinkability. Therefore, upon device registration, the user commits to *vupk* and lets the issuer additionally sign this commitment. Using commuting signatures [Fuc11] additionally allows us to perform the required proofs with only publishing a commitment to *vupk*, but not *vupk* itself, therefore achieving the required privacy goals.

What is left, is to bind this signature to the specific credential. To do so, the issuer, instead of only signing the vault public key, signs a combination of *vupk* and the revocation handle, *rh*, for the credential issued. Thus, once the credential gets revoked, the signature of the issuer becomes useless. Since revocation handles do not necessarily belong to the message (key) space of commuting signatures, we sign the value $\phi(rh)$ instead of signing the actual revocation handle *rh*, for some appropriate mapping function ϕ. Finally, we also use a proof system that allows to verify that commitments to *rh* and $\phi(rh)$ are consistent.

3.1 Building Blocks

In the following, we briefly recap the non-standard building blocks required for the generic construction presented in this section.

Privacy-Enhancing Attribute-Based Credentials. Because of space limitations and to avoid redundancy, we refrain from giving formal definitions for PABC systems, but refer to Camenisch et al. [CKL+14].

Informally, anonymous credentials (without backup) have the same interfaces as introduced in Sect. 2.1, except for all the backup-related parts. That is, they consist of the following algorithms and protocols: $\mathsf{SPGen}, \mathsf{UKGen}, \mathsf{IKGen}, \mathsf{ITGen}, \mathsf{ITVf}, \langle \mathcal{U}.\mathsf{Issue}; \mathcal{I}.\mathsf{Issue} \rangle, \mathsf{Present}, \mathsf{Verify}, \mathsf{Revoke}$. The input/output behavior and the required security properties are again similar to Sects. 2.1 and 2.2, respectively.

Commuting Signatures and Verifiable Encryption. On a high level, commuting signatures combine digital signatures, encryption, and proof systems, such that one can commit to (any subset of) signature verification key, message, and signature, and still be able to prove that the three values are a valid key/message/signature tuple. In the following we give a slightly simplified version of the interfaces introduced by Fuchsbauer [Fuc11].

$\mathsf{SPGen}_{\mathsf{CS}}$. On input global system parameters $spar_{\mathrm{g}}$, this *system parameter generation* algorithms outputs signature parameters $spar_{\mathsf{CS}}$. These system parameters are input to all algorithms in the following, but we will sometimes omit this for notational convenience.

$\mathsf{KeyGen}_{\mathsf{CS}}$. On input $spar_{\mathsf{CS}}$, this *key generation* algorithm outputs a signature key pair (sk, pk).

$\mathsf{Com}_{\mathsf{CS}}$. On input a message m in the signature or the key space, this *commitment* algorithm outputs a commitment $cs_{\mathcal{M}}$ and opening information $ocs_{\mathcal{M}}$.

$\mathsf{Com}_{\mathcal{M}}$. On input a message m from the message space, this *commitment* algorithm extends $\mathsf{Com}_{\mathsf{CS}}$ by, among others, proofs of consistency. Note that the key space is consistent with the message space and therefore one can also use this algorithm to commit to verification keys.

$\mathsf{DCom}_{\mathsf{CS}}$. On input a commitment $cs_{\mathcal{M}}$ and opening $ocs_{\mathcal{M}}$, this *decommitment* algorithm outputs the committed message m.

$\mathsf{CombCom}_{\mathsf{CS}}$. On input two commitment/opening pairs $(cs_1, ocs_1), (cs_2, ocs_2)$ to m_1, m_2 in the message or key space, this *commitment combining* algorithm outputs a commitment/opening pair (cs_3, ocs_3) of type $\mathsf{Com}_{\mathcal{M}}$ to $m_1 \otimes m_2$.

$\mathsf{VerCombCom}_{\mathsf{CS}}$. On input three commitments cs_i, $i = 1, 2, 3$, this *combined commitment verification* algorithm outputs 1, if and only if cs_3 is the output of $\mathsf{CombCom}_{\mathsf{CS}}$ on input cs_1, cs_2.

$\mathsf{SigCom}_{\mathsf{CS}}$. On input a secret key sk and a commitment $cs_{\mathcal{M}}$, this *commitment signing* algorithm outputs a signature σ, a commitment/opening pair (cs_σ, ocs_σ) to σ, and a proof π_σ of the validity of the signature.

```
$\pi'_\sigma \xleftarrow{\$} \mathsf{AdPrC}_{\mathcal{M}}(pk, (cs_{\mathcal{M}}, ocs_{\mathcal{M}}), (cs_\sigma), \pi_\sigma)$
$\pi'_\sigma \xleftarrow{\$} \mathsf{AdPrDC}(pk, (cs_{\mathcal{M}}), (cs_\sigma, ocs_\sigma), \pi'_\sigma)$
$(cs'_\sigma, ocs'_\sigma) \xleftarrow{\$} \mathsf{Com}_{\mathcal{M}}(\sigma)$
$\pi'_\sigma \xleftarrow{\$} \mathsf{AdPrC}(pk, (cs_{\mathcal{M}}), (cs'_\sigma, ocs'_\sigma), \pi'_\sigma)$
if $cs_{pk} \neq \varepsilon$ and $ocs_{pk} \neq \varepsilon$:
    $\pi'_\sigma \xleftarrow{\$} \mathsf{AdPrC}_{\mathcal{K}}((cs_{pk}, ocs_{pk}), (cs_{\mathcal{M}}), (cs_{\sigma,new}), \pi'_\sigma)$
Output $((cs'_\sigma, ocs'_\sigma), \pi'_\sigma)$
```

Fig. 2. $\mathsf{RandSign}((pk, cs_{pk}, ocs_{pk}), (cs_{\mathcal{M}}, ocs_{\mathcal{M}}), (\sigma, cs_\sigma, ocs_\sigma, \pi_\sigma))$

$\mathsf{Verify}_{\mathsf{CS}}$**.** On input a public key pk, a message m, a signature σ (or commitments to (some of) these values), and a proof π, this *verification* algorithm outputs accept if and only if π is a valid proof that σ is a signature on m for the public key pk.

AdPrC**.** On input pk, $cs_{\mathcal{M}}$, (cs_σ, ocs_σ), and π such that $\mathsf{Verify}_{\mathsf{CS}}(pk, cs_{\mathcal{M}}, \sigma, \pi) = \mathsf{accept}$ (where σ is obtained using $\mathsf{DCom}_{\mathsf{CS}}$), this *committing proof adaption* algorithm outputs a proof π' such that $\mathsf{Verify}_{\mathsf{CS}}(pk, cs_{\mathcal{M}}, cs_\sigma, \pi') = \mathsf{accept}$; to decommit in a proof, AdPrDC works the other way round (i.e., adapts π such that it verifies for σ and not for cs_σ).

$\mathsf{AdPrC}_{\mathcal{M}}$**.** On input pk, $(cs_{\mathcal{M}}, ocs_{\mathcal{M}})$, cs_σ, and π such that $\mathsf{Verify}_{\mathsf{CS}}(pk, m, cs_\sigma, \pi) = \mathsf{accept}$, this *committing proof adaption* algorithm outputs π' such that $\mathsf{Verify}_{\mathsf{CS}}(pk, cs_{\mathcal{M}}, cs_\sigma, \pi') = \mathsf{accept}$; again, $\mathsf{AdPrDC}_{\mathcal{M}}$ works the other way round,

$\mathsf{AdPrC}_{\mathcal{K}}$**.** On input (cs_{pk}, ocs_{pk}), $cs_{\mathcal{M}}$, cs_σ, and π such that $\mathsf{Verify}_{\mathsf{CS}}(pk, cs_{\mathcal{M}}, cs_\sigma, \pi) = \mathsf{accept}$, the *committing proof adaption* algorithm outputs π' such that $\mathsf{Verify}_{\mathsf{CS}}(cs_{pk}, cs_{\mathcal{M}}, cs_\sigma, \pi') = \mathsf{accept}$; $\mathsf{AdPrDC}_{\mathcal{K}}(cs_{pk}, cs_{\mathcal{M}}, cs_\sigma, \pi)$ works the other way round.

We require that $(\mathsf{Com}_{\mathsf{CS}}, \mathsf{DCom}_{\mathsf{CS}})$ is a secure extractable commitment scheme. We also require **strong unforgeability** (under chosen message attack), i.e., the adversary cannot output a new pair message/signature (m, σ). Moreover, all the proofs must be simulatable using an appropriate trapdoor, for details we refer to Fuchsbauer [Fuc11].

In addition to the above algorithms, Fig. 2 specifies the algorithm $\mathsf{RandSign}$ that we will use in our construction. The procedure takes as input a commuting signature for which the signature is given as commitment and adapts it to a commuting signature for which the signature, the message and optionally the public key are given as commitments (the commitment to signature is re-randomized), such that the inputs cs_σ and π_σ cannot be linked to cs'_σ and π'_σ.

3.2 The Construction

In the following we show how to build a BPABC system from a PABC system and a commuting signature scheme $(\mathsf{Com}_{\mathcal{M}}, \dots)$. In the construction, let $\mathsf{H}_{\mathcal{AS}} : \{0,1\}^* \to \mathcal{AS}$ and $\mathsf{H}_{\mathcal{MS}} : \{0,1\}^* \to \mathcal{MS}$ be collision-resistant hash functions, where $\mathcal{AS}$ is the attribute space of the PABC system, and $\mathcal{MS}$ is the message space of presentation tokens used in the PABC system. Let $(\mathsf{Com_c}, \dots)$ be a standard commitment scheme. Furthermore, let ϕ be a homomorphism from the message space of $\mathsf{Com_c}$ to the message space of $\mathsf{Com}_\mathcal{M}$, that additionally has the property that $\phi(m_1) \otimes \phi(m_2) = \phi(m_1 \oplus m_2)$, where $m_1, m_2, m_1 \oplus m_2$ are in the message space of $\mathsf{Com_c}$. Finally, let $(\mathsf{Prove}_\phi, \mathsf{Verify}_\phi)$ be a zero-knowledge proof system for statements:

$$\mathsf{ZKP}\left[(\alpha) : c_1 = \mathsf{Com}_\mathcal{M}(\phi(\alpha)) \ \wedge \ c_2 = \mathsf{Com_c}(\alpha)\right].$$

Below we give a simplified generic construction of our BPABC scheme, based on the PABC credential scheme [CKL+14] and commuting signatures. The complete construction is given in the full version and actually does not build upon PABC's, but rather extends the generic PABC-construction, as we require access to the commitment values produced and consumed by the building blocks of the PABC scheme. In the description below, we assume, for the sake of simplicity, that we can extract the commitments from PABC presentation and issuance tokens, which allows us to focus on the extensions needed to obtain the BPABC.

Setup and Key Generation

SPGen: The system parameters *spar* consist of the parameters $spar_\mathsf{PABC}$ of the PABC system, $spar_\mathsf{CS}$ of the commuting signature scheme, and two attributes $\{\mathsf{initial}, \mathsf{activated}\} \in \mathcal{AS}$. These parameters in particular specify all required domains, e.g., of revocation handles, etc.

UKGen: As in PABC, i.e., compute the user key as $usk \xleftarrow{\$} \mathsf{UKGen}(spar_\mathsf{PABC})$.

VKGen: Compute the vault keys as $(vusk, vupk) \xleftarrow{\$} \mathsf{KeyGen_{CS}}(spar_\mathsf{CS})$, e.g., being keys for the commuting signature scheme.

IKGen: The issuer key consists of an issuer's key for the PABC system and a key for the commuting signature scheme. For device credential issuers the key also comprises two scopes and list to keep track of used pseudonyms:

– Compute $(isk_\mathsf{PABC}, ipk_\mathsf{PABC}, RI_\mathsf{PABC}) \xleftarrow{\$} \mathsf{IKGen_{PABC}}(spar_\mathsf{PABC})$.
– Compute $(isk_\mathsf{CS}, ipk_\mathsf{CS}) \xleftarrow{\$} \mathsf{KeyGen_{CS}}(spar_\mathsf{CS})$.
– Set $ipk = (ipk_\mathsf{PABC}, ipk_\mathsf{CS})$, $isk = (isk_\mathsf{PABC}, isk_\mathsf{CS}, ipk)$, and $RI = RI_\mathsf{PABC}$.
– For *device credential issuers* further generate two scopes $scope_{bup}, scope_{reg} \xleftarrow{\$} \mathcal{SCP}$ (for backup pseudonyms and for registration pseudonyms) and an empty list $\mathcal{L}_{dnym}$ to store used pseudonyms.
  Set $dipk = (ipk, scope_{bup}, scope_{reg})$, $disk = isk$, and $RI_\mathcal{DI} = (RI, \mathcal{L}_{dnym})$.

DMKGen: The device manufacturer's key consists of an issuer's key for the PABC system, a scope for activation pseudonyms and an empty list $\mathcal{L}_\mathtt{cert}$ to store used pseudonyms:

– Compute $(dmsk_\mathsf{PABC}, dmpk_\mathsf{PABC}, RI_\mathsf{PABC}) \xleftarrow{\$} \mathsf{IKGen_{PABC}}(spar_\mathsf{PABC})$.
– Generate empty list $\mathcal{L}_\mathtt{cert}$ and $scope_{act} \xleftarrow{\$} \mathcal{SCP}$.

– Set $dmpk = (dmpk_{\mathsf{PABC}}, RI_{\mathsf{PABC}}, \mathcal{L}_{\mathtt{cert}}, scope_{act})$, and the secret key to $dmsk = (dmsk_{\mathsf{PABC}}, dmpk_{\mathsf{PABC}})$.

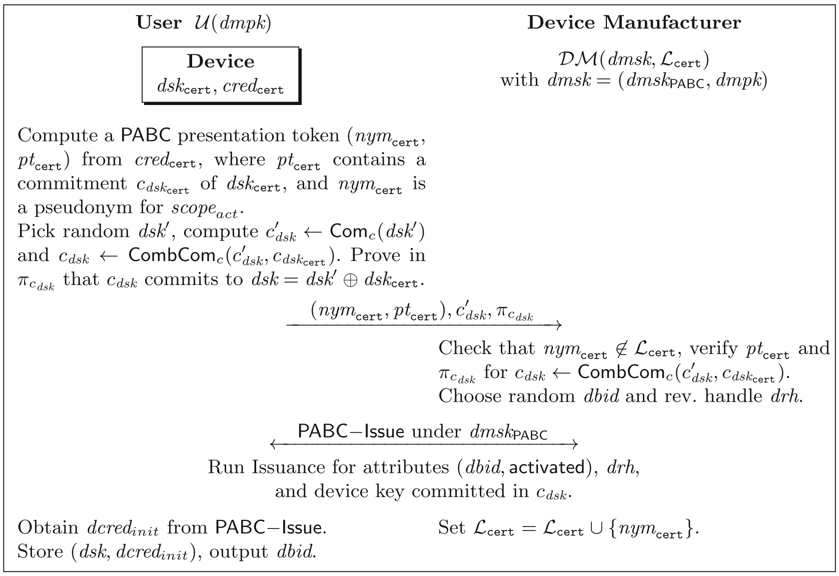
Produce: Generate an initial device secret key $dsk_{\mathtt{cert}}$ and issue a device credential under $dmsk$.

– Compute device secret key $dsk_{\mathtt{cert}} \overset{\$}{\leftarrow} \mathsf{UKGen}_{\mathsf{PABC}}(spar_{\mathsf{PABC}})$.
– Choose a random revocation handle drh.
– Issue (locally) a PABC credential $cred_{\mathtt{cert}}$ for the attribute initial, device key $dsk_{\mathtt{cert}}$ and revocation handle drh under the device manufacturer's issuance key $dmsk_{\mathsf{PABC}}$.
– Set $\mathtt{cert} = (cred_{\mathtt{cert}}, dsk_{\mathtt{cert}})$.

Device Activation and Registration

Activate: To activate a device, the user runs a protocol with the device manufacturer where $\mathcal{U}$ derives a new device secret key dsk as $dsk = dsk' \oplus dsk_{\mathtt{cert}}$ for a randomly chosen dsk'. The user also obtains a credential on dsk from $\mathcal{DM}$. Thereby, $\mathcal{DM}$ can verify that dsk is correctly derived from a previously certified $dsk_{\mathtt{cert}}$ but does not learn the new device key. A simplified version of the activation protocol is depicted in Fig. 3. For the sake of simplicity, we therein omit the openings from the in- and outputs of the commitment algorithms.

Register: To register a device with a device credential issuer $\mathcal{DI}$, the user (assisted by his device) runs a protocol with $\mathcal{DI}$. Therein, $\mathcal{U}$ shows that he has an activation credential $dcred_{init}$ from $\mathcal{DM}$ on some (secret) device key dsk and identifier $dbid$. The device credential issuer then blindly carries over dsk, $dbid$ into a


User $\mathcal{U}(dmpk)$ | **Device Manufacturer**

Device $dsk_{\mathtt{cert}}, cred_{\mathtt{cert}}$

$\mathcal{DM}(dmsk, \mathcal{L}_{\mathtt{cert}})$ with $dmsk = (dmsk_{\mathsf{PABC}}, dmpk)$

Compute a PABC presentation token $(nym_{\mathtt{cert}}, pt_{\mathtt{cert}})$ from $cred_{\mathtt{cert}}$, where $pt_{\mathtt{cert}}$ contains a commitment $c_{dsk_{\mathtt{cert}}}$ of $dsk_{\mathtt{cert}}$, and $nym_{\mathtt{cert}}$ is a pseudonym for $scope_{act}$.
Pick random dsk', compute $c'_{dsk} \leftarrow \mathsf{Com}_c(dsk')$ and $c_{dsk} \leftarrow \mathsf{CombCom}_c(c'_{dsk}, c_{dsk_{\mathtt{cert}}})$. Prove in $\pi_{c_{dsk}}$ that c_{dsk} commits to $dsk = dsk' \oplus dsk_{\mathtt{cert}}$.

$$\xrightarrow{(nym_{\mathtt{cert}}, pt_{\mathtt{cert}}), c'_{dsk}, \pi_{c_{dsk}}}$$

Check that $nym_{\mathtt{cert}} \notin \mathcal{L}_{\mathtt{cert}}$, verify $pt_{\mathtt{cert}}$ and $\pi_{c_{dsk}}$ for $c_{dsk} \leftarrow \mathsf{CombCom}_c(c'_{dsk}, c_{dsk_{\mathtt{cert}}})$.
Choose random $dbid$ and rev. handle drh.

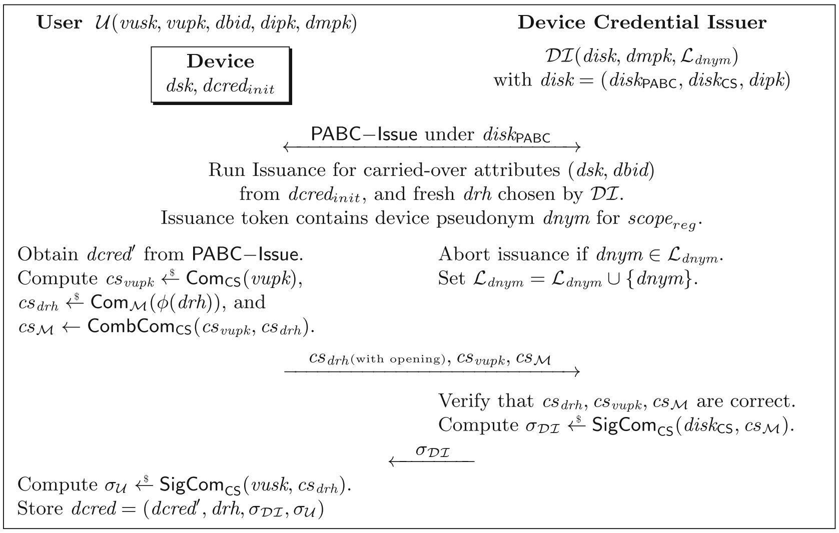
$$\xleftrightarrow{\mathsf{PABC{-}Issue}\text{ under } dmsk_{\mathsf{PABC}}}$$

Run Issuance for attributes $(dbid, \mathsf{activated})$, drh, and device key committed in c_{dsk}.

Obtain $dcred_{init}$ from PABC−Issue.
Store $(dsk, dcred_{init})$, output $dbid$.

Set $\mathcal{L}_{\mathtt{cert}} = \mathcal{L}_{\mathtt{cert}} \cup \{nym_{\mathtt{cert}}\}$.

Fig. 3. Device activation



User $\mathcal{U}(vusk, vupk, dbid, dipk, dmpk)$

Device $dsk, dcred_{init}$

Device Credential Issuer $\mathcal{DI}(disk, dmpk, \mathcal{L}_{dnym})$ with $disk = (disk_{\mathsf{PABC}}, disk_{\mathsf{CS}}, dipk)$

$\xleftrightarrow{\mathsf{PABC{-}Issue} \text{ under } disk_{\mathsf{PABC}}}$

Run Issuance for carried-over attributes $(dsk, dbid)$ from $dcred_{init}$, and fresh drh chosen by $\mathcal{DI}$. Issuance token contains device pseudonym $dnym$ for $scope_{reg}$.

Obtain $dcred'$ from $\mathsf{PABC{-}Issue}$. Compute $cs_{vupk} \xleftarrow{\$} \mathsf{Com}_{\mathsf{CS}}(vupk)$, $cs_{drh} \xleftarrow{\$} \mathsf{Com}_{\mathcal{M}}(\phi(drh))$, and $cs_{\mathcal{M}} \leftarrow \mathsf{CombCom}_{\mathsf{CS}}(cs_{vupk}, cs_{drh})$.

Abort issuance if $dnym \in \mathcal{L}_{dnym}$. Set $\mathcal{L}_{dnym} = \mathcal{L}_{dnym} \cup \{dnym\}$.

$\xrightarrow{cs_{drh}(\text{with opening}),\ cs_{vupk},\ cs_{\mathcal{M}}}$

Verify that $cs_{drh}, cs_{vupk}, cs_{\mathcal{M}}$ are correct. Compute $\sigma_{\mathcal{DI}} \xleftarrow{\$} \mathsf{SigCom}_{\mathsf{CS}}(disk_{\mathsf{CS}}, cs_{\mathcal{M}})$.

$\xleftarrow{\sigma_{\mathcal{DI}}}$

Compute $\sigma_{\mathcal{U}} \xleftarrow{\$} \mathsf{SigCom}_{\mathsf{CS}}(vusk, cs_{drh})$. Store $dcred = (dcred', drh, \sigma_{\mathcal{DI}}, \sigma_{\mathcal{U}})$

Fig. 4. Device registration

new credential that also includes a revocation handle *drh*. In addition, $\mathcal{DI}$ also signs a commitment on a combination of *drh* and the user's vault public key *vupk*, and the user signs a commitment of $\phi(drh)$, both using the commuting signature scheme. The registration information then consists of the credential and both signatures. The simplified registration protocol is given in Fig. 4. For simplicity, we again omit the openings from the in- and outputs of the commitment algorithms.

Credential Issuance and Presentation

ITGen: This algorithm produces an issuance token for input $(usk, scope_{\mathcal{U}}, rh, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k+1}, E, M, vupk, dbid_{\mathcal{D}})$ and combines an PABC issuance token with a commuting signature as follows:

1. *Compute* PABC *issuance token.* First, the input to $\mathsf{ITGen}_{\mathsf{PABC}}$ needs to be prepared such that is also captures the device-binding property, i.e., the relation E gets extended to express which credentials are device-bound, and that they are all bound to the same device. More precisely, given k credentials $cred_i = (cred'_i, dbid_i, rh_i, (\sigma_{\mathcal{I},i}, cs_{\sigma,i}, ocs_{\sigma,i}, \pi_{\sigma,i}))$, one first verifies whether all device-bound credentials with $dbid_i \neq \varepsilon$ contain the same device identifier $dbid^*$. We denote $\mathcal{BD}$ for the set of all device-bound credentials. If at least one credential is device-bound, then the new credential should be also device-bound as well, i.e., verify that $dbid_{\mathcal{D}} = dbid^*$ and set $dbid_{k+1} = dbid_{\mathcal{D}}$. Abort with output $\perp$ if any of the checks fails. Update the relation expression to $E' = E \cup \{((i, n_i + 1), (j, n_j + 1)) : i, j \in \mathcal{BD} \cup \{k + 1\}\}$ and use E' to obtain the PABC issuance token. That is, we run $\mathsf{ITGen}_{\mathsf{PABC}}(usk, scope_{\mathcal{U}}, rh, (ipk_i, RI_i, cred'_i, ((a_{i,j})_{j=1}^{n_i}, dbid_i), R_i)_{i=1}^{k+1}, E', M)$ receiving (nym', pit', sit').

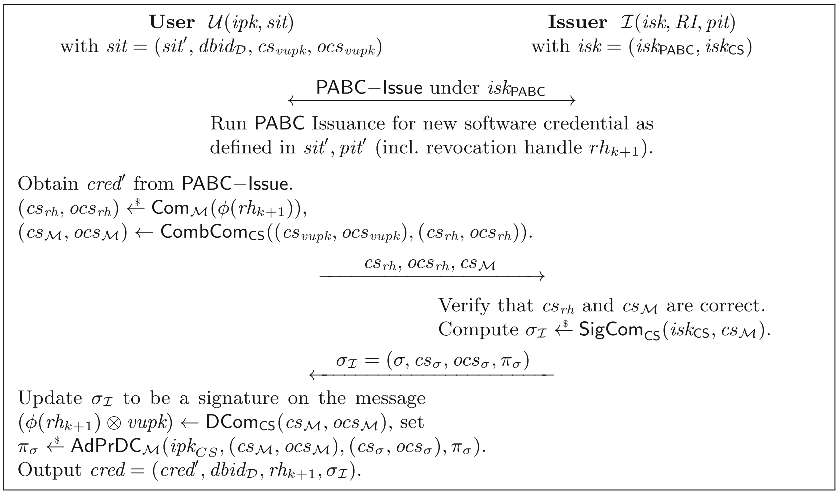
2. *Adapt and randomize the Issuer's signatures* $\sigma_{\mathcal{I},i}$. The second part is based on the commuting signatures the user has obtained for all credentials $cred_i$. The issuer's signatures $\sigma_{\mathcal{I},i} = (\sigma_i, cs_{\sigma,i}, ocs_{\sigma,i}, \pi_{\sigma,i})$ originally signed $\phi(rh_i) \otimes vupk$. We now adapt them to be signatures on fresh commitments of the same messages: we first compute CS commitments $cs_{rh,i}$ (with openings $ocs_{rh,i}$) for the individual revocation handles $\phi(rh_i)$ of all credentials. Then, we compute a CS commitment cs_{vupk} (with opening ocs_{vupk}) for the user's vault public key $vupk$ and combine cs_{vupk} with each $cs_{rh,i}$ via $\mathsf{CombCom}_{\mathsf{CS}}$ to obtain commitments $cs_{\mathcal{M},i}$ for messages $\phi(rh_i) \otimes vupk$. We then adopt and re-randomize the issuer's signatures using $\mathsf{RandSign}$, which adapts the committed signatures to be signatures on the freshly computed commitments $cs_{\mathcal{M},i}$. We denote the randomized and adapted signatures as $(cs'_{\sigma,i}, \pi'_{\sigma,i})$.
3. *Combine both parts.* Finally, we have to bind the PABC and the commuting signature part together. This is done by proving that the revocation handles committed in $cs_{rh,i}$ are the same revocation handles to which the PABC-issuance token commits as $(c_{rh,i}, o_{rh,i})$ (which we extract from pit'). For each rh_i we therefore compute $\pi^{\phi}_{rh,i} \xleftarrow{\$} \mathsf{Prove}_{\phi}((c_{rh,i}, o_{rh,i}), (cs_{rh,i}, ocs_{rh,i}))$. We set $pit = (pit', \mathcal{BD}, cs_{vupk}, (cs_{\mathcal{M},i}, cs_{rh,i}, \pi^{\phi}_{rh,i}, cs'_{\sigma,i}, \pi'_{\sigma,i})_{i=1}^{k})$, $sit = (sit', dbid_{\mathcal{D}}, cs_{vupk}, ocs_{vupk})$ and $nym = nym'$.

<u>ITVf</u>: Verify the PABC issuance token pit', proofs $\pi^{\phi}_{rh,i}$, that $(cs'_{\sigma,i}, \pi'_{\sigma,i})$ are valid issuer signatures on $cs_{\mathcal{M},i}$, and that $\mathsf{VerCombCom}_{\mathsf{CS}}(cs_{rh_i}, cs_{vupk}, cs_{\mathcal{M},i}) = 1$ for $i = 1, \ldots, k$. Output reject if any of the check fails, and accept otherwise.

<u>Issue</u>: Issuance of a software credential (possibly bound to a device if $dbid_{\mathcal{D}} \neq \varepsilon$) consists of a PABC issuance protocol, and a commuting signature generated by the issuer as depicted in Fig. 5.

<u>Present</u>: A presentation token for $(usk, scope_{\mathcal{U}}, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M, dsk, scope_{\mathcal{D}}, (dipk, RI_{\mathcal{DI}}, dcred), \mathcal{BD}, vupk)$ consists of two PABC presentation tokens (one for the software credentials, and one for the device credential) and randomized commuting signatures:

1. *Compute* PABC *presentation token* pt' *for the software credentials.* Parse each credential as $cred_i = (cred'_i, dbid_i, rh_i, \sigma_{\mathcal{I},i})$. Adapt the relation E to include the device-binding relations, i.e., use the indices in set $\mathcal{BD}$ to compute $E' = E \cup \{((i, n_i+1), (j, n_j+1)) : i, j \in \mathcal{BD}\}$. Compute (nym, pt') for all software credentials $cred_1, \ldots, cred_k$ by running $\mathsf{Present}_{\mathsf{PABC}}(usk, scope_{\mathcal{U}}, (ipk_i, RI_i, cred'_i, ((a_{i,j})_{j=1}^{n_i}, dbid_i), R_i)_{i=1}^{k}, E', M)$.
2. *Compute* PABC *presentation token* pt'' *for the device credential.* If $dcred \neq \varepsilon$, parse $dcred = (dcred', drh, \sigma_{\mathcal{DI}}, \sigma_{\mathcal{U}})$, and compute presentation token $(dnym, pt'')$ as $\mathsf{Present}_{\mathsf{PABC}}(dsk, scope_{\mathcal{D}}, (ipk_{\mathcal{DI}}, RI_{\mathcal{DI}}, dcred', (dbid_{\mathcal{D}})), \emptyset, M)$. If $dcred = \varepsilon$ abort with output $\perp$.
3. *Bind* pt' *and* pt'' *together.* Extract $(c_{dbid,\mathcal{D}}, o_{dbid,\mathcal{D}})$ from pt'' and, for some $i \in \mathcal{BD}$, $(c_{dbid,i}, o_{dbid,i})$ from pt'. Prove that both commit to the same value in π_{dbid}.



User $\mathcal{U}(ipk, sit)$ with $sit = (sit', dbid_{\mathcal{D}}, cs_{vupk}, ocs_{vupk})$

Issuer $\mathcal{I}(isk, RI, pit)$ with $isk = (isk_{\mathsf{PABC}}, isk_{\mathsf{CS}})$

$\xleftrightarrow{\mathsf{PABC-Issue}\text{ under } isk_{\mathsf{PABC}}}$

Run PABC Issuance for new software credential as defined in sit', pit' (incl. revocation handle rh_{k+1}).

Obtain $cred'$ from $\mathsf{PABC-Issue}$.
$(cs_{rh}, ocs_{rh}) \xleftarrow{\$} \mathsf{Com}_{\mathcal{M}}(\phi(rh_{k+1}))$,
$(cs_{\mathcal{M}}, ocs_{\mathcal{M}}) \leftarrow \mathsf{CombCom}_{\mathsf{CS}}((cs_{vupk}, ocs_{vupk}), (cs_{rh}, ocs_{rh}))$.

$\xrightarrow{cs_{rh},\, ocs_{rh},\, cs_{\mathcal{M}}}$

Verify that cs_{rh} and $cs_{\mathcal{M}}$ are correct.
Compute $\sigma_{\mathcal{I}} \xleftarrow{\$} \mathsf{SigCom}_{\mathsf{CS}}(isk_{\mathsf{CS}}, cs_{\mathcal{M}})$.

$\xleftarrow{\sigma_{\mathcal{I}} = (\sigma, cs_{\sigma}, ocs_{\sigma}, \pi_{\sigma})}$

Update $\sigma_{\mathcal{I}}$ to be a signature on the message
$(\phi(rh_{k+1}) \otimes vupk) \leftarrow \mathsf{DCom}_{\mathsf{CS}}(cs_{\mathcal{M}}, ocs_{\mathcal{M}})$, set
$\pi_{\sigma} \xleftarrow{\$} \mathsf{AdPrDC}_{\mathcal{M}}(ipk_{CS}, (cs_{\mathcal{M}}, ocs_{\mathcal{M}}), (cs_{\sigma}, ocs_{\sigma}), \pi_{\sigma})$.
Output $cred = (cred', dbid_{\mathcal{D}}, rh_{k+1}, \sigma_{\mathcal{I}})$.

Fig. 5. Credential issuance

4. *Adapt and randomize the Issuer's signatures $\sigma_{\mathcal{I},i}$ and $\sigma_{\mathcal{DI}}$.* Similarly as in the ITGen algorithm we adapt the issuer's signatures $\sigma_{\mathcal{I},i} = (\sigma_i, cs_{\sigma,i}, ocs_{\sigma,i}, \pi_{\sigma,i})$ on $\phi(rh_i) \otimes vupk$ to be signatures on fresh commitments of the same messages. To this end, we first compute CS commitments $cs_{rh,i}$ (with openings $ocs_{rh,i}$) for the individual revocation handles $\phi(rh_i)$ for all $i = 1, \ldots, k, \mathcal{D}$. Then, we compute a CS commitment cs_{vupk} (with opening ocs_{vupk}) for the user's vault public key $vupk$ and combine cs_{vupk} with each $cs_{rh,i}$ via $\mathsf{CombCom}_{\mathsf{CS}}$ to obtain commitments $cs_{\mathcal{M},i}$ for messages $\phi(rh_i) \otimes vupk$. We then adopt and re-randomize the issuer's signatures using $\mathsf{RandSign}$, which adapts the committed signatures to be signatures on the freshly computed commitments $cs_{\mathcal{M},i}$. We denote the randomized and adapted signatures for all $i = 1, \ldots, k, \mathcal{D}$ as $(cs'_{\sigma,i}, \pi'_{\sigma,i})$, where $(cs'_{\sigma,\mathcal{D}}, \pi'_{\sigma,\mathcal{D}})$ stands for the adapted signature of the device credential issuer.
5. *Adapt and randomize the User's signature $\sigma_{\mathcal{U}}$ (contained in dcred).* In a similar vein, we adapt the user's signature $\sigma_{\mathcal{U}} = (\sigma, cs_{\sigma,\mathcal{U}}, ocs_{\sigma,\mathcal{U}}, \pi_{\sigma,\mathcal{U}})$ on the revocation handle of the device credential $\phi(drh)$ using procedure $\mathsf{RandSign}$. However, here we also adapt the proof using a commitment to $vupk$. Note that this ensures that the final signature proof is not only for a committed signature on a committed message, but that also the public key is given as a commitment. This is required here, as the user's public key would serve as a unique identifier otherwise. We denote the randomized user signature on message $cs_{rh,\mathcal{D}}$ and under key cs_{vupk} (computed in the step above) as $(cs'_{\sigma,\mathcal{U}}, \pi'_{\sigma,\mathcal{U}})$.
6. *Bind the* PABC *part and* CS *part together.* This is again similar to the ITGen algorithm: we bind both parts together by extracting the commitments for all revocation handles from the PABC tokens and proving that they contain the

same values as the CS commitments used in the commuting signatures part. That is, for all $i = 1, \ldots, k$ we extract $(c_{rh,i}, o_{rh,i})$ from pt' and $(c_{rh,\mathcal{D}}, o_{rh,\mathcal{D}})$ from pt'' and compute $\pi^{\phi}_{rh,i} \xleftarrow{\$} \mathsf{Prove}_{\phi}((c_{rh,i}, o_{rh,i}), (cs_{rh,i}, ocs_{rh,i}))$.
Finally, we compose the presentation token as $pt = ((pt', pt'', \pi_{dbid}, cs_{vupk}, (cs_{\mathcal{M},i}, cs_{rh,i}, \pi^{\phi}_{rh,i}, cs'_{\sigma,i}, \pi'_{\sigma,i})^{k,\mathcal{D}}_{i=1}, cs'_{\sigma,\mathcal{U}}, \pi'_{\sigma,\mathcal{U}}), nym, dnym)$.

<u>Verify</u>: Verify both presentation tokens (nym, pt'), $(dnym, pt'')$, proofs π_{dbid}, $\pi^{\phi}_{rh,i}$ and that all $cs_{\mathcal{M},i}$ are valid combinations of commitments $cs_{rh,i}$ and cs_{vupk} (for $i = 1, \ldots, k, \mathcal{D}$). Moreover, verify the credential issuer's signatures under $cs_{\mathcal{M},i}$ and the user's signature under $cs_{rh,\mathcal{D}}$ and for key cs_{vupk}. Output reject if any of the checks fails, and accept otherwise.

<u>Revoke</u>: On input (isk, RI, rh) parse isk as $(isk_{\mathsf{PABC}}, isk_{\mathsf{CS}}, ipk)$ and run the PABC revocation algorithm $\mathsf{Revoke}_{\mathsf{PABC}}(isk_{\mathsf{PABC}}, RI_{\mathsf{PABC}}, rh)$.

Backup and Restore

<u>BTGen</u>: Upon input $(dsk, vupk, dipk, RI_{\mathcal{DI}}, dcred, dbid_{\mathcal{D}})$ parse $dcred = (dcred', drh, \sigma_{\mathcal{DI}}, \sigma_{\mathcal{U}})$ and compute the presentation token $(dnym, pt) \leftarrow \mathsf{Present}_{\mathsf{PABC}} (dsk, scope_{bup}, (ipk_{\mathcal{DI}}, RI_{\mathcal{DI}}, dcred', (dbid_{\mathcal{D}})), \emptyset, M)$ for the device credential. Extract the commitment and opening (c_{dbid}, o_{dbid}) to the device identifier $dbid_{\mathcal{D}}$ and commitment and opening (c_{drh}, o_{drh}) to the device revocation handle. Set the public part to $pbt = (pt, drh, o_{drh})$, $sbt = (dbid, c_{dbid}, o_{dbid})$ and output $(dnym, pbt, sbt)$.

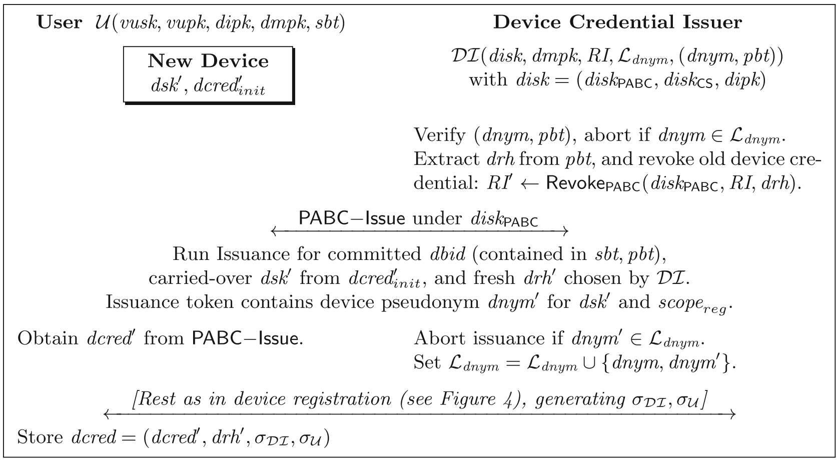
<u>Restore</u>: The restore procedure is initiated by the user when he obtained his new (and activated) device including a new device key dsk' and activation credential $dcred'_{init}$. The user then runs the restore protocol with the device credential issuer $\mathcal{DI}$ where he obtains a new device credential $cred'$ that contains the device identifier $dbid$ from the backup token. The simplified protocol is given in Fig. 6.

Theorem 1 (Informal). *The* BPABC*-system constructed above is unforgeable for all types of devices, if the underlying* PABC*- and the* CS*-schemes are unforgeable and* Prove_{ϕ} *is extractable and sound. It is further private for devices with secure memory without PINs, if the* PABC*-scheme is private,* Prove_{ϕ} *is zero-knowledge, and the* CS *scheme can be simulated given commitments to messages or/and verification keys.*

3.3 Intuition Underlying the Security Proofs

Due to space limitations, we only give the intuition of our security proofs.

Privacy. Our generic construction can be shown to be private, if one assumes secure memory without strong PINs, cf. Sect. 2.2. Note that without assuming secure memory, the adversary could in particular learn the user's secret key from which it could (deterministically) derive pseudonyms for arbitrary scopes, and could thus easily link arbitrary presentations to the user; achieving privacy



User $\mathcal{U}(vusk, vupk, dipk, dmpk, sbt)$

Device Credential Issuer

New Device
$dsk', dcred'_{init}$

$\mathcal{DI}(disk, dmpk, RI, \mathcal{L}_{dnym}, (dnym, pbt))$
with $disk = (disk_{\mathsf{PABC}}, disk_{\mathsf{CS}}, dipk)$

Verify $(dnym, pbt)$, abort if $dnym \in \mathcal{L}_{dnym}$.
Extract drh from pbt, and revoke old device credential: $RI' \leftarrow \mathsf{Revoke}_{\mathsf{PABC}}(disk_{\mathsf{PABC}}, RI, drh)$.

$\xleftrightarrow{\mathsf{PABC}-\mathsf{Issue} \text{ under } disk_{\mathsf{PABC}}}$

Run Issuance for committed $dbid$ (contained in sbt, pbt),
carried-over dsk' from $dcred'_{init}$, and fresh drh' chosen by $\mathcal{DI}$.
Issuance token contains device pseudonym $dnym'$ for dsk' and $scope_{reg}$.

Obtain $dcred'$ from $\mathsf{PABC}-\mathsf{Issue}$.

Abort issuance if $dnym' \in \mathcal{L}_{dnym}$.
Set $\mathcal{L}_{dnym} = \mathcal{L}_{dnym} \cup \{dnym, dnym'\}$.

$\xleftrightarrow{\textit{[Rest as in device registration (see Figure 4), generating } \sigma_{\mathcal{DI}}, \sigma_{\mathcal{U}}\textit{]}}$

Store $dcred = (dcred', drh', \sigma_{\mathcal{DI}}, \sigma_{\mathcal{U}})$

Fig. 6. Restore protocol

with insecure memory would therefore require to update *usk* upon re-issuance if scope-exclusive pseudonyms are required.

To prove the above statement we essentially need to show that the commuting signature part, the PABC scheme and the Prove_ϕ prove system can be simulated. As our privacy definitions extend those given by Camenisch et al. [CKL+14], it can be shown that the parts related to the classic PABC system used in our construction can be simulated using their simulator. Simulating the Prove_ϕ system that is used to bind revocation handles in PABC and CS commitments can be done by the zero-knowledge property of the used proof system. Finally we have to show that commuting signatures can be simulated. First note that in all used commuting signatures, the messages and the signatures are given as commitments. Moreover, the verification keys for those commuting signatures are either the publicly known keys of issuers or the users vault public key given also as commitment. Therefore, we can use the results from [Fuc11], that there exists a simulator, that given commitments to messages and verification keys can compute commitments to signatures and valid proofs of correctness.

Unforgeability. Our generic construction satisfies the unforgeability property for all types of devices. This means that there exists no forgery among the issuance and presentation tokens returned by the adversary in the unforgeability game.

First note that credentials in our generic construction are classic PABC system credentials with an additional commuting signature under $\phi(rh) \otimes vupk$. From the unforgeability property of the PABC system, it follows that all presentation or issuance tokens returned by the adversary satisfy that all software credentials are bound to the same user secret key, the pseudonym *nym* is sound for the given $scope_{\mathcal{U}}$, the revealed attribute values are correct, the equality relation E for blinded attributes is satisfied, the committed device binding identifier are the same in all presented credentials (this is also ensured by the E relation) except the ones that are given opening information to ε, and the device pseudonym *dnym* is sound for the given $scope_{\mathcal{U}}$. Moreover, from the binding property of the CS commitments scheme and the soundness property of the proof system Prove_{ϕ} we have that the adversary cannot return presentation or issuance tokens that present credentials bound to different vault public keys. The initial credential and scope specific pseudonyms for static scopes ensure that the adversary cannot create tokens for devices that are restored from a backup token twice and restored or register without activation. Finally, the unforgeability of commuting signatures and the soundness property of Prove_{ϕ} ensure that the adversary cannot create presentation tokens for software credentials with a device the device credential of which was revoked.

3.4 Instantiation

Due to space restrictions, we omit a full instantiation of our generic construction here. Similar to Camenisch et al. [CKL+14], it can be instantiated using Pedersen commitments [Ped91,DF02], CL-signatures [CL02], a variant of the Nakanishi et al. revocation scheme [NFHF09], and the pseudonym scheme used in the IBM identity mixer [IBM10]. The new parts of the construction based on commuting signatures can be instantiated using the scheme proposed by Fuchsbauer [Fuc11], and the proof system Prove_{ϕ} can be obtained using standard techniques.

References

[BCC+09] 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)

[BL13] Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: ACM CCS 2013, pp. 1087–1098. ACM (2013)

[Bra99] Brands, S.: Rethinking public key infrastructure and digital certificates - building in privacy. Ph.D. thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands (1999)

[CH02] Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: Atluri, V. (ed.) ACM CCS 2002, pp. 21–30. ACM (2002)

[Cha81] Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981)

[Cha85] Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985)

[CKL+14] Camenisch, J., Krenn, S., Lehmann, A., Mikkelsen, G.L., Neven, G., Pedersen, M.Ø.: Formal treatment of privacy-enhancing credential systems. Cryptology ePrint Archive, Report 2014/708 (2014). http://eprint.iacr.org/

[CL01] Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)

[CL02] Camenisch, J.L., 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)

[CL04] Camenisch, J.L., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)

[CMZ14] Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic MACs and keyed-verification anonymous credentials. In: ACM CCS 2014, pp. 1205–1216. ACM (2014)

[DF02] Damgård, I.B., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)

[Fuc11] Fuchsbauer, G.: Commuting signatures and verifiable encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 224–245. Springer, Heidelberg (2011)

[GGM14] Garman, C., Green, M., Miers, I.: Decentralized anonymous credentials. In: NDSS 2013. The Internet Society (2014)

[IBM10] IBM Research Zurich - Security Team. Specification of the identity mixer cryptographic library. IBM Technical report RZ 3730 (99740) (2010)

[NFHF09] Nakanishi, T., Fujii, H., Hira, Y., Funabiki, N.: Revocable group signature schemes with constant costs for signing and verifying. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 463–480. Springer, Heidelberg (2009)

[Ped91] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)

[PZ13] Paquin, C., Zaverucha, G.: U-prove Cryptographic specification v1.1 (revision 2). Technical report, Microsoft Corporation, April 2013