Formal Treatment of Privacy-Enhancing Credential Systems*

(Full version)

Jan Camenisch[1], Stephan Krenn[1], Anja Lehmann[1],
Gert Læssøe Mikkelsen[2], Gregory Neven[1], Michael Østergaard Pedersen[3]

[1] IBM Research – Zurich
{jca, skr, anj, nev}@zurich.ibm.com
[2] Alexandra Institute
gert.l.mikkelsen@alexandra.dk
[3] Miracle A/S
mop@miracleas.dk

Abstract. Privacy-enhancing attribute-based credentials (PABCs) are the core ingredient to privacy-friendly authentication systems, allowing users to obtain credentials on attributes and prove possession of these credentials in an unlinkable fashion while revealing only a subset of the attributes. To be useful in practice, however, PABCs typically need additional features such as *i*) revocation, *ii*) pooling prevention by binding credentials to users secret keys, *iii*) pseudonyms as privacy-friendly user public keys, *iv*) proving equality of attributes without revealing their values, *v*) or advanced issuance where attributes can be blindly carried over into new credentials. Provably secure solutions exist for most of these features in isolation, but it is unclear how they can be securely combined into a full-fledged PABC system, or even which properties such a system would aim to fulfill. We provide a formal treatment of PABC systems supporting the mentioned features by defining their syntax and security properties, resulting in the most comprehensive definitional framework for PABCs so far. Unlike previous efforts, our definitions are not targeted at one specific use-case; rather, we try to capture generic properties that can be useful in a variety of scenarios. We believe that our definitions can also be used as a starting point for diverse application-dependent extensions and variations of PABCs. We present and prove secure a generic and modular construction of a PABC system from simpler building blocks, allowing for a "plug-and-play" composition based on different instantiations of the building blocks. Finally, we give secure instantiations for each of the building blocks, including in particular instantiations based on CL- and Brands-signatures which are the core of the Idemix and U-Prove protocols.

Keywords. Privacy, attribute-based credentials, anonymous credentials, provable security, strong authentication.

*This work was funded by the European Commission's FP7 project ABC4Trust. This document is an updated version of [CKM+14].

Contents

Chapter 1

Introduction

Privacy-enhancing attributed-based credentials systems (*privacy ABCs* or PABCs, also known as *anonymous credentials* or *pseudonym systems*) allow for cryptographically strong user authentication while preserving the users' privacy by giving users full control over the information that they reveal. There are three types of parties a PABC system: *users* who obtain credentials from *issuers* certifying a set of attribute values. They can then present these credentials to *verifiers* in an unlinkable manner, while revealing only a subset of the attributes from one or more credentials. The verifier can check the validity of the claimed attribute values using the issuers' public keys.

The importance of privacy and data minimization in authentication systems has been emphasized, e.g., by the European Commission in the European privacy standards [PotEU01, PotEU09] and by the US government in the National Strategy for Trusted Identities in Cyberspace (NSTIC) [Sch10]. With IBM's identity mixer based on CL-signatures [CH02, CL01, CL02, CL04], and Microsoft's U-Prove based on Brands' signatures [Bra99, PZ13] practical solutions for privacy-ABCs exist and are currently deployed in several pilot projects [ABC, IRM, Tea10, Cor11]. Despite this large body of work, however, a unified approach in terms of security definitions for PABC systems is still missing. Rather, existing schemes are either proved secure for very use-case-specific definitions [CL01, BCC+09, GGM13, CMZ13] or do not provide provable security guarantees at all [PZ13, NP13, Zav13].

One reason for the lack of a generic framework might be that schemes tweaked for specific scenarios are often more efficient than generic solutions. However, given the complexity of PABC's, defining, instantiating and re-proving a tailored PABC-variant for every use case is not desirable either. We believe that this is one of the hurdles on the way of PABCs to practice. We take a major step towards such a unified theoretical framework by formally defining the characteristic properties of PABC systems, detached from specific instantiations or use-cases. To ease the design of such schemes, we also define a number of more standard, and easier-to-prove, building blocks, and present a generic construction that securely composes the building blocks into a full-fledged PABC system. Finally, we sketch concrete instantiations of the single components. We believe that our framework comprises the most relevant features of a PABC system, and that it can act as a solid foundation for further extensions.

1.1 Contributions

Our definition of PABC systems comprises the richest feature set of holistic PABC schemes presented thus far. In particular, it supports the following features:

Attributes: A credential may contain any fixed number of attributes. Any subset of these attributes can be revealed at presentation.

Revocation: Issuers can revoke credentials, excluding them from being used in future presentations.

Multi-credential presentations and equality predicates: A single presentation can be based on several credentials. The user can prove equality of attribute values from different credentials without revealing the value.

Key binding: Credentials can be bound to a user's secret key, so that in multi-credential presentations, the verifier is ensured that all credentials belong to the same user.

Advanced and blind issuance: Attributes can be issued blindly, without the issuer learning their value. They can also be carried over from existing credentials, meaning that the issuer is guaranteed that it is equal to an attribute in an existing credential.

Pseudonyms: Users can derive pseudonyms from their secret keys that are unique for a given scope string (e.g., the verifier's identity), but that are unlinkable across scopes. Users can obtain credentials under one pseudonym, and later prove possession of the credential under a different pseudonym.

In terms of security, informally *unforgeability* and either *privacy* or *weak privacy* should be satisfied:

Unforgeability: Users can only perform presentations for attributes they got previously certified by an issuer and if the credential has not yet been revoked.

Privacy: A verifier does not learn more information from a presentation than what is intentionally revealed by the user. Also, a presentation session cannot be linked to the corresponding issuance session. Finally, depending on the required flavor of privacy, multiple presentations of the same (set of) credential(s) should be unlinkable.

Weak privacy: A verifier does not learn more information from a presentation than what is intentionally revealed by the user. Also, a presentation session cannot be linked to the corresponding issuance session. However, in weak privacy no guarantee is given regarding unlinkability, so it might be possible for verifies to link presentations done with the same credential.

Weak privacy is implied by privacy, so schemes fulfilling the privacy property also fulfills the weak privacy property.

While intuitively these properties are rather clear, defining them formally is far from trivial. This is because our aimed system is very complex, e.g., allowing the user to obtain credentials on (*i*) revealed, (*ii*) blindly carried-over and (*iii*) fully blind attributes. Each type comes with different security expectations that must be covered by a single definition. For instance, carry-over attributes presented a challenge in the unforgeability definition: While the issuer never learns these attributes, they must provide essentially the same security guarantees as attributes that the issuer has signed in clear. So how can we later distinguish a forgery from a legitimately obtained credential? For privacy, one challenge was to formalize the exact information users intend to reveal, as they might reveal the same and possibly identifying attributes in different presentation tokens. Also, supporting revocation gives the issuer additional power that needed to be covered in a way that reflects the capabilities of the issuer but without introducing trivial "linking attacks" that would rule out any privacy guarantees.

Consequently, the resulting definitions are rather complex, and directly proving that a concrete scheme satisfies them from scratch can be very challenging and tedious. Also such direct proofs typically give hardly any modularity and thus tend to be hard to verify. We thus also

define a set of building blocks, strongly inspired by existing work, and show how they can be generically composed to build secure PABC-systems. Our construction is efficient in the sense that its complexity is roughly the sum of the complexities of the building blocks. Additionally, it allows for simple changes of individual components, e.g., the underlying revocation scheme can be replaced by any instantiation that meets the described security properties of the revocation building block without affecting any other blocks and without having to reprove the security of the entire scheme.

Finally, we discuss concrete instantiations for all our building blocks based on existing protocols. The "core" building block of privacy-enhancing attribute-based signatures relies on either CL-signatures [CL02] or Brands signatures [Bra93] and the presented revocation scheme is a variant of the scheme of Nakanishi et al. [NFHF09]. The commitment scheme is a direct combination of the commitment scheme Pedersen [Ped91], its generalization to an integer commitment scheme by Damgård and Fujisaki [DF02] and the zero-knowledge proofs of knowledge by Fujisaki et al. [FO97, DF02].

1.2 Use Case

As an example use case of these features, consider government-issued electronic identity cards (eIDs) based on Privacy-ABCs, allowing citizens to selectively disclose certified attributes to online services without being linkable across transactions. Those eID cards can be used in combination with other credentials, e.g., vouchers for a particular service or form a trusted base for further credentials. For instance, a library membership card can be derived from an eID, meaning that it will bound to the same secret key as the identity card or contain attributes which were blindly carried over from the eID. Revocation is obviously a crucial feature for any large-scale eID system, restraining misuse of lost or stolen cards. Scope-exclusive pseudonyms can be used, for example, for anonymous opinion polls: To cast a vote in the poll, a user must prove that he owns a valid, non-revoked eID and provide a corresponding scope-exclusive pseudonym, using a unique identifier of the poll (e.g., its URL) as the scope string. This allows to prevent multiple votes cast by the same user, yet hides individual voting behavior.

Of course, additional measures must be taken to guarantee anonymity on all communication layers, such as using an onion routing protocol like Tor [DMS04] to hide network addresses and prevent traffic analysis.

1.3 Related Work

Our definitions are inspired by the work of Chase et al. [Cha08, BCC+09], who provide formal, so-called property-based definitions of delegatable anonymous credential systems and give a generic construction from P-signatures [BCKL08]. Their work, however, focuses solely on pseudonymous access control with delegation, and lacks additional features such as attributes, revocation, and advanced issuance supporting carried-over attributes.

PABCs were originally envisioned by Chaum [Cha81, Cha85], and subsequently a large number of schemes have been proposed. The currently most prevalent solutions are IBM's Identity Mixer based on CL-signatures [CH02, CL01, CL02, CL04] and Microsoft's U-Prove based on Brands' signatures [Bra99, PZ13]. A first formal definition in the ideal-real world paradigm is by Camenisch and Lysyanskaya [CL01], who covered the basic functionalities without attributes and revocation though. Moreover, their definition is stand-alone, i.e., it does not allow composability as honest parties never output any cryptographic value such as a credential or pseudonym. This restriction makes it infeasible to use their definitions and schemes as building block in a larger system. These drawbacks are shared by the definition of Garman et al. [GGM13].

A recent MAC-based credential scheme [CMZ13] allows for multiple attributes per credential, but requires issuer and verifier to be the same entity. Furthermore, it does not cover pseudonyms,

carry-over attributes, or revocation and provides rather informal security definitions. Another recent work [BL13a] defines an efficient blind signature scheme with attributes and claims that it yields an efficient *linkable* anonymous credential scheme, again without giving formal definitions. The scheme can be seen as a weakened version of our signature building block not guaranteeing unlinkability or extractability of hidden attributes. However, these properties are crucial for our PABC system.

Chapter 2

Preliminaries

In the following we first introduce some notation, and then recap the cryptographic background required for the rest of this document. Readers familiar with the topic may safely skip this section upon first reading, and come back to the definitions whenever necessary.

2.1 Notation

Algorithms and parties are denoted by sans-serif fonts, e.g., A, B. For deterministic algorithms we write $a \leftarrow \mathsf{A}(in)$, if a is the output of A on inputs in. If A is probabilistic, we write $a \xleftarrow{\$} \mathsf{A}(in)$. For an interactive protocol (A, B) let $(out_{\mathsf{A}}, out_{\mathsf{B}}) \leftarrow \langle \mathsf{A}(in_{\mathsf{A}}), \mathsf{B}(in_{\mathsf{B}}) \rangle (in)$ denote that, on private inputs in_{A} to A, in_{B} to B and common input in, A and B obtained outputs out_{A} and out_{B}, respectively.

Let ε be the empty string or empty list, depending on the context. For a set $\mathcal{S}$, $s \xleftarrow{\$} \mathcal{S}$ denotes that s is drawn uniformly at random from $\mathcal{S}$. We write $\Pr[\mathcal{E} : \Omega]$ to denote the probability of event $\mathcal{E}$ over the probability space Ω. For example, $\Pr[f(x) = 1 : x \xleftarrow{\$} \{0,1\}^k]$ is the probability that $f(x) = 1$ if x is drawn uniformly at random in $\{0,1\}^k$. We write vectors as $\vec{x} = (x_i)_{i=1}^k = (x_1, \ldots, x_k)$.

A *negligible* function $\nu : \mathbb{N} \to \mathbb{R}$ vanishes faster than every inverse polynomial, i.e., for every k there exists an n_k such that for all $n > n_k$ we have that $\nu(n) < \frac{1}{n^k}$.

For a positive integer n, $\mathbb{Z}_n^*$ and QR_n denote the multiplicative group and the group of quadratic residues modulo n, respectively.

Let $\pm\{0,1\}^k$ be $[-2^k + 1, 2^k - 1] \cap \mathbb{Z}$, and $\mathbb{P}$ be the set of primes.

Throughout the paper, κ denotes the main security parameter.

2.2 Cryptographic Background

We next recap the notions of zero-knowledge proofs of knowledge, semantically secure encryption, and signatures that are unforgeable under chosen message attacks. For each of the primitives we also recap basic instantiations that we are using for the constructions in the subsequent sections.

2.2.1 Zero-Knowledge Proofs of Knowledge

Informally, a zero-knowledge (ZK) proof of knowledge (PoK) is a two-party protocol between a prover and a verifier which allows the former to convince the latter that it knows some secret piece of information, without revealing anything else than what the claim already reveals. For a formal definition we refer to Bellare and Goldreich [BG92].

Our constructions make use of ZK proofs of knowledge of discrete logarithms and related problems. We use the standard notation introduced by Camenisch and Stadler [CS97] to denote

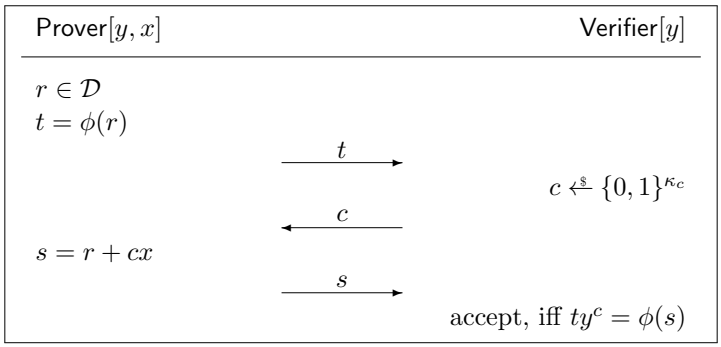

Figure 2.1: Σ-protocol for proving $y = \phi(x)$

such proof goals. For instance, the expression:

$$\mathsf{ZKP}\left[(\alpha, \beta, \gamma) : y_1 = g_1^{\alpha} h_1^{\beta} \ \wedge \ y_2 = g_2^{\alpha} h_2^{\gamma} \ \wedge \ (L < \alpha < R)\right]$$

denotes a zero-knowledge proof of knowledge of values α, β, γ such that the given relations are satisfied. We follow the convention that knowledge of values denoted by Greek letters has to be proved, while all other values are assumed to be public.

There exist efficient techniques for proving knowledge of discrete algorithms in groups of known and hidden order [Sch91, DF02, FO97], for demonstrating linear and multiplicative relations among secret values [Bra97, FO97], for showing that secrets lie in some interval [Bou00, Lip03, CFT98, CM99], and arbitrary Boolean compositions of such statements [CDS94].

All interval claims we will deploy can be realized virtually for free by accepting a small soundness gap, i.e., an honest user has to use witnesses from a slightly smaller interval than what the verifier is ensured.

Σ-Protocols for Group Homomorphisms. Figure 2.1 recaps how a proof goal $\mathsf{ZKP}[(\chi) : y = \phi(\chi)]$ can be realized for a group homomorphism $\phi : \mathcal{G} \to \mathcal{H}$. The computational costs for either party are essentially given by one evaluation of ϕ, and the communication complexity is roughly of the size of the public image y.

If the domain $\mathcal{G}$ of ϕ is finite, we can set $\mathcal{D} = \mathcal{G}$ and the protocol is perfectly zero-knowledge. Otherwise, if $\mathcal{G} = \mathbb{Z}^d$ and $x = (x_1, \ldots, x_d)$, we can set $\mathcal{D} = \prod_{i=1}^{d} \pm\{0,1\}^{\ell_i + \kappa_{zk}}$, where ℓ_i is a publicly known upper bound on the bitlength of x_i, and κ_{zk} is a security parameter controlling the zero-knowledge property of the protocol. In this latter case, fuzzy interval checks can be performed by additionally checking that s lies in some integer interval, cf. [DF02, CKY09].

It can be shown that every Σ-protocol is zero-knowledge against honest-but curious verifiers, and a proof of knowledge with knowledge error $1/2$ if $\kappa_c = 1$. Most proof goals used in practice, and all proof goals used in this document, achieve a knowledge error of $2^{-\kappa_c}$ also for larger κ_c, such that a negligible knowledge error can already be achieved in one run of the protocol. For a detailed discussion on the choice of κ_c we refer to the original literature.

The homomorphisms used in this document are the following:

- For Pedersen commitments (§ 7.2), we have:

$$\phi : \mathbb{Z}^2 \to \mathbb{Z}_n^* : (m, r) \mapsto g^m h^r \,,$$

  and ℓ_1 is the bitlength of the longest message to be committed to and ℓ_2 is the bitlength of the longest possible randomness. The mapping for CL-signatures (cf. § 2.2.3) is similar.

- For Paillier encryptions (§ 2.2.2), we have:

$$\phi : \mathbb{Z}_n \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^2}^* : (m, r) \mapsto g^m r^n \,.$$

- Product homomorphisms of the previous types.

Σ-protocols can be made non-interactive in the random oracle model by substituting the verifier's message c by a call to the random oracle on input t, i.e., $c = \mathsf{H}(t)$. By doing so, the resulting protocol can also be proved secure against arbitrary adversaries.

By setting $c = \mathsf{H}(t, M)$, a signature proof of knowledge for message M can be obtained, cf. Fiat and Shamir [FS87], for which we use the following notation:

$$\mathsf{ZKP}_{\mathsf{FS}}\left[(\alpha, \beta, \gamma) : \ldots\right](M)$$

denotes the Fiat-Shamir transformation [FS87] of the Σ-protocol for the given statement on message M. More formally, it denotes the following:

- First, a Σ-protocol for the given statement is constructed.
- Using the notation from Figure 2.1, instead of letting the verifier choose $c \xleftarrow{\$} \{0,1\}^{\kappa_c}$, the prover computes the challenge as $c \leftarrow \mathsf{H}(y, t, M)$, where H is a random oracle.
- The prover outputs (c, s).
- The verifier accepts, if and only $\mathsf{H}(y, t', M) = c$, where $t' = \phi(s)y^{-c}$.

Non-Interactive Zero-Knowledge and Proofs of Knowledge. In the following we recap the notions of non-interactive zero-knowledge proofs and proofs of knowledge. For a more detailed discussion we refer to, e.g., De Santis et al. [SCO+01].

Informally, a non-interactive zero-knowledge proof lets a prover generate a proof π claiming that some $x \in L$ for an $\mathcal{NP}$-language L. On the one hand, a malicious prover should not be able to cheat the verifier if $x \notin L$, while on the other hand π should not allow the verifier to perform computations it could not have done before. A proof of knowledge ensures the verifier not only that $x \in L$, but also that the prover actually knows a witness $w \in W(x)$ for x, i.e., to convince the verifier the prover needs to know w such that $(x, w) \in R$, where $R = \{(x, w) : x \in L, w \in W(x)\}$.

Definition 2.2.1 (Non-interactive zero-knowledge proof)**.** *A protocol* $(\mathsf{SPGen}, \mathsf{P}, \mathsf{V})$ *is a* non-interactive zero-knowledge proof system *for a language* $L \in \mathcal{NP}$*, if all algorithms run in probabilistic polynomial time, and if there exists a negligible function* ν *such that the following conditions are satisfied for all* κ*:*

Correctness. *For all* $x \in L$ *of length* κ *and all witnesses* w *such that* $(x, w) \in R$ *it holds that:*

$$\Pr[\mathsf{V}(x, \mathsf{P}(x, w, spar), spar) = \mathsf{accept} \;:\; spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)] = 1\,.$$

Soundness. *For all PPT adversaries* A *the following holds:*

$$\Pr[\mathsf{V}(x, \pi, spar) = \mathsf{accept} \;\wedge\; x \notin L \;:\; spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa), (x, \pi) \xleftarrow{\$} \mathsf{A}(spar)] \leq \nu(\kappa)\,.$$

Zero-Knowledge. *There exists a PPT algorithm* $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$*, called* simulator*, such that for all non-uniform PPT adversaries* A *we have that:*

$$\begin{aligned}\big|\,&\Pr[\mathsf{A}^{\mathsf{P}(\cdot,\cdot,spar)}(spar) = 1 : spar \xleftarrow{\$} \mathsf{SPGen}(spar)] - \\ &\Pr[\mathsf{A}^{\mathsf{S}_2'(\cdot,\cdot,spar,\tau)}(spar) = 1 : (spar, \tau) \xleftarrow{\$} \mathsf{S}_1(1^\kappa, spar)]\big| \leq \nu(\kappa)\,,\end{aligned}$$

where $\mathsf{S}_2'(\cdot, \cdot, spar, \tau)$ *removes all secret inputs, forwards the public inputs to* $\mathsf{S}_2(\cdot, spar, \tau)$*, and returns whatever* S_2 *returns.*

Definition 2.2.2 (Non-interactive proof of knowledge)**.** *A protocol* $(\mathsf{SPGen}, \mathsf{P}, \mathsf{V})$ *is a* non-interactive proof of knowledge *for a language* $L \in \mathcal{NP}$*, if all algorithms run in probabilistic polynomial time, and if there exists a negligible function* ν *such that the following conditions are satisfied for all* κ*:*

Correctness. *For all* $x \in L$ *of length* κ *and all witnesses* w *such that* $(x, w) \in R$ *it holds that:*

$$\Pr[\mathsf{V}(x, \mathsf{P}(x, w, spar), spar) = \mathsf{accept} \ : \ spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)] = 1\,.$$

Soundness. *For all PPT adversaries* A *the following holds:*

$$\Pr[\mathsf{V}(x, \pi, spar) = \mathsf{accept} \ \wedge \ x \notin L \ : \ spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa), (x, \pi) \xleftarrow{\$} \mathsf{A}(spar)] \leq \nu(\kappa)\,.$$

Extractability. *There exists a PPT algorithm* $\mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2)$*, called the* knowledge extractor*, such that the following is satisfied:*

- E_1 *outputs parameters and trapdoors, such that the parameters are indistinguishable from correctly computed system parameters:*

$$\{spar \ : \ spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)\} \sim \{spar \ : \ (spar, \tau) \xleftarrow{\$} \mathsf{E}_1(1^\kappa)\}, \text{ and}$$

- *for every efficient adversary* A *there exists a negligible function* ν *such that the following holds:*

$$\begin{aligned}\Pr[\mathsf{V}(x, \pi, spar) = \mathsf{accept} \ \wedge \ (x, w') \notin R \ : \\ (spar, \tau) \xleftarrow{\$} \mathsf{E}_1(1^\kappa), (x, \pi) \xleftarrow{\$} \mathsf{A}(spar), w' \xleftarrow{\$} \mathsf{E}_2(spar, \tau, x, \pi)] \leq \nu(\kappa).\end{aligned}$$

Finally, we say that a protocol is a *non-interactive zero-knowledge proof of knowledge*, if it simultaneously satisfies Definitions 2.2.1 and 2.2.2.

2.2.2 Semantically Secure Encryption

A public key encryption scheme is a triple of algorithms $(\mathsf{EncKGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows: On input 1^κ, $\mathsf{EncKGen}$ outputs a secret key/public key pair (esk, epk). The encryption algorithm Enc takes the public key epk and a message m from some message space $\mathcal{M}$, and outputs a ciphertext c. Finally, the decryption algorithm Dec takes the secret key esk and a ciphertext c and returns a message $m' \in \mathcal{M} \cup \{\perp\}$.

Definition 2.2.3. *An encryption scheme* $(\mathsf{EncKGen}, \mathsf{Enc}, \mathsf{Dec})$ *is called IND-CPA (or* semantically*) secure for a message space* $\mathcal{M}$*, if the following properties hold:*

Correctness. *For every* $m \in \mathcal{M}$ *we have that* $\mathsf{Dec}(esk, \mathsf{Enc}(epk, m)) = m$ *whenever* $(esk, epk) \xleftarrow{\$} \mathsf{EncKGen}(1^\kappa)$.

Semantic security. *For every PPT adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{IND} - \mathsf{CPA}_\mathsf{A} = 1] \leq 1/2 + \nu(\kappa)\,,$$

where the experiment $\mathsf{IND} - \mathsf{CPA}_\mathsf{A}$ *is defined as in Figure 2.2.*

We will sometimes write $\mathsf{Enc}(m; r)$ to make the randomness being used explicit.

The Paillier Encryption Scheme. The Paillier encryption scheme [Pai99] is defined as described in the following.

Experiment $\mathsf{IND}-\mathsf{CPA}_{\mathsf{A}}(1^{\kappa})$**:**
$(esk, epk) \xleftarrow{\$} \mathsf{EncKGen}(1^{\kappa})$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathsf{A}_1(epk)$
$c = \mathsf{Enc}(epk, m_b)$
$b' \xleftarrow{\$} \mathsf{A}_2(\mathsf{st}, c)$
Return 1 if and only if:
$b = b'$.

Figure 2.2: $\mathsf{IND}-\mathsf{CPA}_{\mathsf{A}}(1^{\kappa})$

Key generation. On input 1^{κ}, $\mathsf{EncKGen}$ computes κ_n according to [Blu13] for security level κ. It then chooses two primes p, q of length $\kappa_n/2$ and sets $n = pq$ as well as $\gamma = \mathrm{lcm}(p-1, q-1)$. It further selects $g \xleftarrow{\$} \mathbb{Z}^*_{n^2}$ such that $n | \operatorname{ord}(g)$ and sets $\mu = (L(g^{\gamma} \mod n^2))^{-1} \mod n$, where $L(u) = \frac{u-1}{n}$ and division is done over the integers[1]. It outputs

$$(esk, epk) = ((\gamma, \mu, n), (n, g)) .$$

Encryption. Encryption of a message $m \in \mathbb{Z}_n$ is done as follows:

$$\mathsf{Enc}(epk, m) = g^m r^n \bmod n^2 \quad \text{for} \quad r \xleftarrow{\$} \mathbb{Z}^*_n .$$

Decryption. Decryption of a ciphertext c is defined as:

$$\mathsf{Dec}(esk, c) = L(c^{\gamma} \bmod n^2)\mu \bmod n .$$

It can be shown that the Paillier encryption scheme is IND-CPA secure under the quadratic residuosity assumption defined next:

Definition 2.2.4 (Quadratic Residuosity Assumption)**.** *Let n be a random safe RSA modulus, and $\mathcal{X} \supseteq QR_n$ be the subgroup of elements from $\mathbb{Z}^*_n$ with Jacobi symbol equal to 1. The* quadratic residuosity assumption *then states that for $x \xleftarrow{\$} \mathcal{X}$ it is computationally infeasible to decide whether or not $x \in QR_n$.*

2.2.3 Digital Signatures

A digital signature scheme is a triple of algorithms $(\mathsf{SignKGen}, \mathsf{Sign}, \mathsf{SignVf})$ defined as follows: On input 1^{κ}, $\mathsf{SignKGen}$ outputs a secret key/public key pair (ssk, spk). The signing algorithm Sign takes the secret key ssk and a message m from some message space $\mathcal{M}$, and outputs a signature sig. Finally, the verification algorithm SignVf takes the public key spk, a message m, and a signature sig and returns a accept or reject.

Definition 2.2.5. *A signature scheme* $(\mathsf{SignKGen}, \mathsf{Sign}, \mathsf{SignVf})$ *is called* existentially *(also* weakly*)* UF-CMA *(unforgeable under adaptive chosen-message attacks) secure for a message space $\mathcal{M}$, if the following properties hold:*

Correctness. *For every $m \in \mathcal{M}$ we have that* $\mathsf{SignVf}(spk, m, \mathsf{Sign}(ssk, m)) = \mathsf{accept}$ *whenever* $(ssk, spk) \xleftarrow{\$} \mathsf{SignKGen}(1^{\kappa})$.

Existential unforgeability under chosen-message attacks. *For every PPT adversary* A *there exists a negligible function ν such that*

$$\Pr[\mathsf{UF}-\mathsf{CMA}_{\mathsf{A}} = 1] \leq \nu(\kappa) ,$$

[1] This is possible since $g^{\gamma} \equiv 1 \bmod n$

Experiment $\mathsf{UF}-\mathsf{CMA}_{\mathsf{A}}(1^{\kappa})$**:**
$(ssk, spk) \xleftarrow{\$} \mathsf{SignKGen}(1^{\kappa})$
$(m^*, sig^*) \xleftarrow{\$} \mathsf{A}^{\mathcal{O}(ssk,\cdot)}(spk)$
Return 1 if and only if:
$\mathsf{SignVf}(spk, m^*, sig^*) = \mathsf{accept}$, and
$m^* \notin \mathcal{L}$.

Figure 2.3: $\mathsf{UF}-\mathsf{CMA}_{\mathsf{A}}(1^{\kappa})$

where the experiment $\mathsf{UF}-\mathsf{CMA}_{\mathsf{A}}$ *is defined as in Figure 2.3. There, the signing oracle* $\mathcal{O}(ssk,\cdot)$ *takes messages* m *from* $\mathcal{M}$ *as inputs, adds* m *to the initially empty list* $\mathcal{L}$*, and returns* $sig \xleftarrow{\$} \mathsf{Sign}(ssk, m)$ *to the adversary.*

Camenisch-Lysyanskaya Signatures. We next recap the Camenisch-Lysyanskaya (CL) signature scheme [CL02], for which we need the following computational assumption [BP97,FO97]:

Definition 2.2.6 (Strong RSA Assumption)**.** *For a random strong RSA modulus* n*, i.e.,* $n = pq$*, where* $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ *are all prime, and a random* $g \xleftarrow{\$} \mathbb{Z}_n^*$*, it is computationally hard to find* $e > 1$ *and* $h \in \mathbb{Z}_n^*$ *such that* $g = h^e \mod n$*.*

It can then be shown that under this assumption the following signature scheme for a block of L messages is secure against adaptive chosen message attacks [CL02]:

Key generation. On input of security parameter κ_n, choose a strong RSA modulus for which p and q have length $\kappa_n/2$, and choose $R_1, \ldots, R_L, Z, S \xleftarrow{\$} QR_n$. The public key is given by $spk = (R_1, \ldots, R_L, Z, S, n)$, and the secret key is given by $ssk = p$.

Signing. To sign a tuple $(m_1, \ldots, m_L) \in \pm\{0,1\}^{\ell_m}$ of messages, choose a random prime e of length $\ell_e > \ell_m + 2$, as well as $v \in \{0,1\}^{\kappa_n+\ell_m+\kappa_v}$, where κ_v is a security parameter. Compute:

$$A = \left(\frac{Z}{S^v \prod_{i=1}^{L} R_i^{m_i}}\right)^{1/e} .$$

The signature is given by (e, A, v).

Verification. Given messages $m_1, \ldots, m_L$ and a signature (e, A, v), check that $m_i \in \pm\{0,1\}^{\ell_m}$, $2^{\ell_e-1} < e < 2^{\ell_e}$, and

$$Z = A^e S^v \prod_{i=1}^{L} R_i^{m_i} .$$

One feature of the CL-signature scheme is that it allows one to prove possession of a signature, without revealing any other information about it. In particular, none of the values e, A, v needs to be made public, and thus such proofs can be made unlinkable. This is achieved through re-randomization: for a valid signature (e, A, v), the triple $(e, A' = AS^{-r}, v' = v + er)$ can easily be seen to also be a valid signature. Now, as n is a strong RSA modulus, if $A \in \langle S \rangle$ (which is the case with overwhelming probability), and if r was chosen uniformly from $\{0,1\}^{\kappa_n+1}$, we have that A' is statistically close to uniform over $\mathbb{Z}_n^*$. Thus, to prove possession of a valid signature, the user can always re-randomize (e, A, v), reveal A', and run the following protocol:

$$\mathsf{ZKP}\left[(\varepsilon, \nu', \mu_1, \ldots, \mu_L) \ : \ Z = \pm A'^{\varepsilon} S^{\nu'} \prod_{i=1}^{L} R_i^{\mu_i} \ \wedge \ \mu_i \in \pm\{0,1\}^{\ell_m} \ \forall i \ \wedge \ 2^{\ell_e-1} < \varepsilon < 2^{\ell_e}\right].$$

Note that this proof only ensures that one knows a valid signature with respect to Z or $-Z$. While this is a technicality that arises for all known efficient zero-knowledge proofs of discrete logarithms in RSA groups, it is not a problem in our setting. We refer to the literature for a profound discussion, e.g., [CKY09, DF02]. We note that this problem could be avoided by alternatively using the signed quadratic residues instead of QR_n [HK09].

The required interval checks can be realized for free, if in the description of the signature scheme the domain of e and the message space are slightly decreased. Instead of choosing e from $\{0,1\}^{\ell_e}$, we can choose it from $[2^{\ell_e-1} - 2^{\ell'_e} + 1, 2^{\ell_e-1} + 2^{\ell'_e} - 1]$, where $\ell'_e < \ell_e - \kappa_c - 3$ and κ_c is a security parameter controlling the knowledge error of the protocol. Similarly, all messages m_i must be from $\pm\{0,1\}^{\ell_m-\kappa_c-2}$, cf. [CG08].

UProve. Here we describe the signature scheme for blocks of L messages upon which U-Prove is based [Paq13]. This is a modified version of the blind signature scheme by Brands [Bra93].

So far no formal security proof of the following scheme is known, and Baldimtsi and Lysyanskaya have shown that for the unmodified Brands blind signature scheme [Bra93] underlying this construction all known approaches for proving security in the random oracle model will fail under essentially any assumption [BL13b]. However, it is easy to see that for the following scheme the discrete logarithm assumption is necessary: if an adversary could compute discrete logarithms, he could find different $(m_i)_{i=1}^L \neq (m'_i)_{i=1}^L$ mapping to the same t value in the protocol, and therefore a signature for $(m_i)_{i=1}^L$ would also be a valid signature for $(m'_i)_{i=1}^L$.

Key generation. On input of security parameter κ the algorithm behaves as follows:

- It first computes κ_p for security level κ, and
- then chooses a prime p of length κ_p and a prime q such that $q|p-1$.
- The algorithm picks a random generator g of the unique subgroup of order q in $\mathbb{Z}_p^*$, and
- then chooses random values $y_i \in_R \mathbb{Z}_q$ and sets $g_i = g^{y_i}$ for $i = 0, \ldots, L$.

The algorithm outputs public key $spk = (g, p, q, g_0, g_1, \ldots, g_L)$ and secret key $ssk = y_0$.

Signing. To obtain a signature on a tuple $(m_1, \ldots, m_L) \in \mathbb{Z}_q^L$ of messages, the user runs the protocol shown in Figure 2.4 with the signer.

Verification. Given messages $m_1, \ldots, m_L$ and a signature (s, h, z', c', r'), algorithm SignVf checks whether the following two equations are satisfied:

$$h \stackrel{?}{=} (g_0 \prod_{i=1}^{L} g_i^{m_i})^s$$
$$c' \stackrel{?}{=} \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'} (z')^{-c'})$$

Informal comparison of CL- and UProve-signatures. Note that the UProve-signature scheme works substantially different than CL-signatures. In the CL-scheme, the signer learns the signature he sends to the user. However, the user is able to re-randomize the signature, and then prove possession of such a re-randomization. In this way, the values revealed to the verifier are statistically independent of the actual signature obtained from the signer, and thus presentations become unlinkable from signing sessions, and also multiple presentations of the same signature cannot be linked to each other.

For UProve-signatures, the user is not able to perform such a re-randomization. However, the signer does not learn the actual signature given to the user. In this way the user can reveal

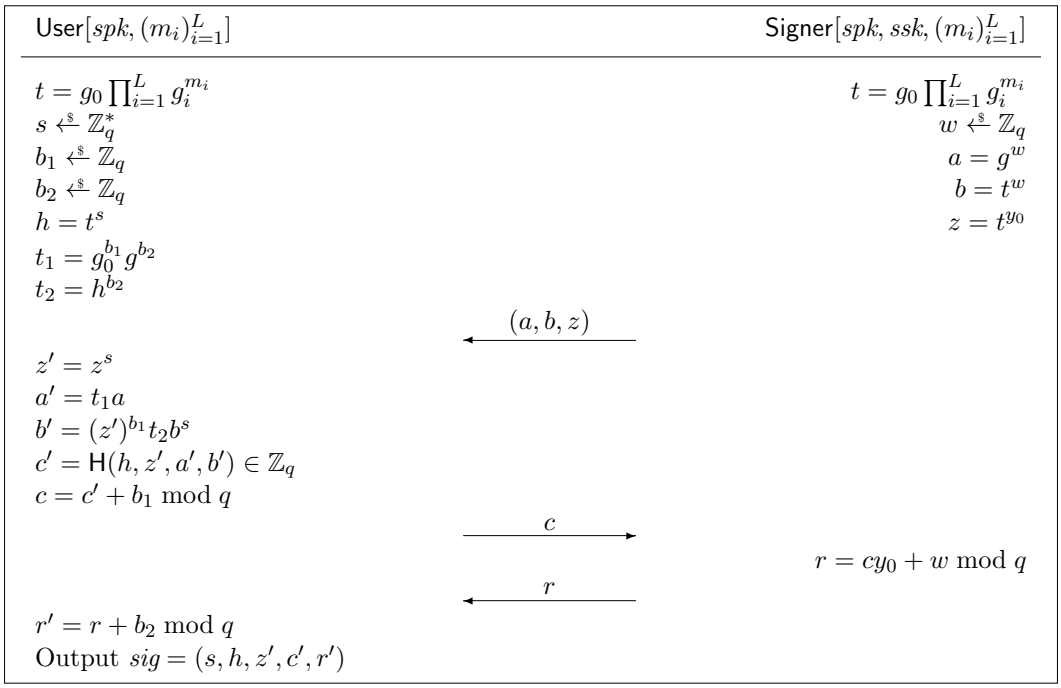

Figure 2.4: Issuance of a signature on messages $m_1, \ldots, m_L$.

the signature itself as a prove of possession of a valid signature, without presentation and signing sessions becoming linkable. On the downside, any two presentations of the same signature are trivially linkable.

For a formal discussion of the resulting privacy properties we refer to § 8.2 and § 8.3, respectively.

Chapter 3

Privacy ABC Systems

A privacy-enhancing attribute-based credential (PABC) system for an attribute space $\mathcal{AS} \subseteq \pm\{0,1\}^{\ell}$ is a set of algorithms SPGen, UKGen, IKGen, Present, Verify, ITGen, ITVf, and Revoke and a protocol $\langle \mathcal{U}.\mathsf{Issue}, \mathcal{I}.\mathsf{Issue} \rangle$. Parties are grouped into issuers, users, and verifiers. The publicly available system parameters are generated using SPGen by a trusted party (this algorithm might be implemented using multiparty techniques in practice). There are multiple issuers, each of which can issue and revoke credentials that certify a list of attribute values under his issuer public key. Users hold secret keys that can be used to derive *pseudonyms* that are unique for a given scope string, but are unlinkable across scopes. Using Present, users can create non-interactive *presentation tokens* from their credentials in which they can reveal any subset of attributes from any subset of their credentials, or prove that certain attribute values are equal without revealing them. Presentation tokens can be publicly verified using Verify, on input the token and the public keys of the issuers of the contained credentials. To obtain a credential, a user generates an *issuance token* defining the attribute values of the credential to be issued using ITGen. Once the issuer has received an issuance token (and verified it using ITVf), the user and the issuer run $\langle \mathcal{U}.\mathsf{Issue}, \mathcal{I}.\mathsf{Issue} \rangle$, at the end of which the user obtains a credential. Issuance can be combined with the presentation of existing credentials to obtain a partially blind form of issuance, where some of the attribute values are known to the issuer, others are hidden but proved to be equal to attributes in credentials the user already owns, and even others are chosen by the user and unknown to the issuer. Hence, issuance tokens can be seen as an extension of presentation tokens. Credentials can optionally be bound to a user's secret key, meaning that knowledge of this key is required to prove possession of the credential. Now, if a pseudonym or multiple key-bound credentials are used in a presentation token, then all credentials and the pseudonym must be bound to the same key. Finally, an issuer can revoke a credential using the Revoke algorithm. This algorithm outputs some public revocation information RI that is published and should be used as input to the verification algorithm of presentation and issuance tokens.

We assume that issuers and users agree on the various parameters for the issuance token including a revocation handle rh and the attributes of the new credential (upon which the user has generated the issuance token pit) in a step preceding issuance, and that issuers verify the validity of these tokens before engaging in this protocol. There are no requirements on how revocation handles are chosen. However, in practice the handle should be different for each credential an issuer issues.

3.1 Syntax

Before formalizing the security properties, we introduce the syntax of PABCs.

System parameter generation. The system parameters of a PABC-system are generated as $spar \xleftarrow{\$} \mathsf{SPGen}(1^{\kappa})$. For simplicity we assume that the system parameters in particular

contain an integer L specifying the maximum number of attributes that can be certified by one credential, as well as a description of the attribute space $\mathcal{AS}$. For the rest of this document, we assume that all honest parties only accept attributes from $\mathcal{AS}$ and abort otherwise.

These parameters are input to all algorithms presented in the following. However, for notational convenience, we will sometimes not make this explicit.

User key generation. Each user generates a secret key as $usk \xleftarrow{\$} \mathsf{UKGen}(spar)$.

Issuer key generation. Each issuer generates a public/private issuer key pair and some initial revocation information as $(ipk, isk, RI) \xleftarrow{\$} \mathsf{IKGen}(spar)$. We assume that RI implicitly also defines the set of all supported revocation handles for this issuer, and that honest parties only accept revocation handles from this set and abort otherwise.

Presentation. A user generates a pseudonym nym and presentation token pt as

$$(nym, pt) \xleftarrow{\$} \mathsf{Present}\Big(usk, scope, \big(ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i\big)_{i=1}^{k}, E, M\Big), \text{ where}$$

- usk is the user's secret key, which can be ε if $scope = \varepsilon$ and none of the credentials $(cred_i)_{i=1}^{k}$ is bound to a user secret key;
- $scope$ is the scope of the generated pseudonym nym, where $scope = \varepsilon$ if no pseudonym is to be generated (in which case $nym = \varepsilon$);
- $(cred_i)_{i=1}^{k}$ are k credentials owned by the user that are involved in this presentation;
- ipk_i and RI_i are the public key and current revocation information of the issuer of $cred_i$;
- $(a_{i,j})_{j=1}^{n_i}$ is the list of attribute values certified in $cred_i$;
- $R_i \subseteq \{1, \ldots, n_i\}$ is the set of attribute indices for which the value is revealed;
- E is an equivalence relation on $\{(i,j) : 1 \leq i \leq k \wedge 1 \leq j \leq n_i\}$, where $((i,j),(i',j')) \in E$ means that the presentation token will prove that $a_{i,j} = a_{i',j'}$ without revealing the actual attribute values. That is, E enables one to prove equality predicates;
- $M \in \{0,1\}^*$ is a message to which the presentation token is to be bound. In practical applications this might, e.g., be a nonce chosen by the verifier to prevent replay attacks, where a verifier uses a valid presentation token to impersonate a user towards another verifier.

If $k = 0$ and $scope \neq \varepsilon$, only a pseudonym nym is generated while $pt = \varepsilon$.

Presentation verification. A verifier can check the validity of a pseudonym nym and a presentation token pt by executing

$$\mathsf{accept/reject} \leftarrow \mathsf{Verify}\Big(nym, pt, scope, \big(ipk_i, RI_i, (a_{i,j})_{j\in R_i}\big)_{i=1}^{k}, E, M\Big),$$

where the inputs are as for presentation. For notational convenience, we assume here and in the following, that a term like $(a_{i,j})_{j\in R_i}$ implicitly also describes the set R_i.

Issuance token generation. Before issuing a credential, a user needs to create 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). Similarly to a presentation token, an issuance token is generated as

$$(nym, pit, sit) \xleftarrow{\$} \mathsf{ITGen}\Big(usk, scope, rh, \big(ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i\big)_{i=1}^{k+1}, E, M\Big),$$

where most of the inputs and outputs are as before, but

- *pit* and *sit* are the public and secret parts of the issuance token,
- $cred_{k+1} = \varepsilon$ is a placeholder for the new credential to be issued,
- *rh* is the revocation handle for $cred_{k+1}$ (maybe chosen by the issuer before),
- ipk_{k+1} and RI_{k+1} are the public key and current revocation information of the issuer of the new credential,
- $(a_{k+1,j})_{j \in R_{k+1}}$ 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, and
- $((k+1, j), (i', j')) \in E$ means that the jth attribute of the new credential will have the same value as the j'th attribute of the i'th credential.

Issuance token verification. The issuer verifies an issuance token as follows:

$$\mathsf{accept/reject} \xleftarrow{\$} \mathsf{ITVf}\Big(nym, pit, scope, rh, \big(ipk_i, RI_i, (a_{i,j})_{j \in R_i}\big)_{i=1}^{k+1}, E, M\Big)$$

where for $j = 1, \ldots, k$ all inputs and outputs are as for Verify, but for $k+1$ they are for the new credential to be issued based on *pit*.

Issuance. Issuance of credentials is an interactive protocol:

$$(cred, RI') \xleftarrow{\$} \langle (\mathcal{U}.\mathsf{Issue}(sit); \mathcal{I}.\mathsf{Issue}(isk, pit, RI)\rangle\,,$$

executing between a user and an issuer, where the inputs are defined as before, and *pit* has been verified by the issuer before. The user obtains a credential as an output, while the issuer receives an updated revocation information RI'.

Revocation. To revoke a credential with revocation handle *rh*, the issuer runs:

$$RI' \xleftarrow{\$} \mathsf{Revoke}(isk, RI, rh)$$

to generate the new revocation information RI' based on the issuer's secret key, the current revocation information, and the revocation handle to be revoked.

3.2 Oracles for Our Security Definitions

The definitions of the security properties of a PABC system require a number of oracles modeling the honest parties. Since some of the oracles are used in multiple definitions, we present them all in one place. The oracles are initialized with a set of honestly generated keys of n_{I} honest issuers $(ipk_1^*, isk_1^*, RI_1^*), \ldots, (ipk_{n_{\mathrm{I}}}^*, isk_{n_{\mathrm{I}}}^*, RI_{n_{\mathrm{I}}}^*)$ and n_{U} users with keys $usk_1^*, \ldots, usk_{n_{\mathrm{U}}}^*$, respectively. Let $IK^* = \{ipk_1^*, \ldots, ipk_{n_{\mathrm{I}}}^*\}$. The oracles also maintain initially empty sets $\mathcal{C}$, $\mathcal{HP}$, $\mathcal{IT}$, $\mathcal{IRH}$, $\mathcal{RRH}$, $\mathcal{RI}$. Here, $\mathcal{C}$ contains all credentials that honest users have obtained as instructed by the adversary, while $\mathcal{HP}$ contains all presentation tokens generated by honest users. All public issuance tokens that the adversary used in successful issuance protocols with honest issuers are stored in $\mathcal{IT}$. The set $\mathcal{IRH}$ contains all issued revocation handles, i.e., the revocation handles of credentials issued by honest issuers, while $\mathcal{RRH}$ contains the revoked handles per issuer. Finally, $\mathcal{RI}$ contains the history of the valid revocation information of honest issuers at any point in time. Time is kept per issuer I through a counter $epoch_I^*$ that is initially zero and increased at each issuance and revocation by issuer I.

Honest Issuer Oracle $\mathcal{O}^{\mathtt{issuer}}$. The issuer oracle allows the adversary to obtain and revoke credentials from honest issuers. It provides the following interfaces:

- On input $(\texttt{issue}, nym, pit, scope, rh, (ipk_i, RI_i, (a_{i,j})_{j\in R_i})_{i=1}^{k+1}, E, M)$ the oracle checks that ITVf accepts the issuance token pit and that the revocation information of all honest issuers is authentic, i.e., that a tuple $(ipk_i, RI_i, \cdot) \in \mathcal{RI}$ exists for all honest issuers ipk_i. Further, it verifies that ipk_{k+1} is the public key of an honest issuer ipk_I^* with corresponding secret key isk_I^* and current revocation information $RI_I^* = RI_{k+1}$. If one of the checks fails, the oracle outputs $\perp$, otherwise it proceeds as follows.

  The oracle runs $\mathcal{I}.\mathsf{Issue}(isk_{k+1}, pit, RI_I^*)$ in interaction with A until the protocol outputs RI_{k+1}. It returns RI_{k+1} to A, sets $RI_I^* \leftarrow RI_{k+1}$, increases $epoch_I^*$, adds $(ipk_I^*, RI', epoch_I^*)$ to $\mathcal{RI}$. It also adds $\big(nym, pit, scope, (ipk_i, RI_i, (a_{i,j})_{j\in R_i})_{i=1}^{k+1}, E, M\big)$ to $\mathcal{IT}$ and adds (ipk_I^*, rh) to the set $\mathcal{IRH}$.

- On input $(\texttt{revoke}, I, rh)$ the oracle checks that the revocation handle rh has been the input of a successful issuance protocol with an honest issuer with key ipk_I^*, or returns $\perp$ otherwise. The oracle runs $RI_I^* \xleftarrow{\$} \mathsf{Revoke}(isk_I^*, RI_I^*, rh)$, increases $epoch_I^*$, adds $(ipk_I^*, rh, epoch_I^*)$ to $\mathcal{RRH}$, adds $(ipk_I^*, RI', epoch_I^*)$ to $\mathcal{RI}$, and returns RI_I^* to the adversary.

Honest User Oracle $\mathcal{O}^{\texttt{user}}$. The user oracle gives the adversary access to honest users, which he can trigger to obtain credentials on inputs of his choice and request presentation tokens on inputs of his choice. The adversary does not get the actual credentials, but is only given a unique credential identifier cid by which he can refer to the credential. It provides the following interfaces:

- On input $(\texttt{present}, U, scope, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k}, E, M)$ the oracle checks if $U \in \{1, \ldots, n_{\mathrm{U}}\}$ and if, for all $i = 1, \ldots, k$, a tuple $(U, cid_i, cred_i, (a_{i,j})_{j=1}^{n_i}) \in \mathcal{C}$ exists. Further, for all credentials from honest issuers the oracle verifies if the provided revocation information is authentic, i.e., if for all honest issuer public keys $ipk_i = ipk_I^*$ there exists a tuple $(ipk_i, RI_i) \in \mathcal{RI}$. If any of the checks fails, the oracle returns $\perp$, otherwise it computes $(nym, pt) \xleftarrow{\$} \mathsf{Present}\big(usk_U^*, scope, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)$ and adds the tuple $\big(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j\in R_i})_{i=1}^{k}, E, M\big)$ to $\mathcal{HP}$. Finally, it returns (nym, pt) to A.

- On input $(\texttt{obtain}, U, scope, rh, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k+1}, E, M, (a_{k+1,j})_{j=1}^{n_{k+1}})$ the oracle checks that $U \in \{1, \ldots, n_{\mathrm{U}}\}$ and that, for all $i = 1, \ldots, k$, a tuple $(U, cid_i, cred_i, (a_{i,j})_{j=1}^{n_i}) \in \mathcal{C}$ exists. It further checks that the revocation information of honest issuers is authentic, i.e., that a tuple $(ipk_i, RI_i, \cdot) \in \mathcal{RI}$ exists for all honest issuers $ipk_i \in IK^*$. The oracle computes the issuance token $(nym, pit, sit) \xleftarrow{\$} \mathsf{ITGen}\big(usk_U^*, scope, rh, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k+1}, E, M\big)$. If $ipk_{k+1} \notin IK^*$, the oracle sends (nym, pit) to the adversary and runs $\mathcal{U}.\mathsf{Issue}(sit)$ in interaction with the adversary until it returns a credential $cred$ and stores $(U, cid_{k+1}, cred, (a_{k+1,j})_{j=1}^{n_{k+1}})$ in $\mathcal{C}$. If $ipk_{k+1} = ipk_I^* \in IK^*$, the oracle runs $\mathcal{U}.\mathsf{Issue}(sit)$ internally against $\mathcal{I}.\mathsf{Issue}(isk_I^*, pit)$ until they output a credential $cred$ and revocation information RI', respectively. The oracle adds $(U, cid_{k+1}, cred, (a_{k+1,j})_{j=1}^{n_{k+1}})$ in $\mathcal{C}$ and adds (ipk_I^*, rh) to $\mathcal{IRH}$. It further increases $epoch_I^*$, sets $RI_I^* \leftarrow RI'$, and adds $(ipk_I^*, RI_I^*, epoch_I^*)$ to $\mathcal{RI}$. Finally, the oracle chooses a fresh and unique credential identifier cid_{k+1} for the new credential and outputs it to A (note that A's choice of cid_{k+1} is ignored).

3.3 Security Definitions for PABC-Systems

In the following we now describe the security properties that need to be satisfied by a PABC-system. We refer to the introduction for an informal discussion.

Experiment $\mathsf{Forge}_{\mathsf{A}}(1^\kappa, n_\mathrm{I}, n_\mathrm{U})$**:**
$spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)$
$(ipk_I^*, isk_I^*, RI_I^*) \xleftarrow{\$} \mathsf{IKGen}(spar)$ for $I = 1, \ldots, n_\mathrm{I}$
$usk_U^* \xleftarrow{\$} \mathsf{UKGen}(spar)$ for $U = 1, \ldots, n_\mathrm{U}$
$\mathcal{FT} \xleftarrow{\$} \mathsf{A}^{\mathcal{O}^{\mathtt{issuer}}, \mathcal{O}^{\mathtt{user}}}(spar, (ipk_I^*, RI_I^*)_{I=1}^{n_\mathrm{I}}, n_\mathrm{U})$
Return 1 if and only if:
$\mathcal{FT}$, $\mathcal{IT}$, $\mathcal{HP}$, $\mathcal{IRH}$, $\mathcal{RRH}$, and $\mathcal{RI}$ are not consistent.

Figure 3.1: $\mathsf{Forge}_{\mathsf{A}}(1^\kappa, n_\mathrm{I}, n_\mathrm{U})$

3.3.1 Correctness

We omit a formal definition here, as the correctness requirements are what one would expect, i.e., whenever all parties are honest and run all algorithms correctly, none of the algorithms aborts or outputs reject. That is, (*i*) if all inputs are correct, Issue always outputs a valid credential, (*ii*) holding valid credentials satisfying the specified relations allows a user to generate valid presentation- and issuance tokens, and (*iii*) issuers are able to revoke any attribute of their choice.

3.3.2 Pseudonym Collision-Resistance

We require that, given fresh system parameters $spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)$ as input, no PPT adversary A can come up with two user secret keys $usk_0 \neq usk_1$ and a scope $scope$ such that $\mathsf{NymGen}(spar, usk_0, scope) = \mathsf{NymGen}(spar, usk_1, scope)$.

We further require that a pseudonym is a deterministic function of the system parameters, user secret and the scope, i.e., that there exists a function NymGen such that for all usk, $scope$, rh, C, C', E, E' and M, M'

$$\begin{aligned}\Pr[nym = nym_i = nym_p : & spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa), nym \leftarrow \mathsf{NymGen}(spar, usk, scope),\\ & (nym_i, pit, sit) \xleftarrow{\$} \mathsf{ITGen}(usk, scope, rh, C', E', M'),\\ & (nym_p, pt) \xleftarrow{\$} \mathsf{Present}(usk, scope, C, E, M)] = 1\end{aligned}$$

3.3.3 Unforgeability

We define unforgeability as a game between the adversary and the oracles from § 3.2, from which he can request credentials from honest issuers, or trigger honest users to receive credentials or make presentation tokens. After having interacted with all these oracles, the adversary outputs a number of presentation tokens and pseudonyms, i.e., a set $\mathcal{FT}$ of tuples $(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k}), E, M)$ as defined in the syntax of the Verify algorithm. The adversary wins the game if at least one of the presentation tokens is a forgery or if at least one of the issuance tokens submitted to the honest issuer oracle was a forgery.

A forgery is informally defined as an issuance or presentation token for which the corresponding credentials were not issued to the adversary or are not supposed to be valid w.r.t. the revocation information stated in the token. Now, as the issuer does not see all attributes nor the user secret key of issued credentials, it is often not clear whether or not a given issued credential is one of the credentials corresponding to a token. However, if we assume that we knew all hidden values for each credential issued (including the user secret key), then we can efficiently test whether or not a given issuance or presentation token is a forgery. Thus, if there is an assignment for all the hidden values of the issued credentials such that all the issuance and presentation tokens presented by the adversary correspond to valid credentials, then there is no forgery among the tokens. Or, in other words, if there is no such assignment, then the adversary has produced a forgery and wins the game. Regarding the validity of credentials,

the adversary also wins if he outputs a valid token for a credential that was already revoked. Thereby, the revocation information must not necessarily be the latest information an honest issuer has published, but can also be a previous version.

Definition 3.3.1 (Unforgeability)**.** *We say that a* PABC*-scheme satisfies* unforgeability*, if for every PPT adversary* A *and all* $n_U, n_I \in \mathbb{N}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{Forge}_{\mathsf{A}} = 1] \leq \nu(\kappa)\,,$$

where the experiment is described in Figure 3.1, and the oracles $\mathcal{O}^{\mathtt{issuer}}$ *and* $\mathcal{O}^{\mathtt{user}}$ *are as defined in § 3.2.*

We now define what *consistency* of the sets $\mathcal{FT}$, $\mathcal{IT}$, $\mathcal{HP}$, $\mathcal{IRH}$, $\mathcal{RRH}$, and $\mathcal{RI}$ means. First, the set $\mathcal{FT}$ must only contain "fresh" and valid presentation tokens, meaning that for each tuple $(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j\in R_i})_{i=1}^k), E, M)$ in $\mathcal{FT}$ must pass Verify and that for all $i = 1, \ldots, k$ where $ipk_i \in IK^*$ there exists a tuple $(ipk_i, RI_i, \cdot) \in \mathcal{RI}$. If any tuple in $\mathcal{FT}$ does not satisfy these conditions, then the adversary loses the game. Let $USK \subset \{0,1\}^*$ be a hypothetical set containing the user secret keys that the adversary may have used throughout the game, and let *CRED* be a hypothetical set containing the credentials from honest issuers that the adversary may have collected during the game. In the latter set, we write $(usk, ipk_I^*, rh, (\alpha_1, \ldots, \alpha_n)) \in CRED$ if the adversary obtained a credential from issuer ipk_I^* with revocation handle rh and attribute values $(\alpha_1, \ldots, \alpha_n)$ bound to user secret usk. Now, we consider the sets $\mathcal{FT}$, $\mathcal{IT}$, $\mathcal{HP}$, $\mathcal{IRH}$, $\mathcal{RRH}$ and $\mathcal{RI}$ to be *consistent* if there exist sets *USK* and *CRED* such that the following conditions hold:

1. *Each credential is the result of a successful issuance.* For all revocation handles rh and honest issuer public keys ipk_I^*, the number of tuples $(\cdot, ipk_I^*, rh, \cdot) \in CRED$ is at most the number of tuples $(ipk_I^*, rh) \in \mathcal{IRH}$.

2. *All presentation or issuance tokens correspond to legitimately obtained unrevoked credentials.* For every tuple $\big(nym, pt, scope, (ipk_i, RI_i, epoch_i, (a_{i,j})_{j\in R_i})_{i=1}^k, E, M\big) \in \mathcal{FT} \cup \mathcal{IT}$ there exists a $usk \in USK$ and a set of credentials $\{cred_i = (usk_i, ipk_i, rh_i, (\alpha_{i,j})_{j=1}^{n_i}) : ipk_i \in IK^*\} \subseteq CRED$ such that:

   (a) $usk_i \in \{usk, \varepsilon\}$ (all key-bound credentials are bound to the same key),

   (b) $nym = scope = \varepsilon$ or $nym = \mathsf{NymGen}(spar, usk, scope)$ (if there is a pseudonym, then it is the pseudonym for usk and $scope$),

   (c) $\alpha_{i,j} = a_{i,j}$ for all $j \in R_i$ (the revealed attribute values are correct),

   (d) $\alpha_{i,j} = \alpha_{i',j'}$ for $((i,j),(i',j')) \in E$ with $ipk_{i'} \in IK^*$ (the attributes satisfy the equality relations to other credentials from honest issuers), and

   (e) there exists a tuple $(ipk_i, RI_i, epoch_i) \in \mathcal{RI}$ such that there exists no tuple $(ipk_i, rh_i, epoch_i') \in \mathcal{RRH}$ with $epoch_i' \leq epoch_i$ (the credentials were not revoked in the epoch where they were presented).

Thus, the adversary wins the game, if there do not exist sets *USK* and *CRED* that satisfy all the information captured in $\mathcal{FT}$, $\mathcal{IT}$, $\mathcal{HP}$, $\mathcal{IRH}$, $\mathcal{RRH}$, and $\mathcal{RI}$.

We had to make a number of design decisions for our definitions, which we want to explain and motivate in the following. First, we do not require strong unforgeability, i.e., we do not consider a token a forgery if it contains the identical elements as a *pt* or *pit* generated by an honest user. However, we believe that this is not a restriction, as tokens are bound to messages M that typically uniquely identify the current session, and thus an adversary will not be able to benefit from a weak forgery in practice. Next, we do not require that adversarially generated issuance tokens are satisfied until the issuance protocol finishes. That is, we consider forgery

of issuance tokens to be a problem only if they are subsequently used in an successful issuance protocol. Requiring the stronger property that issuance tokens are unforgeable by themselves is possible, but would make our definition considerably more complicated (e.g., issuance tokens without successful issuance would have to be entered in the list of $\mathcal{P}$ without the specification of the to-be-issued credential). However, this relaxation does not pose a problem in practice, as an invalid issuance token without subsequent successful issuance does not give an adversary any meaningful power. Third, for blindly issued attributes we only require that they satisfy the relation defined by E, but do not forbid, e.g., that an adversary runs the issuance protocol twice with the same issuance token, but with the resulting credentials containing different values for the blinded attributes as long as they satisfy E. Again, this is not a real-world problem, as the adversary could otherwise just run multiple sessions for different issuance tokens, as the issuer will, by definition of blind attributes, not obtain any guarantees on those attributes apart from what E specifies. Finally, we do not consider it a problem if tokens for earlier epochs than the one underlying a credential are generated. We believe that this makes sense in the presence of off-line verifiers who cannot update their revocation information continuously. However, our definition and generic construction could easily be modified to forbid the generation of such tokens.

3.3.4 Simulatable Privacy

This section contains definitions of the two versions of privacy of a PABC scheme: *Privacy* and *Weak Privacy*, the difference between the two is discussed in more detail at the end of this section.

3.3.4.1 Privacy

We define privacy by requiring that all issuance protocols and presentation tokens performed by honest users can be simulated using only the public information that is explicitly revealed during the issuance or presentation. That is, the simulator is not given the actual credentials, values of hidden attributes, or even the index of the user that is supposed to perform the presentation or issuance, but must provide a view to the adversary that is indistinguishable from a real user. Formalizing this is not straightforward, however. It does not suffice to require two separate simulators that work for issuance and presentation, respectively, because pseudonyms and revocation introduce explicit dependencies across different issuance and presentation queries that must also be reflected in the simulation. Moreover, the simulator must not generate presentation tokens that could not have been generated in the real world, e.g., because the user does not have the required credentials. But as the simulator does not see any user indices or hidden attribute values, it cannot know which queries can be satisfied.

We therefore define a game where an adversary A either runs in a real world, where he has access to an honest user oracle performing the actual protocols, or runs in a simulated world, where A's oracle queries are first filtered by a *filter* $\mathcal{F}$ and then responded to by a stateful simulator S. The filter's role is to sanitize the queries from non-public information such as user indices, credential identifiers, etc., and to intercept queries that could not be satisfied in the real world. Note that the filter thereby enforces that the adversary can only obtain presentation tokens for valid inputs. Thus, the same must be guaranteed by the credential system as well, otherwise the adversary could easily distinguish between both worlds.

Definition 3.3.2 (Privacy)**.** *A* PABC *system is* private*, if there exist two PPT algorithms* S_1 *and* S_2 *such that no PPT adversary* A *has a non-negligible chance of winning the game described in Figure 3.2.*

Here, $\mathcal{O}^{\mathtt{user}}(spar)$ *behaves as described in § 3.2, while* $\mathcal{F}$ *maintains initially empty lists* C *and* P*, a counter* $ctr = 0$*, and state information* $\mathsf{st}_{\mathsf{S}} = \tau$*, and responds to queries as follows:*

Experiment $\mathsf{Privacy}_{\mathsf{A}}(1^\kappa, n_{\mathrm{U}})$ and $\mathsf{WeakPrivacy}_{\mathsf{A}}(1^\kappa, n_{\mathrm{U}})$**:**
$b \xleftarrow{\$} \{0,1\}$

If $b = 0$:	Else:
$spar \xleftarrow{\$} \mathsf{SPGen}(1^\kappa)$	$(spar, \tau) \xleftarrow{\$} \mathsf{S}_1(1^\kappa)$
$usk^*_U \xleftarrow{\$} \mathsf{UKGen}(spar)$ for $U = 1, \ldots, n_{\mathrm{U}}$	
$USK^* = \{usk^*_U\}_{U=1}^{n_{\mathrm{U}}}$	
$b' \xleftarrow{\$} \mathsf{A}^{\mathcal{O}^{\mathrm{user}}(USK^*,\cdot)}(spar, n_{\mathrm{U}})$	$b' \xleftarrow{\$} \mathsf{A}^{\mathcal{F}(n_{\mathrm{U}},\cdot)\|\mathsf{S}_2(\tau)}(spar, n_{\mathrm{U}})$

Return 1 if and only if
$b = b'$.

Figure 3.2: $\mathsf{Privacy}_{\mathsf{A}}(1^\kappa, n_{\mathrm{U}})$ or $\mathsf{WeakPrivacy}_{\mathsf{A}}(1^\kappa, n_{\mathrm{U}})$ depending on the definition of $\mathcal{F}$

- *On input* $(\texttt{present}, U, scope, (ipk_i, RI_i, cid_i, R_i)_{i=1}^k, E, M)$*, the filter checks that* $U \in \{1, \ldots, n_U\}$ *and that, for all* $i = 1, \ldots, k$*, a tuple* $(U, cid_i, ipk_i, (a_{i,j})_{j=1}^{n_i}, \mathsf{rev}_i) \in C$ *exists. Here,* rev_i *is the code of an algorithm that on input* RI_i *outputs a bit indicating whether the credential is to be considered revoked. Further, for all credentials the filter checks if* $\mathsf{rev}_i(RI_i) = 0$ $\forall i = 1, \ldots, k$ *and that* $a_{i,j} = a_{i',j'}$ *for all* $((i,j),(i',j')) \in E$*. If any of the checks fails, the filter returns* $\perp$*. If* $scope \neq \varepsilon$ *and* $(U, scope, p) \notin P$ *then* $\mathcal{F}$ *sets* $ctr \leftarrow ctr + 1$*,* $p \leftarrow ctr$*, and adds* $(U, scope, p)$ *to* P*. It then executes* $(\mathsf{st}_{\mathsf{S}}, nym, pt) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_{\mathsf{S}}, \texttt{present}, scope, p, (ipk_i, (a_{i,j})_{j \in R_i})_{i=1}^k, E, M)$*. Finally, it returns* (nym, pt) *to* A*.*

- *On input* $(\texttt{obtain}, U, scope, rh, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k+1}, E, M, (a_{k+1,j})_{j=1}^{n_{k+1}})$*, the filter checks that* $U \in \{1, \ldots, n_U\}$ *and that, for all* $i = 1, \ldots, k$*, a tuple* $(U, cid_i, ipk_i, (a_{i,j})_{j=1}^{n_i}, \mathsf{rev}_i) \in C$ *exists. Further, for all credentials the filter checks if* $\mathsf{rev}_i(RI_i) = 0$ $\forall i = 1, \ldots, k$ *and that* $a_{i,j} = a_{i',j'}$ *for all* $((i,j),(i',j')) \in E$*. If any of the checks fails, the filter returns* $\perp$*. The filter then looks up the same value* p *as in* `present`*. It then executes* $(\mathsf{st}_{\mathsf{S}}, nym, pit) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_{\mathsf{S}}, \texttt{obtain}, scope, p, (ipk_i, (a_{i,j})_{j \in R_i})_{i=1}^{k+1}, E, M, rh)$ *and returns* (nym, pit) *to* A*. For the subsequent flows in the issuance protocol,* $\mathcal{F}$ *answers each incoming message* M_{in} *from* A *by running* $(\mathsf{st}_{\mathsf{S}}, M_{\mathrm{out}}) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_{\mathsf{S}}, M_{\mathrm{in}})$*. At the last flow,* S_2 *returns a tuple* $(\mathsf{st}_{\mathsf{S}}, M_{\mathrm{out}}, cid, \mathsf{rev})$*. If* $cid \neq \perp$*,* $\mathcal{F}$ *adds a tuple* $(U, cid, ipk_{k+1}, (a_{k+1,j})_{j=1}^{n_{k+1}}, \mathsf{rev})$ *to* C *and returns* M_{out} *to* A*.*

We require that for every PPT adversary A *and every* $n_{\mathrm{U}} \in \mathbb{N}$ *we have that for some negligible function* ν *it holds that:*

$$\Pr[\mathsf{Privacy}_{\mathsf{A}}(1^\kappa, n_{\mathrm{U}}) = 1] \leq 1/2 + \nu(\kappa)\,.$$

3.3.4.2 Weak Privacy

The definition above ensures a very strong notion of privacy that not all instantiations of PABC schemes are able to satisfy. Namely, some schemes do not provide unlinkability across multiple presentation tokens *pt* that were derived from the same credential. For instance, this is the case for Microsoft's UProve, where an arbitrary number of presentations cannot be linked to a specific issuance session, but any two presentations of the same credential can be linked to each other.

In the following we therefore introduce a relaxed notion of privacy, called *weak privacy*. In this definition we give the simulator some more information to be able to generate "linkable" presentation tokens if the adversary requests multiple presentations tokens for some credential. Formally, we do this by giving S_2 "anonymized" pointers to credential identifiers p_{cid} as input. Thus, the simulator is aware if the same credential (signature) is used in multiple presentation sessions and can prepare the simulated token accordingly. Due to the "anonymization" of

the credential identifier p_{cid}, the simulator still does not learn the connection between an issued credential (or rather its *cid*) and a presentation token, thus untraceability (meaning presentation sessions cannot be linked to the actual issuance of the credential) still holds.

The only essential difference between the previous definition of *Privacy* and the below definition of *Weak Privacy* is that the filter maintains an additional list $\mathcal{L}_{cid}$ of credential identifiers and by using this list it can supply S_2 with the credential identifier p_{cid}. By p_{cid} S_2 can simulate the linkability in the simulated scenario if this is present in the real scenario.

Definition 3.3.3 (Weak Privacy)**.** *A* PABC *system is* weak private*, if there exist two PPT algorithms* S_1 *and* S_2 *such that no PPT adversary* A *has a non-negligible chance of winning the game described in Figure 3.2.*

Here, $\mathcal{O}^{\mathtt{user}}(spar)$ *behaves as described in § 3.2, while* $\mathcal{F}$ *maintains initially empty lists* C*,* P*,* L*, two counter* $ctr_{nym} = 0$ *and* $ctr_{cid} = 0$*, and state information* $\mathsf{st}_\mathsf{S} = \tau$*, and responds to queries as follows:*

- *On input* $(\mathtt{present}, U, scope, (ipk_i, RI_i, cid_i, R_i)_{i=1}^k, E, M)$*, the filter checks that* $U \in \{1, \ldots, n_U\}$ *and that, for all* $i = 1, \ldots, k$*, a tuple* $(U, cid_i, ipk_i, (a_{i,j})_{j=1}^{n_i}, \mathsf{rev}_i) \in C$ *exists. Here,* rev_i *is the code of an algorithm that on input* RI_i *outputs a bit indicating whether the credential is to be considered revoked. Further, for all credentials the filter checks if* $\mathsf{rev}_i(RI_i) = 0\ \forall i = 1, \ldots, k$ *and that* $a_{i,j} = a_{i',j'}$ *for all* $((i,j),(i',j')) \in E$*. If any of the checks fails, the filter returns* $\perp$*. If scope* $\neq \varepsilon$ *and* $(U, scope, p) \notin P$ *then* $\mathcal{F}$ *sets* $ctr_{nym} \leftarrow ctr_{nym}+1$*,* $p \leftarrow ctr_{nym}$*, and adds* $(U, scope, p)$ *to* P*. For each* cid_i *with* $i = 1, \ldots, k$*,* $\mathcal{F}$ *checks if a tuple* (cid_i, p_{cid_i}) *in* L *exist. If not, it sets* $ctr_{cid} \leftarrow ctr_{cid} + 1$*,* $p_{cid} \leftarrow ctr_{cid}$*, and adds a tuple* (cid, p_{cid}) *to* L*. It then executes* $(\mathsf{st}_\mathsf{S}, nym, pt) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_\mathsf{S}, \mathtt{present}, scope, p, (ipk_i, (a_{i,j})_{j\in R_i}, p_{cid_i})_{i=1}^k, E, M)$*. Finally, it returns* (nym, pt) *to* A*.*

- *On input* $(\mathtt{obtain}, U, scope, rh, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k+1}, E, M, (a_{k+1,j})_{j=1}^{n_{k+1}})$*, the filter checks that* $U \in \{1, \ldots, n_U\}$ *and that, for all* $i = 1, \ldots, k$*, a tuple* $(U, cid_i, ipk_i, (a_{i,j})_{j=1}^{n_i}, \mathsf{rev}_i) \in C$ *exists. Further, for all credentials the filter checks if* $\mathsf{rev}_i(RI_i) = 0\ \forall i = 1, \ldots, k$ *and that* $a_{i,j} = a_{i',j'}$ *for all* $((i,j),(i',j')) \in E$*. If any of the checks fails, the filter returns* $\perp$*. The filter then looks up the same values* p *and* $\{p_{cid_i}\}_{i=1}^k$ *as in* `present`*. It then executes* $(\mathsf{st}_\mathsf{S}, nym, pit) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_\mathsf{S}, \mathtt{obtain}, scope, p, (ipk_i, (a_{i,j})_{j\in R_i}, p_{cid_i})_{i=1}^{k+1}, E, M, rh)$ *and returns* (nym, pit) *to* A*. For the subsequent flows in the issuance protocol,* $\mathcal{F}$ *answers each incoming message* M_{in} *from* A *by running* $(\mathsf{st}_\mathsf{S}, M_{\text{out}}) \xleftarrow{\$} \mathsf{S}_2(\mathsf{st}_\mathsf{S}, M_{\text{in}})$*. At the last flow,* S_2 *returns a tuple* $(\mathsf{st}_\mathsf{S}, M_{\text{out}}, cid, \mathsf{rev})$*. If* $cid \neq \perp$*,* $\mathcal{F}$ *adds a tuple* $(U, cid, ipk_{k+1}, (a_{k+1,j})_{j=1}^{n_{k+1}}, \mathsf{rev})$ *to* C *and returns* M_{out} *to* A*.*

We require that for every PPT adversary A *and every* $n_{\text{U}} \in \mathbb{N}$ *we have that for some negligible function* ν *it holds that:*

$$\Pr[\mathsf{WeakPrivacy}_\mathsf{A}(1^\kappa, n_{\text{U}}) = 1] \leq 1/2 + \nu(\kappa)\,.$$

3.3.4.3 Comparison of Privacy and Weak Privacy

The following lemma sums up the relation between the two privacy definitions stated above, showing that privacy is a strictly stronger requirement than weak privacy. However, weak privacy is an interesting property, as for specific applications it still provides sufficiently strong guarantees, but, on the other hand, might be achievable for less computational costs than privacy.

Before we state the lemma we note that the only difference between the two experiments for the definitions is the definition of the filter $\mathcal{F}$. Next we note that the only difference between the two filters is that in Definition 3.3.3, the definition for weak privacy, the filter keeps the list

L, which is used to give the simulator the same identifier, the values p_{cid}, if a credential is to be presented multiple times. This gives the simulator the ability to simulate a scheme where two presentation of the same credential can be linked together.

Lemma 3.3.4. *Definition 3.3.2 implies Definition 3.3.3, but not vice versa. More formally, every private* PABC *scheme is also weakly private, but there exist weakly private* PABC *schemes that are not (fully) private.*

Proof. To see the implication, let S be the simulator of a (fully) private PABC scheme. Then S' defined as follows is a valid simulator for weak privacy: it forwards all inputs an internal instance of S after removing p_{cid}, and outputs whatever S outputs. It is easy to see as after removing p_{cid} the experiments of Definitions 3.3.2 and 3.3.3 coincide, and thus any adversary breaking Definition 3.3.3 would directly break Definition 3.3.2 as well.

To see that the other direction does not hold we refer to our generic construction (§ 5) and the instantiation based on U-Prove (§ 7.4). By Theorem 6.4.2, the resulting PABC scheme is weakly private. However, it is not private, as the presentation tokens in this scheme reveal parts of the credential without re-randomizing them, and thus it is trivial to link presentation tokens of the same credential. □

Chapter 4

Building Blocks

In this section we define several building blocks such as signature or pseudonym schemes, the algorithms and protocols they execute, and the formal security requirements they need to satisfy. Compared to PABC-systems, most of the security requirements presented in the following are relatively easy to formalize and also to prove for a specific instantiation. However, in § 5 we will show that these properties are actually sufficient to obtain PABC-systems by giving a generic construction for PABC-systems from these building blocks.

4.1 Global Setup

Global system parameters allow one to specify shared system parameters that are to be used by all the other building blocks.

Global System Parameter Generation. The global system parameters are generated as

$$spar_{\mathsf{g}} = (1^{\kappa}, \mathcal{AS}, \ell, \ell', L, spar_{\mathsf{c}}, ck, spar'_{\mathsf{g}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{g}}(1^{\kappa}) \,,$$

where:

- ℓ is an integer such that the attribute space $\mathcal{AS}$ of the signature scheme is a subset of $\pm\{0,1\}^{\ell}$, but the revocation and pseudonym systems support inputs from at least $\pm\{0,1\}^{\ell}$,
- $\ell' \geq \ell$ is an integer,
- $spar_{\mathsf{c}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{c}}(1^{\kappa}, \ell')$,
- $ck \xleftarrow{\$} \mathsf{ComKGen}(spar_{\mathsf{c}})$ a public master commitment key,
- L specifies the maximum number of attributes that can signed at once, and
- $spar'_{\mathsf{g}}$ potentially specifies further global parameters.

4.2 Commitment Schemes

In the following we introduce the syntax we use for commitment schemes, as well as the security properties such a scheme has to satisfy.

4.2.1 Syntax

Basically, the commitment schemes we use are an extension of the standard notion of commitment schemes. Besides the usual algorithms and properties, we require the possibility to securely prove knowledge of the content of a commitment.

System parameter generation. The commitment system parameters are generated as

$$spar_{\mathsf{c}} = (1^{\kappa}, \lambda, spar'_{\mathsf{c}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{c}}(1^{\kappa}, \lambda)\,,$$

where:

- λ is the maximum bit length of messages that can be committed to, and
- $spar'_{\mathsf{c}}$ potentially specifies further commitment parameters.

Commitment key generation. This algorithm computes a public commitment key as:

$$ck \xleftarrow{\$} \mathsf{ComKGen}(spar_{\mathsf{c}})\,.$$

Committing to messages. Given a message m, a commitment key ck and system parameters $spar_{\mathsf{c}}$ this algorithm is defined as:

$$(c, o) \xleftarrow{\$} \mathsf{Com}(ck, spar_{\mathsf{c}}, m)\,,$$

where c is the commitment to m, and o is the opening information to c.

Commitment opening verification. This algorithm verifies that the opening o of a commitment c is correct for a message m:

$$\mathsf{accept/reject} \xleftarrow{\$} \mathsf{ComOpenVf}(ck, spar_{\mathsf{c}}, c, m, o)\,.$$

Commitment opening proof. This algorithm non-interactively proves knowledge of content of a commitment:

$$\pi \xleftarrow{\$} \mathsf{ComPf}(ck, spar_{\mathsf{c}}, c, o, m)\,.$$

Commitment proof verification. This algorithm verifies a proof π for a commitment c:

$$\mathsf{accept/reject} \xleftarrow{\$} \mathsf{ComProofVf}(ck, spar_{\mathsf{c}}, c, \pi)\,.$$

For notational convenience, we will omit $spar_{\mathsf{c}}$ and ck as inputs to the algorithms if they are clear from the context.

4.2.2 Security Definitions for Commitment Schemes

We next recapitulate the standard security definitions of commitment schemes using the interfaces defined in § 4.2.1.

Correctness guarantees that honestly computed commitments for honest keys can be successfully opened:

Definition 4.2.1 (Correctness)**.** *A commitment scheme is* correct*, if for every $\lambda \in \mathbb{Z}$ and $m \in \{0,1\}^{\lambda}$ we have that:*

$$\begin{aligned}&\Pr[\mathsf{ComOpenVf}(ck, spar_{\mathsf{c}}, c, m, o) = \mathsf{reject}\,:\\ &\quad spar_{\mathsf{c}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{c}}(1^{\kappa}, \lambda),\, ck \xleftarrow{\$} \mathsf{ComKGen}(spar_{\mathsf{c}}),\, (c, o) \xleftarrow{\$} \mathsf{Com}(ck, spar_{\mathsf{c}}, m)] = 0\,.\end{aligned}$$

The binding property guarantees that no efficient adversary can find valid openings of a single commitment to two different messages:

Definition 4.2.2 (Computationally binding)**.** *A commitment scheme is called* computationally binding*, if it is infeasible to open a commitment to two different messages. That is, for every efficient algorithm* A *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{ComOpenVf}(ck, m, c, o, spar_{\mathsf{c}}) = \mathsf{accept} \ \wedge \ \mathsf{ComOpenVf}(ck, m', c, o', spar_{\mathsf{c}}) = \mathsf{accept} : spar_{\mathsf{c}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{g}}(1^{\kappa}, \lambda), ck \xleftarrow{\$} \mathsf{ComKGen}(spar_{\mathsf{c}}), (c, m, m', o, o') \xleftarrow{\$} \mathsf{A}(ck, spar_{\mathsf{g}})] \leq \nu(\kappa)\,.$$

The hiding property guarantees that a commitment does not leak any information about the contained message:

Definition 4.2.3 (Statistically/computationally hiding)**.** *A commitment scheme is* statistically/computationally hiding*, if for every* $m, m' \in \mathcal{M}$ *the distributions of* $\mathsf{Com}(ck, m, spar_{\mathsf{c}})$ *and* $\mathsf{Com}(ck, m', spar_{\mathsf{c}})$ *are statistically/computationally indistinguishable.*

Additionally to these standard properties, we also require that knowledge of the content of a commitment can be proved in a zero-knowledge way:

Definition 4.2.4 (Opening extractability)**.** *A commitment scheme is* opening extractable*, if and only if* $(\mathsf{SPGen}_{\mathsf{c}}, \mathsf{ComPf}, \mathsf{ComProofVf})$ *is a non-interactive ZK proof of knowledge for the following relation* R *, cf. also* § 2.2.1*:*

$$(c, (o, m)) \in R :\Leftrightarrow \mathsf{ComOpenVf}(c, m, o) = \mathsf{accept}\,.$$

4.3 Privacy-Enhancing Attribute-Based Signatures

We next define the main building block: privacy-enhancing attribute-based signatures (PABS).

4.3.1 Syntax

We first define the parties and the protocols they execute. Informally, parties are split into issuers signing attributes, users obtaining signatures, and verifiers checking whether users possess valid signatures on certain attributes. After setting up some PABS-specific system parameters, each issuer computes his signing/verification key pair, such that everybody can verify that keys are well-formed. At issuance time, users can reveal certain attributes to the issuer, e.g., depending on the issuer's policy, and get the remaining attributes signed blindly. Having received a signature, a user can verify its correctness. Presentation is then done in a non-interactive manner: users compute signature presentation tokens, potentially revealing certain attributes, and verifiers can check these tokens.

System Parameter Generation. The signature system parameters are generated as

$$spar_{\mathsf{s}} = (spar_{\mathsf{g}}, \mathcal{AS}, spar'_{\mathsf{s}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{s}}(spar_{\mathsf{g}})\,,$$

where the input are global system parameters, and

- $\mathcal{AS} \subseteq \pm\{0,1\}^{\ell}$ is the attribute space, and
- $spar'_{\mathsf{s}}$ potentially specifies further signature parameters.

The system parameters are input to any of the other algorithms of the scheme. However, for notational convenience, we will sometimes not make this explicit.

Key Generation. An issuer generates a key pair $(ipk, isk) \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathsf{s}})$.

Key Verification. A public key ipk of an issuer can be verified for correctness with respect to a security parameter 1^{κ} as

$$\mathsf{accept/reject} \leftarrow \mathsf{KeyVf}(ipk)\,.$$

Signature Issuance. Issuance of an attribute signature is an interactive protocol between a user and a signature issuer:

$$(sig/\perp, \varepsilon) \xleftarrow{\$} \langle \mathcal{U}.\mathsf{Sign}(ipk, (c_j, o_j)_{j \notin R}, \vec{a}) \; ; \mathcal{I}.\mathsf{Sign}(isk, (a_i)_{i \in R}, (c_j, \pi_j)_{j \notin R}) \rangle \, ,$$

where:

- $\vec{a} = (a_1, \ldots, a_L)$ are the attributes to be signed,
- R denotes indices of attributes that are revealed to the issuer (the other ones are unrevealed)
- c_j is a verified commitment to a_j, o_j is the associated opening information, and π_j is a non-interactive proof of knowledge of the opening of c_j. In particular, c_j might be re-used from a preceding signature presentation, allowing users to blindly carry over attributes into new credentials.

At the end of the protocol the user either obtains a signature *sig* or $\perp$ in case the issuance failed.

Signature Verification. The correctness of a signature can be verified using SigVf on input a signature *sig*, attributes $\vec{a}$ and a issuer public key *ipk*:

$$\mathsf{accept/reject} \leftarrow \mathsf{SigVf}(sig, \vec{a}, ipk) \, .$$

Signature Presentation Token Generation. The user can compute a signature presentation token *spt* that proves that he possesses a signature for a set of revealed attributes R and committed attributes $(c_j, o_j)_{j \in C}$ where $C \cap R = \emptyset$. Furthermore, a signature presentation token can be bound to a specific message M specifying, e.g., some context information or random nonce to disable adversaries to re-use signature presentation tokens in subsequent sessions:

$$spt/\perp \xleftarrow{\$} \mathsf{SignTokenGen}(ipk, sig, \vec{a}, R, (c_j, o_j)_{j \in C}, M) \, .$$

Signature Presentation Token Verification. The correctness of a signature presentation token can be verified publicly as:

$$\mathsf{accept/reject} \xleftarrow{\$} \mathsf{SignTokenVf}(ipk, spt, (a_i)_{i \in R}, (c_j)_{j \in C}, M) \, .$$

4.3.2 Security Definitions for PABS Schemes

We require that honest parties are always able to succeed in the previous algorithms. That is, none of the above algorithms should output $\perp$ or reject, whenever the inputs were computed honestly and the parties followed the protocol specifications.

Definition 4.3.1 (Signature Completeness)**.** *A* PABS*-system is* complete*, if there exist negligible functions ν_i, $i = 1, 2, 3$, such that the following is satisfied for all $\vec{a} \in \mathcal{AS}(\kappa)^L$, all $C, D, R \subseteq \{1, \ldots, L\}$ with $C \cap D = \emptyset$ and all messages M:*

Honestly computed keys are well-formed with overwhelming probability:

$$\begin{aligned} \Pr[\mathsf{KeyVf}(ipk, 1^\kappa) = \mathsf{reject} \; : \; & spar_{\mathsf{g}} \xleftarrow{\$} \mathsf{SPGen_g}(1^\kappa), \\ & spar_{\mathsf{s}} \xleftarrow{\$} \mathsf{SPGen_s}(spar_{\mathsf{g}}), (isk, ipk) \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathsf{s}})] \leq \nu_1(\kappa) \end{aligned}$$

If all parties are honest, the user obtains a valid signature with overwhelming probability:

$$\begin{aligned} \Pr[sig = \perp \vee \mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} \; : \; & spar_{\mathsf{s}} \xleftarrow{\$} \mathsf{SPGen_s}(\mathsf{SPGen_g}(1^\kappa)), \\ & (isk, ipk) \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathsf{s}}), (c_j, o_j) \xleftarrow{\$} \mathsf{Com}(a_j) \; \forall j \notin R, \\ & (sig, \varepsilon) \xleftarrow{\$} \langle \mathcal{U}.\mathsf{Sign}(ipk, (c_j, o_j)_{j \notin R}, \vec{a}); \mathcal{I}.\mathsf{Sign}(isk, (a_i)_{i \in R}, (c_j)_{j \notin R}) \rangle] \leq \nu_2(\kappa) \end{aligned}$$

For honest issuers, an honest user can produce valid signature presentation tokens with overwhelming probability:

$$\begin{aligned}&\Pr[\mathsf{SignTokenVf}(ipk, spt, (a_i)_{i\in D}, (c_j)_{j\in C}, M) = \mathsf{reject} \ : \\ &\quad spar_{\mathsf{g}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{g}}(1^\kappa), spar_{\mathsf{s}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{s}}(spar_{\mathsf{g}}), (isk, ipk) \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathsf{s}}), \\ &\quad (c_j, o_j) \xleftarrow{\$} \mathsf{Com}(a_j)\ \forall j \notin R,\ (c'_j, o'_j) \xleftarrow{\$} \mathsf{Com}(a_j)\ \forall j \in C \\ &\quad (sig, \varepsilon) \xleftarrow{\$} \langle \mathcal{U}.\mathsf{Sign}(ipk, (c_j, o_j)_{j\notin R}, \vec{a}); \mathcal{I}.\mathsf{Sign}(isk, (a_i)_{i\in R}, (c_j)_{j\notin R})\rangle, \\ &\quad spt \xleftarrow{\$} \mathsf{SignTokenGen}(ipk, sig, \vec{a}, D, (c'_j, o'_j)_{j\in C}, M)] \leq \nu_3(\kappa).\end{aligned}$$

The following definition is similar to the standard unforgeability and gives the user access to an issuance oracle $\mathcal{O}_{\texttt{issuer}}$, from which he can obtain signatures on attributes of his choice. To capture also the information the adversary can gain from acting as a verifier with honest users, we additionally grant A access to a user oracle $\mathcal{O}_{\texttt{user}}$. The adversary can ask the oracle to get signatures for attributes of his choice from the issuer, for which A will only receive a handle but not the signature itself. Subsequently the adversary can request presentation proofs for those unrevealed signatures where the oracle executes the part of the honest user. Finally, the adversary wins the unforgeability game if he manages to output a valid signature for attributes he has never sent to the issuer oracle.

Note that we also consider signatures on attributes sent to $\mathcal{O}_{\texttt{user}}$ as forgeries, and thus our notion also captures the property that a verifier should not able to impersonate the user after having received a presentation proof. On the other hand, we do not require strong unforgeability, i.e., given a signature for attributes $\vec{a}$ an adversary may be allowed to derive a different signature for the same attributes. On the contrary, our construction given in § 7.3 heavily relies on the fact that a user can re-randomize signatures.

Definition 4.3.2 (Signature Unforgeability)**.** *A* PABS*-scheme is called* unforgeable*, if there exists an efficient algorithm* $\mathsf{E}^{\mathsf{s}} = (\mathsf{E}^{\mathsf{s}}_1, \mathsf{E}^{\mathsf{s}}_2)$*, called* signature extractor*, that satisfies the following properties:*

- $\mathsf{E}^{\mathsf{s}}_1$ *outputs parameters and trapdoors, such that the parameters are indistinguishable from correctly computed system parameters:*

$$\{spar_{\mathsf{s}} : \ spar_{\mathsf{s}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{s}}(\mathsf{SPGen}_{\mathsf{g}}(1^\kappa))\} \ \sim \ \{spar_{\mathsf{s}} : \ (spar_{\mathsf{s}}, \tau_{\mathsf{s}}) \xleftarrow{\$} \mathsf{E}^{\mathsf{s}}_1(\mathsf{SPGen}_{\mathsf{g}}(1^\kappa))\},$$

  and

- *for every efficient adversary* A *the following probability is negligible in the security parameter:*

$$\begin{aligned}&\Pr\Big[\mathsf{SignTokenVf}(ipk, spt, (a_i)_{i\in R}, (c_j)_{j\in C}, M) = \mathsf{accept} \ \wedge \ \Big(\mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} \ \vee \\ &\quad \exists k \in C : \mathsf{ComOpenVf}(a_k, c_k, o_k) = \mathsf{reject} \ \vee \ \big((a_i)_{i\in R}, (c_j)_{j\in C}, M) \notin \mathcal{L}_{\text{spt}} \ \wedge \ \vec{a} \notin \mathcal{L}_{\text{iss}}\big)\Big) \ : \\ &\quad (spar_{\mathsf{s}}, \tau_{\mathsf{s}}) \xleftarrow{\$} \mathsf{E}^{\mathsf{s}}_1(\mathsf{SPGen}_{\mathsf{g}}(1^\kappa)),\ (isk, ipk) \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathsf{s}}), \\ &\quad ((a_i)_{i\in R}, (c_j)_{j\in C}, spt, M) \xleftarrow{\$} \mathsf{A}^{\mathcal{O}^{\texttt{issuer}}, \mathcal{O}^{\texttt{user}}}(spar_{\mathsf{s}}, ipk), \\ &\quad (sig, (a_j)_{j\notin R}, (o_k)_{k\in C}) \xleftarrow{\$} \mathsf{E}^{\mathsf{s}}_2(\tau, ipk, (a_i)_{i\in R}, (c_j)_{j\in C}, spt), \vec{a} \leftarrow (a_i)_{i\in R} \cup (a_j)_{j\notin R}\Big].\end{aligned}$$

Here, the oracles $\mathcal{O}_{\texttt{issuer}}$ *and* $\mathcal{O}_{\texttt{user}}$ *are defined as follows:*

- *The oracle* $\mathcal{O}^{\texttt{issuer}}$ *allows the adversary to receive signatures from the issuer. It maintains an initially empty list* $\mathcal{L}_{\text{iss}}$*. On input* $((a_i)_{i\in R}, (c_j, \pi_j)_{j\notin R})$*, the oracle first checks that* $\mathsf{ComProofVf}(c_j, \pi_j) = \mathsf{accept}$*, and then extracts* $(a_j, o_j) \xleftarrow{\$} \mathsf{E}^{\mathsf{c}}_2(\tau_{\mathsf{c}}, c_j, \pi_j)$ *for all* $j \notin R$*, and defines* $\vec{a}$ *in the canonical way. It initiates an instance of the issuance protocol for* $((a_i)_{i\in R}, (c_j)_{j\notin R})$ *acting as an honest signature issuer using isk. Finally, the oracle adds* $\vec{a}$ *to* $\mathcal{L}_{\text{iss}}$*.*

- *The oracle* $\mathcal{O}^{\texttt{user}}(\cdot)$ *allows the adversary to interact with "honest users". It maintains a list* $\mathcal{L}_{\text{sig}}$ *of tuples* $(cid, \vec{a}, sig)$ *where sig denotes a signature on attributes* $\vec{a}$*, associated with the unique handle cid. The adversary can query the oracle in two modes:*
  - *Being called with* $(\texttt{obtain}, \vec{a})$ *the oracle initiates an instance of the issuance protocol* $\mathcal{U}.\mathsf{Issue}(ipk, \varepsilon, \vec{a})$ *in interaction with* $\mathcal{I}.\mathsf{Issue}(isk, \vec{a}, \varepsilon)$*. When the former outputs a signature sig, it adds the tuple* $(cid, \vec{a}, sig)$ *to* $\mathcal{L}_{\text{sig}}$*, where cid is a unique handle to the signature. It returns cid to* A*.*
  - *Being called with* $(\texttt{token}, cid, R, (c_j, o_j)_{j\in C}, M)$ *where* $C \cap R = \emptyset$ *the oracle generates a signature token. It first checks that a tuple* $(cid, \vec{a}, sig) \in \mathcal{L}_{\text{sig}}$ *exists and that* $\mathsf{ComOpenVf}(a_j, c_j, o_j) = \mathsf{accept}\ \forall j \in C$*. If so, it returns spt* $\xleftarrow{\$} \mathsf{SigTokenGen}(ipk, sig, \vec{a}, R, (c_j, o_j)_{j\in C}, M)$ *to the adversary, and adds* $((a_i)_{i\in R}, (c_j)_{j\in C}, M)$ *to the initially empty list* $\mathcal{L}_{\text{spt}}$*.*

We now define *user privacy*, i.e., the security guarantees for users against malicious signers and signature token verifiers. In a nutshell, we require that malicious issuers and verifiers cannot learn more than the attributes a user willingly reveals. This includes that issuers and verifiers must not be able to link different transactions unless this is implied by the revealed attributes (e.g., because an attribute is unique).

Similarly to the privacy definitions of PABC, privacy of PABS schemes also comes in two flavors, a strong and a weaker. As we will see later there exist schemes only fulfilling *Weak User Privacy* and this property is enough to achieve a weakly private PABC scheme. For both the weak and the strong version we define user privacy by requiring that the user-side sign protocol and the generation of signature tokens can be simulated given only the public information, i.e., the revealed attributes and the commitments to the hidden attributes. Here we are, however, in a much simpler situation compared with the full PABC scheme, as it is actually sufficient to just consider a single sign protocol and for the strongest version of privacy only a single signature token. To be able to simulate the user with only the public inputs, the system parameters for the signature scheme need to come with a trapdoor that can be exploited by the simulators $\mathsf{S}_{\texttt{iss}}$ (that simulates the user-side of the issuance) and $\mathsf{S}_{\texttt{pres}}$ (that simulates the token generation). Thus, in both experiments we require a third simulator $\mathsf{S}_{\texttt{params}}$ that outputs system parameters which are indistinguishable from the real ones, but provide such a trapdoor. For the strong version of privacy we can follow the composable zero-knowledge paradigm [GS08, GS12], and this trapdoor is given to both, the simulator *and the adversary*. While this might look surprising at first glance, it turns out to be necessary for proving the security of the generic construction presented in the following section, as otherwise privacy guarantees could only be made for a single presentation, but not for polynomially many by potentially different users. In the weak version of privacy we cannot use the composable zero-knowledge paradigm and to be able to give security guaranties for more than a single presentation, we use an approach very similar to the privacy definitions of PABC where in the real world case of the experiment the adversary has access to an oracle representing a real user and in the simulated case the adversary accesses the simulator trough a filter filtering all secret information and thereby only giving the public information to the simulator.

Finally, one needs to rule out that dishonest signers can generate public keys that allow them to somehow bias the derived signatures and thus to link them. To thwart such schemes, we require that issuers' public keys are well-formed, i.e., verify under KeyVf. This is defined as follows.

Definition 4.3.3 (Key Correctness)**.** *No efficient adversary can compute an issuer public key* ipk^* *satisfying* $\mathsf{KeyVf}(ipk^*) = \mathsf{accept}$ *such that there do not exist random coins for* IKGen *generating the same public key, with more than negligible probability.*

We are now ready to present our definition of user privacy.

Experiment $\mathsf{BlindIss}_{\mathsf{A}}(1^\kappa)$**:**
$(spar_{\mathtt{s}}, \tau_{\mathtt{s}}) \xleftarrow{\$} \mathsf{S}_{\mathtt{params}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa))$
$b \xleftarrow{\$} \{0,1\}$
$(ipk, R, \vec{a}, \mathsf{st}) \xleftarrow{\$} \mathsf{A}_{1,\mathtt{iss}}(spar_{\mathtt{s}}, \tau_{\mathtt{s}})$
$(c_i, o_i) \xleftarrow{\$} \mathsf{Com}(a_i)\ \forall i \notin R$

If $b = 0$:	Else:
$b' \xleftarrow{\$} \mathsf{A}_{2,\mathtt{iss}}^{\mathcal{O}^{\mathtt{s.obtain}}(spar_{\mathtt{s}}, ipk, \vec{a}, (c_i,o_i)_{i\notin R})}((c_i)_{i\notin R}, \mathsf{st})$	$b' \xleftarrow{\$} \mathsf{A}_{2,\mathtt{iss}}^{\mathsf{S}_{\mathtt{iss}}(spar_{\mathtt{s}}, \tau_{\mathtt{s}}, ipk, (a_i)_{i\in R}, (c_i)_{i\notin R})}((c_i)_{i\notin R}, \mathsf{st})$

Return 1 if and only if:
$b = b'$ and
$\mathsf{KeyVf}(ipk) = \mathsf{accept}$.

Figure 4.1: $\mathsf{BlindIss}_{\mathsf{A}}(1^\kappa)$

Experiment $\mathsf{PtPrivacy}_{\mathsf{A}}(1^\kappa)$**:**
$(spar_{\mathtt{s}}, \tau_{\mathtt{s}}) \xleftarrow{\$} \mathsf{S}_{\mathtt{params}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa))$
$b \xleftarrow{\$} \{0,1\}$
$(ipk, sig, (c_i, o_i)_{i\in C}, R, \vec{a}, M, \mathsf{st}) \xleftarrow{\$} \mathsf{A}_{1,\mathtt{pres}}(spar_{\mathtt{s}}, \tau_{\mathtt{s}})$

If $b = 0$:	Else:
$spt \xleftarrow{\$} \mathsf{SignTokenGen}(ipk, sig, \vec{a}, R, (c_i, o_i)_{i\in C}, M)$	$spt \xleftarrow{\$} \mathsf{S}_{\mathtt{pres}}(ipk, \tau_{\mathtt{s}}, (a_i)_{i\in R}, (c_i)_{i\in C}, M)$

$b' \xleftarrow{\$} \mathsf{A}_{2,\mathtt{pres}}(spt, (c_i)_{i\in C}, \mathsf{st})$
Return 1 if and only if:
$b = b'$,
$\mathsf{ComOpenVf}(c_i, a_i, o_i) = \mathsf{accept}$ for all $i \in C$,
$\mathsf{KeyVf}(ipk) = \mathsf{accept}$, and
$\mathsf{SignVf}(sig, \vec{a}, ipk) = \mathsf{accept}$.

Figure 4.2: $\mathsf{PtPrivacy}_{\mathsf{A}}(1^\kappa)$

Definition 4.3.4 (User privacy)**.** *For every tuple of PPT adversaries* $(\mathsf{A}_{1,\mathtt{iss}}, \mathsf{A}_{2,\mathtt{iss}}, \mathsf{A}_{1,\mathtt{pres}}, \mathsf{A}_{2,\mathtt{pres}})$*, there exist PPT algorithms* $(\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}, \mathsf{S}_{\mathtt{pres}})$ *and a negligible function* ν *such that*

$$\{(spar_{\mathtt{s}} : spar_{\mathtt{s}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{s}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa))\} \sim \{spar_{\mathtt{s}} : (spar_{\mathtt{s}}, \tau_{\mathtt{s}}) \xleftarrow{\$} \mathsf{S}_{\mathtt{params}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa))\}\,,$$

and for the experiments in Figures 4.1 and 4.2 we have:

$$\Pr[\mathsf{BlindIss}_{\mathsf{A}}(1^\kappa) = 1] \le 1/2 + \nu(\kappa) \quad \textit{and}$$
$$\Pr[\mathsf{PtPrivacy}_{\mathsf{A}}(1^\kappa) = 1] \le 1/2 + \nu(\kappa)\ .$$

Here, the oracle $\mathcal{O}^{\mathtt{s.obtain}}$ used in the experiment $\mathsf{BlindIss}_{\mathsf{A}}$ allows the adversary to interact with "honest users". Being called with $(spar_{\mathtt{s}}, ipk, \vec{a}, (c_i, o_i)_{i\notin R})$ the oracle initiates an instance of the issuance protocol with the adversary acting as issuer.

When comparing this definition with the privacy definition of PABC (Definition 3.3.2), we see that here we have defined the experiments for signing (issuing of credentials) and for signature tokens (presentation tokens) separately. Furthermore, here we do not use a filter functionality $\mathcal{F}$. We made both these choices for simplicity. In fact, the filter functionality here would be rather trivial, it would just not pass the openings of the commitments and the non-revealed attributes to the simulator. In the experiments above, this is explicitly dropped from the inputs to the simulators. Merging the experiments into a single would complicate the experiments without reason. Also, separate experiments are useful for analyzing schemes that satisfy only one of the two, as described below with a weaker definition of privacy, where $\mathsf{BlindIss}_{\mathsf{A}}$ is the same in both definitions.

Experiment $\mathsf{Untraceability}_{\mathsf{A}}(1^\kappa)$**:**
$(spar_{\mathtt{s}}, \tau_{\mathtt{s}}) \xleftarrow{\$} \mathsf{S}_{\mathtt{params}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa))$
$b \xleftarrow{\$} \{0,1\}$

If $b = 0$:	Else:
$b' \xleftarrow{\$} \mathsf{A}_{1,\mathtt{pres}}^{\mathcal{O}^{\mathtt{s.user}}(\cdot)}(spar_{\mathtt{s}}, \tau_{\mathtt{s}})$	$b' \xleftarrow{\$} \mathsf{A}_{1,\mathtt{pres}}^{\mathcal{F}^{\mathtt{SigU}}(\cdot)\mid \mathsf{S}_{\mathtt{pres}}(\tau_s)}(spar_{\mathtt{s}}, \tau_{\mathtt{s}})$

Return 1 if and only if
$b = b'$

Figure 4.3: $\mathsf{Untraceability}_{\mathsf{A}}(1^\kappa)$

In the above definition, the adversary learns the signature itself, and must still not be able to distinguish real and fake presentations. However, many practically used schemes such as Microsoft's UProve do not achieve this goal. We therefore next describe an alternative, weaker notion user privacy, called *weak user privacy*, where apart from blind issuance, the adversary must only be unable to link presentations to specific issuance sessions, but might be able to link any two presentation tokens that were derived from the same signature. This property is referred to as *untraceability*.

The idea of the following definition is that the adversary, acting as the issuer, can ask an honest-user oracle to obtain an arbitrary number of signatures. Furthermore, receiving handles to those signatures (but not the signatures themselves) the adversary can request arbitrarily many presentation tokens for any of these signatures, where the presentation tokens will be honestly computed in the real world, but will be simulated only using the revealed attributes and the handle to the signature (but not the signature itself) in the ideal world. Giving the handle to the simulator is crucial, as in the real world presentation tokens might be linkable, and the simulator needs to know whether or not it has to simulate this linkability as well when receiving multiple presentation requests. The adversary wins if he can distinguish the two worlds.

Definition 4.3.5 (Weak User Privacy)**.** *For every tuple of PPT adversaries* $(\mathsf{A}_{1,\mathtt{iss}}, \mathsf{A}_{2,\mathtt{iss}}, \mathsf{A}_{1,\mathtt{pres}}, \mathsf{A}_{2,\mathtt{pres}})$*, there exist PPT algorithms* $(\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}, \mathsf{S}_{\mathtt{pres}})$ *and a negligible function* ν *such that for the experiments in Figures 4.1 and 4.3 we have:*

$$\Pr[\mathsf{BlindIss}_{\mathsf{A}}(1^\kappa) = 1] \leq 1/2 + \nu(\kappa) \quad \textit{and}$$
$$\Pr[\mathsf{Untraceability}_{\mathsf{A}}(1^\kappa) = 1] \leq 1/2 + \nu(\kappa) \ .$$

Honest User Oracle $\mathcal{O}^{\mathtt{s.user}}$**.** The user oracle gives the adversary access to honest users, which he can trigger to obtain signatures on inputs of his choice and request signature presentation tokens on inputs of his choice. The adversary does not get to see the actual signatures, but is only given a unique signature identifier id_{sig} by which he can refer to the signature. It provides the following interfaces:

- On input $(\mathtt{obtain}, ipk, \vec{a}, R)$ the oracle commits to all messages $(c_i, o_i) \xleftarrow{\$} \mathsf{Com}(a_i) \forall i \notin R$ and executes $sig/\perp \xleftarrow{\$} \mathcal{U}.\mathsf{Sign}(ipk, (c_j, o_j)_{j \notin R}, \vec{a})$, with the adversary playing the role of the issuer. The oracle adds $(id_{sig}, sig, \vec{a})$ to an initially empty list L, where id_{sig} is a unique pointer, and returns id_{sig} to the adversary.

  In the experiment, we write $sig(id_{sig})$ to refer to the (unique) signature sig such that $(id_{sig}, sig, \vec{a}) \in L$, or $sig(id_{sig}) = \perp$ if no such entry exists.

- On input $(\mathtt{pres}, ipk, id_{sig}, \vec{a}, R, (c_i, o_i)_{i \in C}, M)$, the oracle looks up $(id_{sig}, sig, \vec{a}) \in L$, returning $\perp$ if no tuple is found. It then computes $spt \xleftarrow{\$} \mathsf{SignTokenGen}(ipk, sig, \vec{a}, R, (c_i, o_i)_{i \in C}, M)$ and returns spt to the adversary.

Honest User Filter $\mathcal{F}^{\mathtt{SigU}}$. $\mathcal{F}^{\mathtt{SigU}}$ gives the adversary access to a simulator, which simulates the generation of a signature proof token from only the released attributes. It provides the following interfaces:

- On input $(\mathtt{obtain}, ipk, \vec{a}, R)$, $\mathcal{F}^{\mathtt{SigU}}$ commits to all messages $(c_i, o_i) \xleftarrow{\$} \mathsf{Com}(a_i)$ for all $i \notin R$ and executes $sig/\perp \xleftarrow{\$} \mathcal{U}.\mathsf{Sign}(ipk, (c_j, o_j)_{j \notin R}, \vec{a})$, with the adversary playing the role of the issuer. $\mathcal{F}^{\mathtt{SigU}}$ adds $(id_{sig}, \vec{a})$ to an initially empty list L, where id_{sig} is a unique pointer, and returns id_{sig} to the adversary.
- On input $(\mathtt{pres}, ipk, id_{sig}, \vec{a}, R, (c_i, o_i)_{i \in C}, M)$, the filter looks up $(id_{sig}, \vec{a}) \in L$, returning $\perp$ if no tuple is found. It then computes $spt \xleftarrow{\$} \mathsf{S}_{\mathtt{pres}}(ipk, \tau, id_{sig}, R, (a_i)_{i \in R}, (c_i)_{i \in C}, M)$ and returns spt.

4.3.3 Relation of Privacy Definitions

Lemma 4.3.6. *Definition 4.3.4 implies Definition 4.3.5, but not vice versa. More formally, every user private* PABS *scheme is also weakly user private, but there exist user weakly private* PABS *schemes that are not (fully) user private.*

Proof. Let be given a PABS satisfying (full) privacy, and let $(\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}, \mathsf{S}_{\mathtt{pres}})$ be the respective simulators. Define $\mathsf{S}'_{\mathtt{pres}}$ such that it just forwards all inputs except for id_{sig} to an internal copy of $\mathsf{S}_{\mathtt{pres}}$, and outputs whatever $\mathsf{S}_{\mathtt{pres}}$ returns. We then show that $(\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}, \mathsf{S}'_{\mathtt{pres}})$ is a valid simulator for weak privacy.

Blind issuance is clear. For untraceability, consider the following hybrid argument: in hybrid H_i, the first i presentation requests of an adversary A are answered by $\mathcal{O}^{\mathtt{s.user}}$, and all subsequent requests are answered by $\mathsf{S}'_{\mathtt{pres}}$. Let q be an integer such that with overwhelming probability A makes at most q presentation requests to its oracle. Note that q is polynomially bounded by the running time of A. If A can win Experiment Untraceability with some probability ε, then he can distinguish H_k and H_{k+1} with probability at least negligibly close to ε/q for some k.

Consider now the following adversary A', which internally executes a copy of A. Whenever A makes an $\mathtt{obtain}$-request to its oracle, A' (in both, A'_1 and A'_2) perfectly simulates the oracle, i.e., it plays the role of the honest user. Furthermore, A'_1 answers the first k $\mathtt{pres}$-requests with simulated presentation tokens. Upon receiving the $(k+1)^{\text{st}}$ presentation requests, A'_1 outputs the respective $(ipk, sig, C, R, \vec{a}, M, \mathsf{st})$, and replies to the internal copy of A whatever it receives in the experiment, i.e., either a honestly computed or a simulated presentation token. All further presentation requests are answered honestly by A'_2.

It can now be seen that A' can win Experiment PtPrivacy with the same probability as A wins in Experiment Untraceability. As the given scheme is user private by assumption, this probability is negligible, and the weak user privacy follows.

To see that the other direction does not hold, we refer to Section 7.4. The scheme there is weakly user private by Lemma 8.3.3, but not user private, as the signature presentation tokens reveal parts of the signature without re-randomizing them before. □

4.4 Revocation Schemes

Suppose a set of users, e.g., employees, that is granted access to some online resource. Then this set will often change over time. While adding users would be possible with the features presented so far, revoking access for specific users was not. In the following we thus define revocation for signature systems.

The following definition uses a blacklisting approach rather than whitelisting. We believe that for our scenario this is the natural choice, as in real-world applications the number of active signatures will usually be higher than that of revoked ones. Furthermore, whitelists

would require verifiers to update their local copy every time a new signature was issued, while for blacklisting different verifiers may obtain updates at different intervals, depending on their security policies, making it easier to realize offline applications.

4.4.1 Syntax

After having set up system parameters, a revocation authority generates a secret revocation key, together with some public revocation key and revocation information. Using its secret key, the authority can revoke attributes by updating the revocation information accordingly. Proving that an attribute has not yet been revoked is again done non-interactively: a user can generate a token showing that some commitment contains an unrevoked attribute. This token can later be publicly verified.

System Parameter Generation. The system parameters of a revocation system are generated as

$$spar_{\mathtt{r}} = (spar_{\mathtt{g}}, \mathcal{RS}, spar'_{\mathtt{r}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{r}}(spar_{\mathtt{g}}) \,,$$

where the input are global system parameters, and

- $\mathcal{RS}$ specifies the set of supported revocation handles, and
- $spar'_{\mathtt{r}}$ potentially specifies further revocation parameters.

The system parameters are input to any of the other algorithms of the scheme. However, for notational convenience, we will sometimes not make this explicit.

Revocation Setup. The revocation authority (possibly, but not necessarily the issuer) runs the revocation setup to obtain a secret key rsk, an associated public key rpk and a public revocation information RI:

$$(rsk, rpk, RI) \xleftarrow{\$} \mathsf{RKGen}(spar_{\mathtt{r}}) \,.$$

Attribute Revocation. An attribute can get revoked by a revocation authority by "excluding" a certain attribute a from the public revocation information RI.

$$RI' \xleftarrow{\$} \mathsf{Revoke}(rsk, RI, a) \,.$$

Revocation Token Generation. A user can generate a token proving that a certain attribute, committed to in c, has not been revoked before:

$$rt/\perp \xleftarrow{\$} \mathsf{RevTokenGen}(a, c, o, RI, rpk) \,.$$

In practice, a revocation presentation will always be tied to a signature presentation, to prove that a presentation signature is valid. The value of c in the former will therefore be one of the commitments from the presentation step.

Revocation Token Verification. A revocation token can be verified as follows:

$$\mathsf{accept/reject} \leftarrow \mathsf{RevTokenVf}(rt, c, RI, rpk) \,.$$

4.4.2 Security Definitions for Revocation Schemes

We require that whenever an honestly computed revocation information RI is used, an honest user is able to successfully generate valid tokens:

Definition 4.4.1 (Revocation correctness)**.** *There exists a negligible function ν such that the following holds for all ordered sets A of attributes and $a' \notin A$:*

$$\begin{aligned}\Pr[&\mathsf{RevTokenVf}(rt, c', RI, rpk) = \mathsf{reject} \ : \\ &spar_{\mathtt{g}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{g}}(1^\kappa), spar_{\mathtt{r}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{r}}(spar_{\mathtt{g}}), (c', o') \xleftarrow{\$} \mathsf{Com}(a'), \\ &(rsk, rpk, RI) \xleftarrow{\$} \mathsf{RKGen}(spar_{\mathtt{r}}), RI \xleftarrow{\$} \mathsf{Revoke}(rsk, RI, a)\ \forall a \in A, \\ &rt \xleftarrow{\$} \mathsf{RevTokenGen}(a', c', o', RI, rpk)] \leq \nu(\kappa)\,.\end{aligned}$$

Revocation soundness captures the following: To make the verifier accept, the user must know the attribute contained in the commitment it computes a revocation token for. Further, nobody except for the revocation authority can come up with a new valid revocation information, i.e., the revocation information is always authentic. Finally, this attribute must not have been revoked in an earlier revocation step.

Definition 4.4.2 (Revocation Soundness)**.** *A revocation scheme is* sound *if there exists an efficient algorithm $\mathsf{E}^{\mathtt{r}} = (\mathsf{E}_1^{\mathtt{r}}, \mathsf{E}_2^{\mathtt{r}})$, called the* extractor*, that satisfies the following properties:*

- $\mathsf{E}_1^{\mathtt{r}}$ *outputs parameters and trapdoors, such that the parameters are indistinguishable from correctly computed system parameters:*

  $$\{spar_{\mathtt{r}} : \ spar_{\mathtt{r}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{r}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa, \lambda))\} \ \sim \{spar_{\mathtt{r}} : \ (spar_{\mathtt{r}}, \tau_{\mathtt{r}}) \xleftarrow{\$} \mathsf{E}_1^{\mathtt{r}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa, \lambda))\}\,,$$

  and

- *for every efficient adversary* A *there exists a negligible function such that:*

$$\begin{aligned}\Pr\Big[&\mathsf{RevTokenVf}(rt, c, RI_{\mathsf{A}}, rpk) = \mathsf{accept} \ \wedge \ \Big(\mathsf{ComOpenVf}(c, a, o) = \mathsf{reject} \ \vee \\ &\nexists\, (RI_{\mathsf{A}}, epoch, a') \in \mathcal{L} \ \vee \ \exists\, (RI_{\mathsf{A}}, epoch, a'), (RI', epoch', a) \in \mathcal{L} \ : \ epoch' \leq epoch\Big) : \\ &(spar_{\mathtt{r}}, \tau_{\mathtt{r}}) \xleftarrow{\$} \mathsf{E}_1^{\mathtt{r}}(\mathsf{SPGen}_{\mathtt{g}}(1^\kappa, \lambda)), \ (rsk, rpk, RI_0) \xleftarrow{\$} \mathsf{RKGen}(spar_{\mathtt{r}}), \\ &(RI_{\mathsf{A}}, rt, c) \xleftarrow{\$} \mathsf{A}^{\mathcal{O}^{\mathtt{revoke}}}(rpk, RI_0, spar_{\mathtt{r}}), \ (a, o) \xleftarrow{\$} \mathsf{E}_2^{\mathtt{r}}(\tau_{\mathtt{r}}, rpk, rt, c)\Big] \leq \nu(\kappa).\end{aligned}$$

Here, the oracle $\mathcal{O}^{\mathtt{revoke}}$ simulates an honest revocation authority as follows, where initially $\mathcal{L} = \emptyset$ and $epoch = 0$:

- *On input $(\mathtt{revoke}, a)$ for some attribute $a \in \mathcal{AS}$, the oracle first revokes a by computing $RI' \xleftarrow{\$} \mathsf{Revoke}(rsk, RI, a)$. It then updates its internal state as $RI \leftarrow RI'$ and $epoch \leftarrow epoch + 1$. It then adds $(RI, epoch, a)$ to $\mathcal{L}$, and hands back the updated revocation information RI to the adversary.*

Revocation privacy ensures that no adversary can tell which of two unrevoked attributes a_0, a_1 underlies a revocation token. It is formally defined through the following experiment.

Definition 4.4.3 (Revocation Privacy)**.** *A revocation scheme is* private*, if for every efficient adversary* A *there exists a negligible function ν such that the following holds:*

$$\begin{aligned}\Pr\big[&b' = b \ \wedge \ rt_0 \neq \bot \ \wedge \ rt_1 \neq \bot \ : \ spar_{\mathtt{g}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{g}}(1^\kappa), \\ &spar_{\mathtt{r}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{r}}(spar_{\mathtt{g}}), (rpk, RI, a_0, a_1, \mathsf{st}) \xleftarrow{\$} \mathsf{A}(spar_{\mathtt{r}}), \\ &(c_i, o_i) \xleftarrow{\$} \mathsf{Com}(a_i), \ rt_i \xleftarrow{\$} \mathsf{RevTokenGen}(spar_{\mathtt{r}}, a_i, c_i, o_i, RI, rpk), \ i = 0, 1, \\ &b \xleftarrow{\$} \{0,1\}, b' \xleftarrow{\$} \mathsf{A}(c_b, rt_b, \mathsf{st})\big] \leq \frac{1}{2} + \nu(\kappa)\,.\end{aligned}$$

4.5 Pseudonyms

Users can be known under different pseudonyms to different credential issuers and verifiers, which are all mutually unlinkable.

4.5.1 Syntax

System Parameter Generation. The system parameters of a pseudonym system are generated as

$$spar_{\mathsf{p}} = (spar_{\mathsf{g}}, spar'_{\mathsf{p}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{p}}(spar_{\mathsf{g}}) \,,$$

where the input are global system parameters, and

- $spar'_{\mathsf{p}}$ potentially specifies further pseudonym parameters.

The system parameters are input to any of the other algorithms of the scheme. However, for notational convenience, we will sometimes not make this explicit.

User key generation. A user generates his secret key as $usk \xleftarrow{\$} \mathsf{UKGen}(spar_{\mathsf{p}})$.

Pseudonym generation. A pseudonym nym for a given user secret key and a $scope \in \{0,1\}^*$ can be computed deterministically as:

$$nym \leftarrow \mathsf{NymGen}(usk, scope) \,.$$

Pseudonym presentation. On input a user's secret key usk, a commitment c to usk with opening information o, and a scope string $scope \in \{0,1\}^*$, the pseudonym presentation algorithm generates a pseudonym nym with a proof π:

$$(nym, \pi) \xleftarrow{\$} \mathsf{NymPres}(usk, c, o, scope) \,.$$

Pseudonym verification. A pseudonym nym and pseudonym proof π are verified for a commitment c and scope $scope$ as

$$\mathsf{accept/reject} \leftarrow \mathsf{NymVf}(spar_{\mathsf{p}}, c, scope, nym, \pi) \,.$$

4.5.2 Security Definitions for Pseudonyms

In a pseudonym system, users can derive pseudonyms from their secret key and arbitrary scope strings, and prove that they indeed know the secret key used to compute this pseudonym. Correctness and key extractability guarantee that honest users can perform such proofs, and that the verifier is guaranteed that the user indeed knows the used secret key. Collision resistance then guarantees that for each scope string, any two users will have different pseudonyms with overwhelming probability. Finally, unlinkability guarantees that users cannot be linked across scopes.

Definition 4.5.1 (Correctness)**.** *There exists a negligible function ν such for all $scope \in \{0,1\}^*$ it holds that:*

$$\Pr\Big[\mathsf{NymVf}(spar_{\mathsf{p}}, c, scope, nym, \pi) = \mathsf{reject} \ \vee \ nym \neq \mathsf{NymGen}(usk, scope) \ : \\ spar_{\mathsf{g}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{g}}(1^\kappa), spar_{\mathsf{p}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{p}}(spar_{\mathsf{g}}), usk \xleftarrow{\$} \mathsf{UKGen}(spar_{\mathsf{p}}), \\ (c, o) \xleftarrow{\$} \mathsf{Com}(usk), (nym, \pi) \xleftarrow{\$} \mathsf{NymPres}(usk, c, o, scope)\Big] \leq \nu(\kappa) \,.$$

Experiment $\mathsf{LinkNyms}_{\mathsf{A}}(1^\kappa)$**:**
$spar_{\mathsf{g}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{g}}(1^\kappa)$
$spar_{\mathsf{p}} \xleftarrow{\$} \mathsf{SPGen}_{\mathsf{s}}(spar_{\mathsf{g}})$
$usk_0, usk_1 \xleftarrow{\$} \mathsf{UKGen}(spar_{\mathsf{p}})$
$b \xleftarrow{\$} \{0,1\}$
$(scope^*, \mathsf{st}) \xleftarrow{\$} \mathsf{A}_1^{\mathcal{O}_0(usk_0,\cdot),\mathcal{O}_1(usk_1,\cdot)}(spar_{\mathsf{p}})$
$(c^*, o^*) \xleftarrow{\$} \mathsf{Com}(usk_b)$
$(nym^*, \pi^*_{nym}) \xleftarrow{\$} \mathsf{NymPres}(usk_b, c^*, o^*, scope^*)$
$b' \xleftarrow{\$} \mathsf{A}_2^{\mathcal{O}_0(usk_0,\cdot),\mathcal{O}_1(usk_1,\cdot)}(nym^*, \pi^*_{nym}, c^*)$
Return 1 if and only if:
$b = b'$,
$scope^* \notin \mathcal{L}_{\mathtt{scope}}$, and
$c^* \neq c \ \forall (usk, c, o) \in \mathcal{L}_{\mathtt{com}}$.

Figure 4.4: $\mathsf{LinkNyms}_{\mathsf{A}}(1^\kappa)$

Definition 4.5.2 (Key Extractability)**.** *A pseudonym system is* key extractable*, if* $(\mathsf{SPGen}_{\mathsf{p}}, \mathsf{NymGen}, \mathsf{NymVf})$ *is a non-interactive proof of knowledge for the following relation, cf. § 2.2.1:*

$$((c, scope, nym), (usk, o)) \in R :\Leftrightarrow \mathsf{NymGen}(usk, scope) = nym \wedge \mathsf{ComOpenVf}(usk, c, o) = \mathsf{accept}\,.$$

Definition 4.5.3 (Collision resistance)**.** *A pseudonym system is* collision resistant*, if for every PPT algorithm* A *there is a negligible function* ν *such that:*

$$\Pr[\mathsf{NymGen}(usk_0, scope) = \mathsf{NymGen}(usk_1, scope) : (usk_0, usk_1, scope) \xleftarrow{\$} \mathsf{A}(\mathsf{SPGen}_{\mathsf{p}}(\mathsf{SPGen}_{\mathsf{g}}(1^\kappa)))] \leq \nu(\kappa)\,.$$

Definition 4.5.4 (Pseudonym unlinkability)**.** *We define pseudonym unlinkability as a game between the adversary and two oracles* $\mathcal{O}_0(usk_0, \cdot), \mathcal{O}_1(usk_1, \cdot)$ *simulating honest users as follows, where the oracles share initially empty lists* $\mathcal{L}_{\mathtt{scope}}$ *and* $\mathcal{L}_{\mathtt{com}}$*:*

- *On input* $(c, scope)$*, oracle* $\mathcal{O}_i$ *checks whether there exists* $(usk_i, c, o) \in \mathcal{L}_{\mathtt{com}}$ *if* $c \neq \perp$*. If it does, it returns* $(nym, \pi) \xleftarrow{\$} \mathsf{NymPres}(usk_i, c, o, scope)$ *to* A *and aborts otherwise. If* $c = \perp$*, the oracle computes* $(c', o') \xleftarrow{\$} \mathsf{Com}(usk_i)$*, adds* (usk_i, c', o') *to* $\mathcal{L}_{\mathtt{com}}$*, sets* $(nym, \pi_{nym}) \xleftarrow{\$} \mathsf{NymPres}(usk_i, c', o', scope)$ *and returns* (nym, π, c') *to* A*. If no abort occurred, the oracle adds scope to* $\mathcal{L}_{\mathtt{scope}}$*.*

We now require that for every PPT adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *we have that for some negligible function* ν *it holds that:*

$$\Pr[\mathsf{LinkNyms}_{\mathsf{A}}(1^\kappa) = 1] \leq 1/2 + \nu(\kappa)\,.$$

Chapter 5

Generic Construction of PABCs

Our generic construction of a PABC system uses the following components:

- A global setup algorithm $\mathsf{SPGen}_\mathtt{g}$,
- a commitment scheme

$$(\mathsf{SPGen}_\mathtt{c}, \mathsf{ComKGen}, \mathsf{Com}, \mathsf{ComPf}, \mathsf{ComOpenVf})\,,$$

- a PABS scheme

$$(\mathsf{SPGen}_\mathtt{s}, \mathsf{IKGen}, \mathsf{KeyVf}, \mathcal{U}.\mathsf{Sign}, \mathcal{I}.\mathsf{Sign}, \mathsf{SigVf}, \mathsf{SignTokenGen}, \mathsf{SignTokenVf})$$

- a revocation scheme

$$(\mathsf{SPGen}_\mathtt{r}, \mathsf{Revoke}, \mathsf{RevTokenGen}, \mathsf{RevTokenVf})\,, \quad \text{and}$$

- a pseudonym scheme

$$(\mathsf{SPGen}_\mathtt{p}, \mathsf{UKGen}, \mathsf{NymGen}, \mathsf{NymPres}, \mathsf{NymVf})\,.$$

5.1 Intuition

We next very briefly describe the intuition underlying our generic construction.

The idea is to use the underlying pseudonym and revocation schemes unchanged to obtain the according properties for the PABC-system. Issuance and presentation are realized via the given PABS-scheme. However, instead of just signing the attributes, the issuer additionally signs the user secret key and the revocation handle whenever applicable. Similarly, whenever a user computes a presentation token for a set of credentials, it proves knowledge of the according signature on the contained attributes, the revocation handle, and the user secret key. Thereby, the user secret key is always treated as an unrevealed attribute.

The seemingly independent components are linked together via the commitments being used. For instance, the same commitment/opening pair is used to generate revocation tokens (RevTokenGen) and signature presentation tokens (SignTokenGen), guaranteeing that indeed the revocation handle contained in the credential was also shown to be unrevoked.

5.2 Formal Description of the Construction

In the following, let eq be a function mapping an attribute index (i, j) to its equivalence class as induced by the equivalence relation E, i.e.,

$$\mathrm{eq}(i,j) = \{(i,j)\} \cup \{(i',j') : ((i,j),(i',j')) \in E\}$$

$(c_{usk}, o_{usk}) \xleftarrow{\$} \mathsf{Com}(usk)$
if $scope = \varepsilon$:
  $(nym, \pi_{nym}) = (\varepsilon, \varepsilon)$
else:
  $(nym, \pi_{nym}) \xleftarrow{\$} \mathsf{NymPres}(usk, c_{usk}, o_{usk}, scope)$
$(c_{\overline{e}}, o_{\overline{e}}) \xleftarrow{\$} \mathsf{Com}(a_{\overline{e}})\ \forall \overline{e} \in \overline{E}$
for $i = 1, \ldots, k$ do:
  if $cred_i$ is revocable:
    $(c_{rh,i}, o_{rh,i}) \xleftarrow{\$} \mathsf{Com}(rh_i)$
    $rt_i \xleftarrow{\$} \mathsf{RevTokenGen}(rh_i, c_{rh,i}, o_{rh,i}, RI_i, rpk_i)$
  else:
    $(c_{rh,i}, o_{rh,i}) \leftarrow (\varepsilon, \varepsilon)$
$\hat{M}' = \hat{M} \| scope \| (ipk_i, (a_{i,j})_{j \in R_i})_{i=1}^{k} \| E$
for $i = 1, \ldots, k$ do:
  $spt_i \xleftarrow{\$} \mathsf{SignTokenGen}(ipk'_i, sig_i, ((a_{i,j})_{j=1}^{n_i}, usk, rh_i), R_i, ((c_{\mathrm{eq}(i,j)}, o_{\mathrm{eq}(i,j)})_{j \in E_i}, c_{usk}, o_{usk}, c_{rh,i}, o_{rh,i}), \hat{M}')$
$pt = (c_{usk}, \pi_{nym}, (c_{\overline{e}})_{\overline{e} \in \overline{E}}, (c_{rh,i}, rt_i, spt_i)_{i=1}^{k})$
if $rt_i \neq \perp$ and $spt_i \neq \perp$ for $i = 1, \ldots, k$:
  Output (nym, pt)
else :
  Output $(\perp, \perp)$

Figure 5.1: $\mathsf{AuxPresent}(usk, scope, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, \hat{M})$

and let $\overline{E}$ be the set of equivalence classes defined by E. By $(a_{\overline{e}})_{\overline{e} \in \overline{E}}$ we denote the common attribute values such that $\mathrm{eq}(i,j) = \overline{e} \Rightarrow a_{i,j} = a_{\overline{e}}$. Finally, let $E_i = \{j : ((i,j),(i',j')) \in E\}$.

System parameter generation. The system parameters for the PABC scheme are generated by generating:

- global system parameters $spar_{\mathtt{g}} = (1^{\kappa}, \ell, \ell', spar_{\mathtt{c}}, ck, spar'_{\mathtt{g}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{g}}(1^{\kappa})$,
- signature system parameters $spar_{\mathtt{s}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{s}}(spar_{\mathtt{g}})$,
- revocation system parameters $spar_{\mathtt{r}} = (spar_{\mathtt{g}}, spar'_{\mathtt{r}}) \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{r}}(spar_{\mathtt{g}})$, and
- pseudonym system parameters $spar_{\mathtt{r}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{p}}(spar_{\mathtt{g}})$,

and outputting:

$$spar = (spar_{\mathtt{g}}, spar_{\mathtt{s}}, spar_{\mathtt{r}}, spar_{\mathtt{p}})\,,$$

We assume that the algorithms of all building blocks take their respective parameters as implicit inputs.

User key generation. Users generate their secret keys as $usk \xleftarrow{\$} \mathsf{UKGen}(1^{\kappa})$.

Issuer key generation. Issuers generate a signature key pair $(ipk', isk') \xleftarrow{\$} \mathsf{IKGen}(spar_{\mathtt{s}})$ and revocation keys $(rsk, rpk, RI) \xleftarrow{\$} \mathsf{RKGen}(spar_{\mathtt{r}})$.

The public key is $ipk = (ipk', rpk)$, the secret key $isk = (isk', rsk)$, and the initial revocation information RI.

Presentation. The algorithm Present takes $(usk, scope, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M)$ as inputs, and outputs whatever AuxPresent specified in Figure 5.1 outputs, where $ipk_i = (ipk'_i, rpk_i)$ and $cred_i = (sig_i, rh_i)$ are as before, and $\hat{M} = \mathtt{pres} \| M$.

Presentation verification. On inputs $(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k}, E, M)$, Verify outputs accept if and only if AuxVerify described in Figure 5.2 outputs accept.

$\hat{M}' = \hat{M} \| scope \| (ipk_i, (a_{i,j})_{j \in R_i})_{i=1}^{k} \| E$
if $scope \neq \varepsilon \wedge \mathsf{NymVf}(c_{usk}, scope, nym, \pi_{nym}) = \mathsf{reject}$:
  Output reject for $i = 1, \ldots, k$ do:
  if $\mathsf{RevTokenVf}(rt_i, c_{rh,i}, RI_i, rpk_i) = \mathsf{reject}$ or
    $\mathsf{SignTokenVf}(ipk_i, spt_i, (a_{i,j})_{j \in R_i}, ((c_{\mathrm{eq}(i,j)})_{j \in E_i}, c_{usk}, c_{rh,i})_{i=1}^{k}, \hat{M}') = \mathsf{reject}$:
      Output reject
Output accept

Figure 5.2: $\mathsf{AuxVerify}(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k}, E, \hat{M})$

$(nym, pt) \xleftarrow{\$} \mathsf{AuxPresent}(usk, scope, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, \mathtt{iss} \| M)$,
  thereby saving the used $(c_{\overline{e}}, o_{\overline{e}})_{\overline{e} \in \overline{E}}$
$(c_j, o_j) \xleftarrow{\$} \mathsf{Com}(a_{k+1,j})\ \forall j \notin R_{k+1} \cup E_{k+1}$
$\pi_j \xleftarrow{\$} \mathsf{ComPf}(c_{\mathrm{eq}(k+1,j)}, o_{\mathrm{eq}(k+1,j)}, a_{k+1,j})\ \forall j \in E_{k+1}$
$\pi_j \xleftarrow{\$} \mathsf{ComPf}(c_j, o_j, a_{k+1,j})\ \forall j \notin R_{k+1} \cup E_{k+1}$
$\pi_{usk} \xleftarrow{\$} \mathsf{ComPf}(c_{usk}, o_{usk}, usk)$
$pit = (pt, rh_{k+1}, (c_j)_{j \notin R_{k+1} \cup E_{k+1}}, \pi_{usk}, (\pi_j)_{j \notin R_{k+1}})$
$sit = ((c_{\mathrm{eq}(k+1,j)}, o_{\mathrm{eq}(k+1,j)})_{j \in E_{k+1}}, (c_j, o_j)_{j \notin R_{k+1} \cup E_{k+1}}, c_{usk}, o_{usk}, ipk'_{k+1}, (a_{k+1,j})_{j=1}^{n_{k+1}}, usk, rh_{k+1})$
Output (pit, sit, nym)

Figure 5.3: $\mathsf{ITGen}(usk, scope, rh_{k+1}, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k+1}, E, M)$

Issuance token generation. An issuance token for inputs $(usk, scope, rh_{k+1}, (ipk_i, RI_i, cred_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k+1}, E, M)$ is generated as specified in Figure 5.3.

Issuance token verification. Given an issuance token $pit = (pt, rh_{k+1}, (c_{k+1,j}, \pi_{k+1,j})_{j \notin R_{k+1}}, \pi_{usk})$, the verifier returns accept iff:

$$\mathsf{AuxVerify}\big(nym, pt, scope, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k}, E, \mathtt{iss} \| M\big) = \mathsf{accept}.$$

Issuance. To get a credential issued, the user runs

$$\mathcal{U}.\mathsf{Sign}(ipk'_{k+1}, ((c_{\mathrm{eq}(k+1,j)}, o_{\mathrm{eq}(k+1,j)})_{j \in E_{k+1}}, (c_j, o_j)_{j \notin R_{k+1} \cup E_{k+1}}, c_{usk}, o_{usk}), ((a_{k+1,j})_{j=1}^{n_{k+1}}, usk, rh_{k+1}))\,,$$

obtaining all the required inputs from sit, while the issuer runs

$$\mathcal{I}.\mathsf{Sign}(isk', ((a_i)_{i \in R_{k+1}}, rh_{k+1}), ((c_{\mathrm{eq}(k+1,j)}, \pi_j)_{j \in E_{k+1}}, (c_j, \pi_j)_{j \notin R_{k+1} \cup E_{k+1}}, (c_{usk}, \pi_{usk})))\,,$$

obtaining its inputs from isk and pit. When the user's protocol returns sig, the user's issuance algorithm returns $cred = (sig, rh_{k+1})$.

Revocation. On input an issuer secret key $isk = (isk', rsk)$, a revocation information RI and a revocation handle rh, the revocation algorithm returns $RI' \xleftarrow{\$} \mathsf{Revoke}(rsk, RI, rh)$.

Chapter 6

Security of the Generic Construction

In the following we prove the security properties of the generic construction.

6.1 Correctness

Theorem 6.1.1. *The constructed scheme is correct, if all building blocks are correct.*

This claim is straightforward to verify, and thus the formal proof is omitted.

6.2 Pseudonym Collision-Resistance

Theorem 6.2.1. *The pseudonyms of our generic construction are collision resistant and deterministic, if the underlying pseudonym system is collision resistant.*

Proof. The claim follows directly from the way pseudonyms are computed in Present and ITGen, using that NymGen in § 4.5 is deterministic by definition. □

6.3 Unforgeability

Theorem 6.3.1. *Our generic construction is unforgeable if the revocation scheme is sound, the pseudonym scheme is key-extractable, the commitment scheme is opening-extractable and binding, and the* PABS *scheme is unforgeable.*

Proof. We prove the above theorem through a sequence of games played with the unforgeability adversary A. The first game Game 0 is the normal security experiment as defined in § 3.3.3. The last game is such that, given an adversary A who has non-negligible success in winning it, we can construct a polynomial-time adversary B against the unforgeability of the signature scheme, cf. Definition 4.3.2. For each hop between intermediate games, we show that they are indistinguishable from the adversary's view under appropriate assumptions.

Game 0: This is the original unforgeability experiment as defined in § 3.3.3.

Game 1: This game relies on the revocation soundness property (see Definition 4.4.2) to generate simulated system parameters $(spar_{\mathtt{r}}, \tau_{\mathtt{r}}) \xleftarrow{\$} \mathsf{E}_1^{\mathtt{r}}(spar_{\mathtt{g}})$ and extract the revocation handles and opening information $(rh_i^{\mathtt{r}}, o_{rh,i}^{\mathtt{r}}) \xleftarrow{\$} \mathsf{E}_2^{\mathtt{r}}(spar_{\mathtt{r}}, \tau_{\mathtt{r}}, c_{rh,i}, rt_i)$ from all commitments $c_{rh,i}$ in presentation and issuance tokens produced by the adversary in the sets $\mathcal{F}$ and $\mathcal{IT}$. Let RI_i be the revocation information used in each token and let $epoch_i$ be the highest value so that $(ipk_i, RI_i, epoch_i) \in \mathcal{RI}$. The game aborts if $\mathsf{ComOpenVf}(c_{rh,i}, rh_i^{\mathtt{r}}, o_{rh,i}^{\mathtt{r}}) = \mathsf{reject}$ or if $rh_i^{\mathtt{r}}$ was revoked at $epoch_i$, i.e., if there exists $(ipk_i, rh_i, epoch_i') \in \mathcal{RRH}$ such that $epoch_i' \leq epoch_i$.

Any adversary A that causes this to happen can be used to build an adversary B that breaks the revocation soundness by, on input $rpk, RI, spar_{\mathtt{r}}$, using rpk and RI as part of the public key and revocation information of a random honest issuer I^*. It runs the code of Game 1, but responds to revocation queries for issuer I^* using its own $\mathcal{O}^{\mathtt{revoke}}$ oracle. At the end of the game, B outputs a commitment $c_{rh,i}$ and revocation token rt_i from a random presentation or issuance token and for a random i such that $ipk_i = ipk^*_{I^*}$.

As simulated and real parameters are indistinguishable, this game is indistinguishable from the previous one, as long as no abort happens. However, by the above arguments, the latter only happens with negligible probability.

Game 2: This game relies on the key extractability of the pseudonym scheme (see Definition 4.5.2) to generate the pseudonym parameters as $(spar_{\mathtt{p}}, \tau_{\mathtt{p}}) \xleftarrow{\$} \mathsf{E}^{\mathsf{p}}_1(spar_{\mathtt{g}})$ and extract the user secret keys and opening information $(usk^{\mathsf{p}}, o^{\mathsf{p}}_{usk}) \xleftarrow{\$} \mathsf{E}^{\mathsf{p}}_2(spar_{\mathtt{p}}, \tau_{\mathtt{p}}, c_{usk}, scope, nym, \pi)$ from all presentation and issuance tokens in $\mathcal{F} \cup \mathcal{IT}$ for which $scope \neq \varepsilon$. Note that the key extractability property guarantees that $\mathsf{NymGen}(usk, scope) = nym$ and $\mathsf{ComOpenVf}(usk^{\mathsf{p}}, c_{usk}, o^{\mathsf{p}}_{usk}) = \mathsf{accept}$.

As simulated and real parameters are indistinguishable, this game is indistinguishable from the previous one.

Game 3: This game uses the opening extractability of the commitment scheme (see Definition 4.2.4) to generate the commitment parameters as $(spar_{\mathtt{c}}, \tau_{\mathtt{c}}) \xleftarrow{\$} \mathsf{E}^{\mathsf{c}}_1(1^\kappa, \ell')$.

As simulated and real parameters are indistinguishable, this game is indistinguishable from the previous one.

Game 4: This game uses the unforgeability of the PABS scheme as per Definition 4.3.2 to extract signatures from all presentation and issuance tokens produced by the adversary. More specifically, it generates $(spar_{\mathtt{s}}, \tau_{\mathtt{s}}) \xleftarrow{\$} \mathsf{E}^{\mathsf{s}}_1(spar_{\mathtt{g}})$ and extracts $(sig_i, ((\alpha^{\mathsf{s}}_{i,j})_{j \notin R_i}, usk^{\mathsf{s}}_i, rh^{\mathsf{s}}_i), ((o^{\mathsf{s}}_{i,j})_{j \in E_i}, o^{\mathsf{s}}_{usk,i}, o^{\mathsf{s}}_{rh,i})) \xleftarrow{\$} \mathsf{E}_2(\tau_{\mathtt{s}}, ipk_i, (a_{i,j})_{j \in R_i}, ((c_{\mathrm{eq}(i,j)})_{j \in E_i}, c_{usk}, c_{rh,i}), spt_i)$ for all presentation and issuance tokens in $\mathcal{F} \cup \mathcal{IT}$ and for all i where $ipk_i \in IK^*$. Let $\alpha^{\mathsf{s}}_{i,j} \leftarrow a_{i,j}$ for all $j \in R_i$. If for some of the extracted signatures, attributes, and opening information $\mathsf{SigVf}(sig_i, (\alpha^{\mathsf{s}}_{i,1}, \ldots, \alpha^{\mathsf{s}}_{i,n_i}, usk^{\mathsf{s}}_i, rh^{\mathsf{s}}_i), ipk) = \mathsf{reject}$, $\mathsf{ComOpenVf}(\alpha^{\mathsf{s}}_{i,j}, c_{\mathrm{eq}(i,j)}, o^{\mathsf{s}}_{i,j}) = \mathsf{reject}$ for some $j \in E_i$, $\mathsf{ComOpenVf}(usk^{\mathsf{s}}_i, c_{usk}, o^{\mathsf{s}}_{usk,i}) = \mathsf{reject}$, or $\mathsf{ComOpenVf}(rh^{\mathsf{s}}_i, c_{rh,i}, o^{\mathsf{s}}_{rh,i}) = \mathsf{reject}$, then the game aborts.

An adversary A that causes the game to abort can be used to build a forger B for the PABS scheme as follows. On input $spar_{\mathtt{s}}, ipk$, algorithm B generates parameters for the pseudonym and revocation scheme and guesses a random issuer index $I^* \xleftarrow{\$} \{1, \ldots, n_{\mathrm{I}}\}$. It generates fresh user secret keys $(usk^*_i)_{i=1}^{n_{\mathrm{U}}}$ for all users and fresh revocation keys $(rsk_i, rpk_i, RI_i)_{i=1}^{n_{\mathrm{I}}}$ for all issuers. It also generates fresh issuer keys $(ipk^*_i, isk^*_i)^{n_{\mathrm{I}}}_{i=1, i \neq I^*}$ for all issuers $i \neq I^*$, but sets $ipk^*_{I^*} \leftarrow ipk$. It answers A's oracle queries as follows:

- $\mathcal{O}^{\mathtt{issuer}}(\mathtt{issue}, nym, pit, scope, rh, (ipk_i, RI_i, (a_{i,j})_{j \in R_i})_{i=1}^{k+1}, E, M)$: Algorithm B first verifies the issuance token pit and, if $ipk_{k+1} \neq ipk^*_{I^*}$, performs a normal issuance protocol since it knows all required keys. If $ipk_{k+1} = ipk^*_{I^*}$, B queries its own oracle $\mathcal{O}^{\mathtt{issuer}}((a_{k+1,j})_{j \in R_{k+1}}, (c_{\mathrm{eq}(k+1,j)}, \pi_{\mathrm{eq}(k+1,j)})_{j \in E_{k+1}}, (c_j, \pi_j)_{j \notin R_{k+1} \cup E_{k+1}})$ and forwards all forthcoming messages back and forth between A and the oracle.
- $\mathcal{O}^{\mathtt{issuer}}(\mathtt{revoke}, I, rh)$: As in the normal game.
- $\mathcal{O}^{\mathtt{user}}(\mathtt{present}, U, scope, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k}, E, M)$: It follows the steps of $\mathsf{Present}$, but for those i where $ipk_i = ipk^*_{I^*}$ B generates spt_i by querying its own $\mathcal{O}^{\mathtt{user}}$ oracle on input $(\mathtt{token}, cid_i, R_i, ((c_{\mathrm{eq}(i,j)}, o_{\mathrm{eq}(i,j)})_{j \in E_i}, (c_{usk}, o_{usk}), (c_{rh,i}, o_{rh,i})), M')$, where $M' = \mathtt{presentation} \| M \| scope \| (ipk_i, (a_{i,j})_{j \in R_i})_{i=1}^{k} \| E$.

- $\mathcal{O}^{\mathtt{user}}(\mathtt{obtain}, U, scope, rh, (ipk_i, RI_i, cid_i, R_i)_{i=1}^{k+1}, E, M, (a_{k+1,j})_{j=1}^{n_{k+1}})$: B generates the issuance token similarly as above, using its own $\mathcal{O}^{\mathtt{user}}$ oracle to generate the parts spt_i whenever $ipk_i = ipk^*_{I^*}$. If $ipk_{k+1} \neq ipk^*_{I^*}$, then it proceeds with issuance as usual. If $ipk_{k+1} = ipk^*_{I^*}$, then it queries its own oracle $\mathcal{O}^{\mathtt{user}}(\mathtt{obtain}, (\vec{a}, usk^*_U, rh))$ and relays the subsequent responses between A and the oracle.

Eventually, when A outputs its set of forgeries $\mathcal{FT}$, B chooses a random index i from a random presentation or issuance token from $\mathcal{FT} \cup \mathcal{IT}$ and outputs the tuple $((a_{i,j})_{j\in R_i}, ((c_{\mathrm{eq}(i,j)}, o_{\mathrm{eq}(i,j)})_{j\in E_i}, (c_{usk}, o_{usk}), (c_{rh,i}, o_{rh,i})), spt_i)$. One can see that A's view is exactly as in a real game. If A has a non-negligible chance of causing Game 3 to abort, however, then the above output has a non-negligible chance of making B win the unforgeability game.

Game 5: This game checks that for all commitments that are extracted multiple times during the game, the same value is always extracted. If at some point different values are extracted for the same commitment, the game aborts; it is clear that any such event directly leads to an attacker breaking the binding property of the commitment scheme. In particular, for all commitments c_{usk} extracted in Game 2 and Game 4, it checks that $usk^{\mathtt{p}} = usk_i^{sig}$ for all $i = 1, \ldots, k$; for all commitments $c_{rh,i}$ extracted in Game 1 and Game 4 that $rh_i^{\mathtt{r}} = rh_i^{\mathtt{s}}$ for all $i = 1, \ldots, k$; for all commitments $c_{\overline{e}}$ extracted in Game 4 that $\alpha_{i,j}^{\mathtt{s}} = \alpha_{i',j'}^{\mathtt{s}}$ for all $((i,j),(i',j')) \in E$. Since they are now all equal, we denote the extracted values as usk, rh_i, and $a_{\overline{e}}$, respectively.

We now show that any adversary A winning Game 5 can be used to build a forging adversary B against the PABS scheme as per Definition 4.3.2. Algorithm B simulates A's input and oracles exactly as in the reduction of Game 4. When A outputs its set of tokens $\mathcal{F}$, consider the set $CRED$ containing all tuples $(usk, ipk^*_I, rh, (\alpha_1, \ldots, \alpha_n))$ that B extracted from the presentation and issuance tokens in $\mathcal{F}$ and $\mathcal{IT}$, and let USK be the set of all extracted user secret keys. To win the PABC unforgeability game, there cannot exist sets $CRED$ and USK satisfying Conditions 1 and 2. We first argue that the sets $CRED$ and USK we just constructed satisfy Condition 2a, so that they must not satisfy Condition 1. We then show how B can use sets violating Condition 1 to win the PABS unforgeability game.

One can see that Condition 2a, requiring that all credentials in the same token are bound to the same user secret, would have caused B to abort due to the changes in Game 5. Likewise, Condition 2b is satisfied by Game 2, Condition 2c is by Game 4, Condition 2d by Game 5, and Condition 2e by Game 1. Since $CRED$ and USK thereby satisfy Condition 1, they must violate Condition 2. That is, there must exist a revocation handle rh and an honest issuer public key ipk such that there are more tuples $(usk, ipk, rh, \vec{a}) \in CRED$ than tuples $(ipk, rh) \in \mathcal{IRH}$. If $ipk \neq ipk^*_{I^*}$, then B aborts. Otherwise, since B's oracle $\mathcal{O}^{\mathtt{issuer}}$ adds at most one vector $(\vec{a}, usk, rh)$ to $\mathcal{L}_{\mathrm{iss}}$ in each call to $\mathcal{O}^{\mathtt{issuer}}$, by a simple counting argument there must be at least one tuple $(usk, ipk, rh, \vec{a}) \in CRED$ such that $(\vec{a}, usk, rh) \notin \mathcal{L}_{\mathrm{iss}}$. Adversary B then selects one of these tuples at random and uses the corresponding extracted signature sig to generate a signature token $spt \xleftarrow{\$} \mathsf{SignTokenGen}(ipk^* I^*, sig, (\vec{a}, usk, rh), \{1, \ldots, n+2\}, \emptyset, M)$ for an arbitrary message M that B never queried to its $\mathcal{O}^{\mathtt{user}}$ oracle. This tuple definitely satisfies the condition that $(\vec{a}, usk, rh), \varepsilon, M) \notin \mathcal{L}_{\mathrm{spt}}$ and with probability at least $1/\sharp CRED$ also satisfies $(\vec{a}, usk, rh) \notin \mathcal{L}_{\mathrm{iss}}$. □

6.4 Simulatable Privacy

We here prove the privacy property of the generic construction in § 5. There are two proofs, one for privacy and one for weak privacy. Which of these the generic construction fulfills depends on security property of the underlying PABS scheme.

Theorem 6.4.1 (Simulatable Privacy)**.** *The* PABC*-system resulting from the generic construction presented in § 5 is private in the sense of Definition 3.3.2, if the underlying commitment scheme is hiding, the* PABS*-scheme is user private according to Definition 4.3.4, the revocation scheme is private according to Definition 4.4.3, and the pseudonym system is unlinkable according to Definition 4.5.4.*

Proof. We prove the statement through a sequence of games played with the adversary, such that each game is computationally indistinguishable from the previous game. The first game is game of experiment t $\mathsf{Privacy}_{\mathsf{A}}(1^{\kappa}, n_{\mathrm{U}})$ for $b = 0$, where the adversary is interacting with honest user oracles, and the last game defines simulators for the same experiment with $b = 1$.

Game 0: This game is the game of experiment $\mathsf{Privacy}_{\mathsf{A}}(1^{\kappa}, n_{\mathrm{U}})$ for $b = 0$, i.e., the case where the adversary is interacting with the honest user oracle.

Game 1a-1c: These games rely on the user privacy of the PABS-scheme (Definition 4.3.4) to simulate the PABS-signatures without knowledge of the hidden attributes.

Game 1a: This game uses simulated system parameters for the PABS-scheme. Instead of executing SPGen, this game execute S_1 where S_1 is defined as computing all system parameters like SPGen, except instead of computing $spar_{\mathtt{s}} \xleftarrow{\$} \mathsf{SPGen}_{\mathtt{s}}(spar_{\mathtt{g}})$, S_1 computes $(spar_{\mathtt{s}}, \tau) \xleftarrow{\$} \mathsf{S}_{\mathtt{params}}(spar_{\mathtt{g}})$, with $\mathsf{S}_{\mathtt{params}}$ from Definition 4.3.4.

Game 1b: This game calls a simulator, using the simulated system parameters to be able to generate simulated presentation tokens.

In this game the oracle $\mathcal{O}^{\mathtt{user}}$ is updated such that:

- instead of executing

$$\mathsf{Present}\big(usk_U^*, scope, (ipk_i, RI_i, cred_i = (sig_i, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)$$

  it executes

$$\mathsf{S}_2\big(\tau, \mathtt{present}, usk_U^*, scope, (ipk_i, RI_i, cred_i' = (\epsilon, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)\,;$$

- and instead of

$$\mathsf{ITGen}\big(usk_U^*, scope, rh, (ipk_i, RI_i, (sig_i, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k+1}, E, M\big)\,,$$

  it executes

$$\mathsf{S}_2\big(\tau, \mathtt{obtain}, usk_U^*, scope, (ipk_i, RI_i, (\epsilon, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M, rh\big)\,.$$

S_2 works exactly like Present and ITGen except for the following change: For both algorithms, AuxPresent (Fig 5.1) is modified such:

$$\mathsf{SignTokenGen}(ipk_i', sig_i, ((a_{i,j})_{j=1}^{n_i}, usk, rh_i), R_i, ((c_{\mathrm{eq}(i,j)}, o_{\mathrm{eq}(i,j)})_{j\in E_i}, c_{usk}, o_{usk}, c_{rh,i}, o_{rh,i}), \hat{M}')$$

is replaced by:

$$\mathsf{S}_{\mathtt{pres}}(ipk', \tau, (a_{i,j})_{j\in R_i}, ((c_{\mathrm{eq}(i,j)})_{j\in E_i}, c_{usk}, c_{rh,i}), \hat{M}')$$

from Definition 4.3.4. That is, for presentation and issuer token generation all private inputs are discarded and the public inputs are passed on to the simulator $\mathsf{S}_{\mathtt{pres}}$.

Game 1c: In this game also the issuances of signatures are simulated. This is done by updating the oracle $\mathcal{O}^{\mathtt{user}}$ such that every call to $\mathcal{U}.\mathsf{Issue}(sit)$ is changed into a call to $\mathsf{S}_2(\tau, \mathtt{obtain}, sit)$. $\mathsf{S}_2(\tau, \mathtt{obtain}, sit)$ differs from $\mathcal{U}.\mathsf{Issue}(sit)$ in that every call to $\mathcal{U}.\mathsf{Sign}$:

$$\mathcal{U}.\mathsf{Sign}(ipk'_{k+1}, ((c_{\mathrm{eq}(k+1,j)}, o_{\mathrm{eq}(k+1,j)})_{j\in E_{k+1}}, (c_j, o_j)_{j\notin R_{k+1}\cup E_{k+1}}, \\ c_{usk}, o_{usk}), ((a_{k+1,j})_{j=1}^{n_{k+1}}, usk, rh_{k+1}))\,,$$

is replaced by the corresponding call to $\mathsf{S}_{\mathtt{iss}}$:

$$\mathsf{S}_{\mathtt{iss}}(spar_{\mathsf{s}}, \tau ipk'_{k+1}, ((a_{k+1,j})_{j\in R_{k+1}}, rh_{k+1}), ((c_{\mathrm{eq}(k+1,j)})_{j\in E_{k+1}}, (c_j)_{j\notin R_{k+1}\cup E_{k+1}}, c_{usk}))\,.$$

That is, in the above games all private inputs to the signature scheme are discarded and the public inputs are passed on to the simulator.

Game 2: This game relies on the privacy property of the revocation scheme (Definition 4.4.3) to simulate the revocation using random revocation handles instead of the real values.

The oracle $\mathcal{O}^{\mathtt{user}}$ is updated such that the tuples in list $\mathcal{C}$, contains an additional value rev_i; and the simulator S_2 is updated such that it does not take rh_i and RI_i as input:

$$\mathsf{S}_2\big(\tau, \mathtt{present}, usk^*_U, scope, (ipk_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)$$
$$\mathsf{S}_2\big(\tau, \mathtt{obtain}, usk^*_U, scope, (ipk_i, (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M, rh\big)$$

Whenever S_2 is computing a commitment and revocation token for some rh in $\mathsf{AuxPresent}$, a commitment and revocation token for a random valid revocation attribute rh_r are computed.

In this game rev_i is the code of an algorithm that on input RI_i outputs a bit indicating whether the credential is to be considered revoked. We let rev_i contain the value rh_i inside and on input RI_i it computes a commitment to rh_i and executes $rt \xleftarrow{\$} \mathsf{RevTokenGen}(a, c, o, RI, rpk)$ and $\mathsf{accept/reject} \leftarrow \mathsf{RevTokenVf}(rt, c, RI, rpk)$; and outputs 1 if the credential is to be considered revoked.

Game 3: In this game S_2 is updated such that it only takes the revealed attributes $(a_{i,j})_{j\in R_i}$ as input and whenever a commitment on a private value has to be computed, it is computed as $\mathsf{Com}(0^\kappa)$.

This game hop follows from the hiding property of the commitment scheme. Note that $\mathsf{S}_{\mathtt{iss}}$ cannot extract the committed values, and therefor has to be able to simulate without knowledge of the committed values, therefore $\mathsf{S}_{\mathtt{iss}}$ can still simulate regardless of the commitments being computed as $\mathsf{Com}(0^\kappa)$.

Game 4: This game relies on the unlinkability of the underlying pseudonym system (Definition 4.5.4) and the hiding property of the commitment scheme. The pseudonyms are simulated using random user secret keys, and keeping a list of these random user secret keys to be able to present the same pseudonym for the same user and the same scope.

The oracle $\mathcal{O}^{\mathtt{user}}$ is updated to hold a initially empty list P and a counter ctr initially set to $ctr = 0$. In both $\mathtt{present}$ and in $\mathtt{obtain}$, if $scope \neq \varepsilon$ and $(U, scope, p) \notin P$ then $\mathcal{O}^{\mathtt{user}}$ sets $ctr \leftarrow ctr + 1$, $p \leftarrow ctr$, and adds $(U, scope, p)$ to P. It then gives S_2 p as additional input, and let it keep state:

$$(\mathsf{st}_{\mathsf{S}}, nym, pt) \xleftarrow{\$} \mathsf{S}_2\big(\mathsf{st}_{\mathsf{S}}, \tau, \mathtt{present}, usk^*_U, scope, p, (ipk_i, (a_{i,j})_{j\in R_i})_{i=1}^{k}, E, M\big)$$
$$(\mathsf{st}_{\mathsf{S}}, nym, pit) \xleftarrow{\$} \mathsf{S}_2\big(\mathsf{st}_{\mathsf{S}}, \tau, \mathtt{obtain}, usk^*_U, scope, p, (ipk_i, (a_{i,j})_{j\in R_i})_{i=1}^{k}, E, M, rh\big)$$

S_2 is updated such that st_{S} contains an initially empty list $\mathcal{L}$, and such that whenever $scope \neq \varepsilon$ it it is checked whether there exists an entry $(p, scope, usk') \in \mathcal{L}$. If this is the

case, $usk = usk'$. Otherwise, $usk \xleftarrow{\$} \mathsf{UKGen}(spar_{\mathsf{g}})$ and $(p, scope, usk)$ is added to $\mathcal{L}$. Finally, the pseudonym and proof to be used is computed by committing to usk $(c_{usk}, o_{usk}) \xleftarrow{\$} \mathsf{Com}(usk)$ and computing $(nym, \pi) \xleftarrow{\$} \mathsf{NymPres}(usk, c_{usk}, o_{usk}, scope)$. If $scope = \varepsilon$ then $(c_{usk}, o_{usk}) \xleftarrow{\$} \mathsf{Com}(0^{\kappa})$ is used.

Game 5: The oracle $\mathcal{O}^{\mathtt{user}}$ is updated to initially setting state information $\mathsf{st}_{\mathsf{S}} = \tau$ and the list $\mathcal{HP}$ is dropped. $\mathcal{HP}$ is only used to store information not used and not relevant for these games.

S_2 is updated such that it is input τ as part of its state and such that it does not take usk_U^* as input as this value is not used anymore, since it is simulated.

$$(\mathsf{st}_{\mathsf{S}}, nym, pt) \xleftarrow{\$} \mathsf{S}_2\big(\mathsf{st}_{\mathsf{S}}, \mathtt{present}, scope, p, (ipk_i, (a_{i,j})_{j\in R_i})_{i=1}^{k}, E, M\big)$$
$$(\mathsf{st}_{\mathsf{S}}, nym, pit) \xleftarrow{\$} \mathsf{S}_2\big(\mathsf{st}_{\mathsf{S}}, \mathtt{obtain}, scope, p, (ipk_i, (a_{i,j})_{j\in R_i})_{i=1}^{k}, E, M, rh\big)$$

If we let $\mathcal{F} = \mathcal{O}^{\mathtt{user}}$ as defined above with rev_i as defined in Game 2; with S_1 as defined in Game 1a; and with S_2 as defined above this game is the game of experiment $\mathsf{Privacy}_{\mathsf{A}}(1^{\kappa}, n_{\mathrm{U}})$ for $b = 1$, i.e., the case where the adversary is interacting with $\mathcal{F}$ and the simulations.

□

Theorem 6.4.2 (Simulatable Weak Privacy)**.** *The* PABC*-system resulting from the generic construction presented in § 5 is weakly private in the sense of Definition 3.3.3, if the underlying commitment scheme is hiding, the* PABS*-scheme is weak user private according to Definition 4.3.5, the revocation scheme is private according to Definition 4.4.3, and the pseudonym system is unlinkable according to Definition 4.5.4.*

Proof. This proof follows analogously to the proof of Theorem 6.4.1 as a series of games, however, as all the games except for Game 1b are identical in these two proofs, we only include Game 1b here. The difference between the two versions of Game 1b, is that in Theorem 6.4.1 the simulator of the PABS-scheme can simulate without knowing if two presentations are done over the same or two different signatures, however, the simulator of a weak user private PABS-scheme needs the knowledge of whether a presentation of a signature is done over a previous presented signature. the simulator needs this to be able to simulate the linkability of these two presentations. This is done in the same way as in Game 4, the step introducing the simulation of the pseudonym, with introducing a list of anonymous numbers, so the simulator gets the same number each time it has to do a presentation of the same signature.

Game 0-1a: These are identical of the proof of Theorem 6.4.1.

Game 1b: This game calls a simulator, using the simulated system parameters to be able to generate simulated presentation tokens of the PABS-scheme.

In this game the oracle $\mathcal{O}^{\mathtt{user}}$ is updated such that the signatures behind the presentations are simulated by the simulator which is given an index to keep track of whether it has simulated presentations over this signature before.

The oracle $\mathcal{O}^{\mathtt{user}}$ is updated to hold a initially empty list L and a counter ctr_{cid} initially set to $ctr_{cid} = 0$. In both `present` and in `obtain`, it first checks if a tuple (cid_i, p_{cid_i}) in L exists. If not, it sets $ctr_{cid} \leftarrow ctr_{cid} + 1$, $p_{cid} \leftarrow ctr_{cid}$, and adds a tuple (cid, p_{cid}) to L. It then gives S_2 $\{p_{cid_i}\}_{i=1}^{k}$ as additional input, and let it keep state such that:

- instead of executing

$$\mathsf{Present}\big(usk_U^*, scope, (ipk_i, RI_i, cred_i = (sig_i, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)$$

it executes

$$\mathsf{S}_2\big(\tau, \texttt{present}, usk_U^*, scope, (ipk_i, RI_i, cred_i' = (\epsilon, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i)_{i=1}^{k}, E, M\big)\,;$$

- and instead of

$$\mathsf{ITGen}\big(usk_U^*, scope, rh, (ipk_i, RI_i, cred_i(sig_i, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i, p_{cid_i})_{i=1}^{k+1}, E, M\big)$$

it executes

$$\mathsf{S}_2\big(\tau, \texttt{obtain}, usk_U^*, scope, (ipk_i, RI_i, cred_i(\epsilon, rh_i), (a_{i,j})_{j=1}^{n_i}, R_i, p_{cid_i})_{i=1}^{k}, E, M, rh\big)\,.$$

S_2 works exactly like $\mathsf{Present}$ and ITGen except for the following change: For both algorithms, $\mathsf{AuxPresent}$ (Fig 5.1) is modified such that:

$$\mathsf{SignTokenGen}(ipk_i', sig_i, ((a_{i,j})_{j=1}^{n_i}, usk, rh_i), R_i, ((c_{\mathrm{eq}(i,j)}, o_{\mathrm{eq}(i,j)})_{j\in E_i}, c_{usk}, o_{usk}, c_{rh,i}, o_{rh,i}), \hat{M}')$$

is replaced by:

$$\mathsf{S}_{\texttt{pres}}(ipk', \tau, p_{cid_i}, (a_{i,j})_{j\in R_i}, ((c_{\mathrm{eq}(i,j)})_{j\in E_i}, c_{usk}, c_{rh,i}), \hat{M}')$$

from Definition 4.3.5. Note that also here the difference from the proof of Theorem 6.4.1 is the value p_{cid}. Therefore under presentation and issuer token generation all private inputs are discarded and only the public inputs are passed on to the simulator $\mathsf{S}_{\texttt{pres}}$, with an index to make simulation of linkability possible..

Game1c-5: Follows trivially along the line of the same games of the proof of Theorem 6.4.1.

The claim follows. □

Chapter 7

Secure Instantiations of Building Blocks

We next present secure instantiation of the building blocks defined in § 4. Our constructions are strongly based on existing schemes. More precisely, we use Pedersen commitments over the integers [Ped91, DF02]. Our signature scheme is based on CL-signatures, cf. § 2.2.3 and [CL02], and our revocation scheme is a variant of that introduced by Nakanishi et al. [NFHF09]. Finally, the used pseudonym system was first used by Identity Mixer [Tea10].

On a very high level, these existing schemes are all equipped with zero-knowledge proofs to obtain the required extractability properties, which are achieved following the "encryption to the sky" paradigm using Paillier encryptions, cf. § 2.2.2.

7.1 Global System Parameter Generation

On input 1^κ, $\mathsf{SPGen_g}$ behaves as specified next:

- It specifies ℓ arbitrarily as in § 4.1,
- chooses an arbitrary integer L, and
- chooses security parameters $\kappa_c \geq \kappa$ specifying the soundness of all involved proofs as well as $\kappa_v \geq \kappa$ controlling the statistical zero-knowledge property of all protocols.
- The algorithm further chooses a Paillier encryption key $(esk, epk) \xleftarrow{\$} \mathsf{EncKGen}(1^\kappa)$ as described in § 2.2.2.
- It sets $spar'_\mathsf{g} = (epk, \kappa_c, \kappa_v)$ and
- computes $spar_\mathsf{c} \xleftarrow{\$} \mathsf{SPGen_c}(1^\kappa, \ell, spar'_\mathsf{c})$ and $ck \xleftarrow{\$} \mathsf{ComKGen}(spar_\mathsf{c})$.

$\mathsf{SPGen_g}$ outputs:

$$spar_\mathsf{g} = (1^\kappa, \ell, \ell, spar_\mathsf{c}, ck, L, spar'_\mathsf{g})\,.$$

7.2 A Commitment Scheme based on Pedersen Commitments

We next describe a commitment scheme that satisfies our requirements. The scheme is a direct combination of the following two components. First, the commitment scheme for message space $\mathbb{Z}_q$, $q \in \mathbb{P}$, by Pedersen [Ped91] and its generalization to an integer commitment scheme by Damgård and Fujisaki [DF02]. Second, the zero-knowledge proofs of knowledge by Fujisaki et al. [FO97, DF02].

7.2.1 System Parameter Generation

On input $(1^\kappa, \ell, spar'_{\mathsf{g}})$, $\mathsf{SPGen_c}$ outputs

$$spar_{\mathsf{c}} = (1^\kappa, \ell, \varepsilon)\,.$$

7.2.2 Commitment Key Generation

On input $spar_{\mathsf{c}}$, ComKGen behaves as follows:

- It chooses a safe RSA modulus n of length κ_n, and
- $h \xleftarrow{\$} \mathbb{Z}_n^*$ as well as $g \xleftarrow{\$} \langle h \rangle$

The algorithm outputs

$$ck = (n, g, h)\,.$$

7.2.3 Committing to Messages

To commit to a message $m \in \pm\{0,1\}^\lambda$, this algorithm computes $o \xleftarrow{\$} [0, n]$ and outputs

$$(c, o) = (g^m h^o, o)\,.$$

7.2.4 Commitment Opening Verification

ComOpenVf outputs accept, if and only if $c = g^m h^o$.

7.2.5 Commitment Opening Proof

On input $ck, spar_{\mathsf{c}}, c, m, o$, ComPf behaves as follows:

- It first computes $\mathfrak{c}_o = \mathsf{Enc}(o)$ and $\mathfrak{c}_m = \mathsf{Enc}(m)$. Let r_o and r_m denote the random coins used in the encryptions.
- Next, it computes the following non-interactive zero-knowledge proof:

$$\pi' \xleftarrow{\$} \mathsf{ZKP_{FS}}\big[(o, \mu, \rho_o, \rho_m) \;:\; c = g^\mu h^o \;\wedge \mathfrak{c}_o = \mathsf{Enc}(o; \rho_o) \;\wedge \\ \mathfrak{c}_m = \mathsf{Enc}(\mu; \rho_m)\big](c, \mathfrak{c}_o, \mathfrak{c}_m, ck, spar_{\mathsf{c}})\,,$$

  where the used encryption key was extracted from the global system parameters contained in $spar_{\mathsf{c}}$.

The algorithm outputs:

$$\pi = (\mathfrak{c}_o, \mathfrak{c}_m, \pi')\,.$$

7.2.6 Commitment Proof Verification

ComProofVf outputs accept, if and only if the input has the correct form, and the given proof verifies correctly.

7.3 A PABS-Scheme Based on CL-Signatures

We now show how to construct a PABS-scheme satisfying the discussed security properties. The scheme is based on the CL-signature scheme recapitulated in § 2.2.3.

7.3.1 System Parameter Generation

On input $spar_{\mathsf{g}}$, $\mathsf{SPGen_s}$ behaves as follows:

- It specifies $\mathcal{AS} = \pm\{0,1\}^{\ell_m}$ for an arbitrary $\ell_m \leq \ell$,
- and defines $spar_{\mathsf{s}} = \varepsilon$.

The algorithm outputs:

$$spar_{\mathsf{s}} = (spar_{\mathsf{g}}, \mathcal{AS}, spar_{\mathsf{s}})\,.$$

7.3.2 Key Generation

This algorithm works similar to the key generation of CL-signatures, cf. § 2.2.3, but additionally to the key also outputs a proof that it is well-formed.

More precisely, on input $spar_{\mathsf{s}}$, IKGen behaves as follows:

- It first computes κ_n according to [Blu13] for security level κ, and
- then chooses a strong RSA modulus for which p and q have length $\kappa_n/2$.
- The algorithm computes $S \xleftarrow{\$} QR_n$, and $R_1, \ldots, R_L, Z \xleftarrow{\$} \langle S \rangle$ by choosing $x_i, x_z \xleftarrow{\$} \{0,1\}^{\kappa_n}$ and setting $R_i := S^{x_i}$ for $i = 1, \ldots, L$, $Z := S^{x_z}$.
- Furthermore, a NIZK proof π is computed as follows:

$$\pi \xleftarrow{\$} \mathsf{ZKP_{FS}}\left[(\chi_1, \ldots, \chi_L, \chi_z) : \bigwedge_{i=1}^{L} R_i = S^{\chi_i} \wedge Z = S^{\chi_z}\right](R_1, \ldots, R_L, Z, S, n, spar_{\mathsf{s}}).$$

The algorithm outputs:

$$(ipk, isk) = ((R_1, \ldots, R_L, Z, S, n, \pi), p)\,.$$

7.3.3 Key Verification

KeyVf outputs accept, if and only if the proof contained in ipk verifies correctly.

7.3.4 Signature Issuance

Figure 7.1 describes the Sign protocol for attributes $\vec{a} = (a_1, \ldots, a_L)$. On a high level, the user first computes a template t for the unrevealed attributes, such that t is consistent with what the issuer would do in a basic CL-issuance protocol, cf. § 2.2.3. Using t, the parties then essentially perform an instance of that protocol.

7.3.5 Signature Verification

The correctness of a signature can be verified on input a signature $sig = (e, A, v)$, attributes $\vec{a} = (a_1, \ldots, a_L)$ and an issuer public key ipk as follows:

$\mathsf{SigVf}(sig, \vec{a}, ipk)$ outputs accept if

$$Z \equiv A^e \prod_{i=1}^{L} R_i^{a_i} S^v \pmod{n} \;\wedge\; 2^{\ell_e - 1} < e < 2^{\ell_e} \;\wedge\; \bigwedge_{i=1}^{L} a_i \in \pm\{0,1\}^{\ell_m}\,,$$

and reject otherwise.

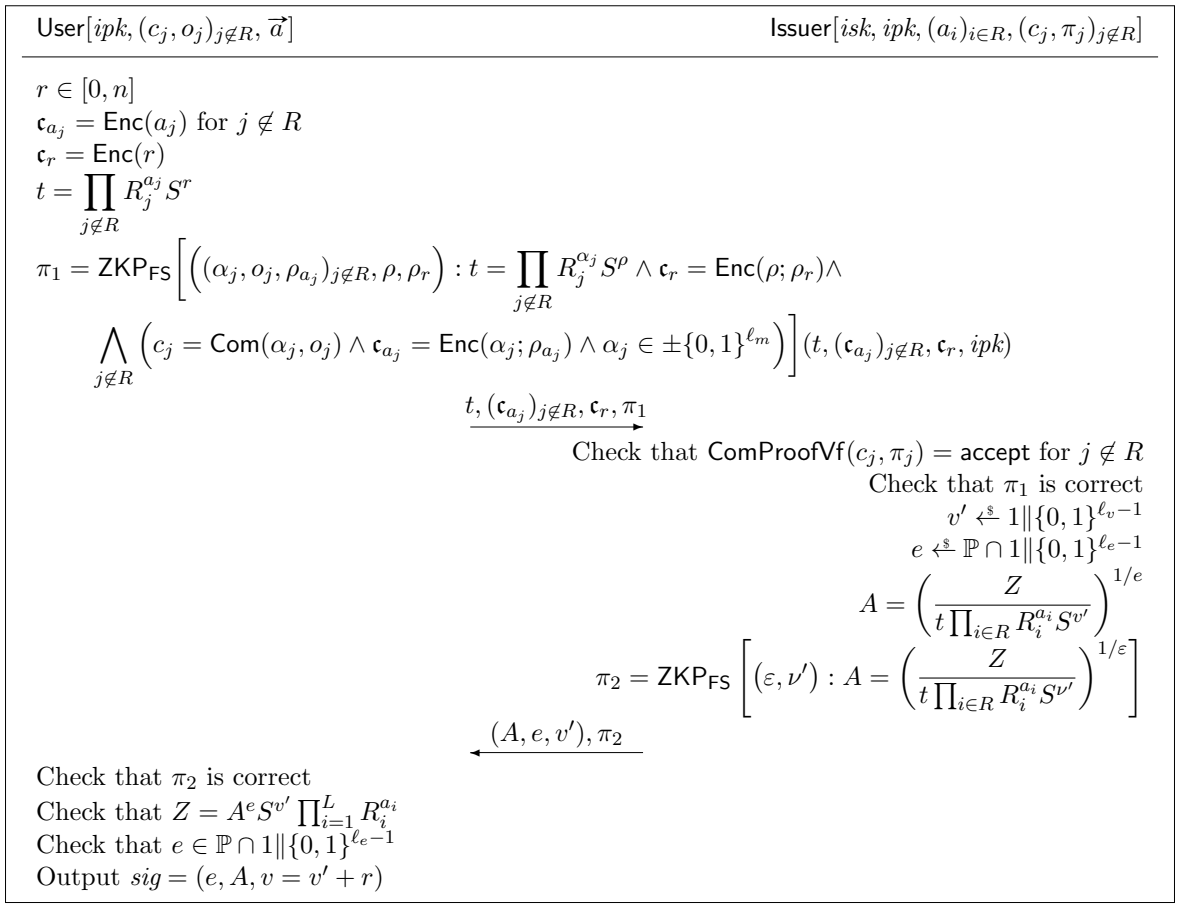

Figure 7.1: Issuance of a signature for attributes $(a_1, \ldots, a_L)$. If any of the checks fails, the protocol aborts.

7.3.6 Signature Presentation Token Generation

On inputs $(ipk, sig, \vec{a}, R, (c_j, o_j)_{j \in C}, M)$, SignTokenGen behaves as follows:

- In a first step, the user re-randomizes his signature (e, A, v), obtaining a signature (e, A', v') as described in § 2.2.3.
- Next, SignTokenGen encrypts all unrevealed attributes and commitment openings under the public key specified in $spar_{\mathsf{s}}$. That is, it computes:

$$\begin{aligned} \mathfrak{c}_{a_j} &= \mathsf{Enc}(a_j) \quad \forall j \notin R\,, & \mathfrak{c}_e &= \mathsf{Enc}(e)\,, \\ \mathfrak{c}_{o_j} &= \mathsf{Enc}(o_j) \quad \forall j \in C\,, & \mathfrak{c}_{v'} &= \mathsf{Enc}(v')\,. \end{aligned}$$

  Let $r_{a_j}, r_{o_j}, r_e, r_{v'}$ be the random coins being used in these encryptions.

- SignTokenGen then generates the following signature proof of knowledge:

$$\begin{aligned} \pi' \leftarrow^{\$} \mathsf{ZKP_{FS}}\Big[& ((\alpha_j, \rho_{a_j})_{j \notin R}, (o_i, \rho_{o_i})_{i \in C}, \nu', \rho_{v'}, \varepsilon, \rho_e) : \\ & Z \prod_{i \in R} R_i^{-a_i} = A'^{\varepsilon} \prod_{j \notin R} R_j^{\alpha_j} S^{\nu'} \ \wedge \ \bigwedge_{j \notin R} \Big(\mathfrak{c}_{a_j} = \mathsf{Enc}(\alpha_j; \rho_{a_j}) \ \wedge \ \alpha_j \in \pm\{0,1\}^{\ell_m} \Big) \ \wedge \\ & \bigwedge_{i \in C} \Big(c_i = \mathsf{Com}(\alpha_i; o_i) \ \wedge \ \mathfrak{c}_{o_i} = \mathsf{Enc}(o_i; \rho_{o_i}) \Big) \ \wedge \ \mathfrak{c}_e = \mathsf{Enc}(\varepsilon; \rho_e) \ \wedge \ \mathfrak{c}_{v'} = \mathsf{Enc}(\nu'; \rho_{v'}) \ \wedge \\ & 2^{\ell_e - 1} < \varepsilon < 2^{\ell_e} \Big] ((a_i)_{i \in R}, A', \mathfrak{c}_e, \mathfrak{c}_{v'}, (c_j, \mathfrak{c}_{o_j})_{j \in C}, (\mathfrak{c}_{a_j})_{j \notin R}, spar_{\mathsf{s}}, ipk, M) \end{aligned}$$

The algorithm outputs:

$$spt = (A', \mathfrak{c}_e, \mathfrak{c}_{v'}, (\mathfrak{c}_{a_j})_{j \notin R}, (\mathfrak{c}_{o_j})_{j \in C}, \pi') \,.$$

7.3.7 Signature Presentation Token Verification

SignTokenVf outputs accept, if and only if *spt* has the correct form and the contained proof verifies correctly, and all revealed attributes are in $\pm\{0,1\}^{\ell_m}$.

7.4 A PABS-Scheme Based on Brands' Signatures

We now show how to construct a PABS-scheme satisfying the discussed security properties. The scheme is based on the Brands signature scheme from § 2.2.3.

7.4.1 System Parameter Generation

On input $spar_{\mathsf{g}}$, $\mathsf{SPGen_s}$ defines behaves as follows:

- It specifies $\mathcal{AS} = \{0,1\}^{\ell_m}$ for an arbitrary $\ell_m \leq \ell$,
- and defines $spar_{\mathsf{s}} = \varepsilon$.

The algorithm outputs:

$$spar_{\mathsf{s}} = (spar_{\mathsf{g}}, \mathcal{AS}, spar_{\mathsf{s}}) \,.$$

7.4.2 Key Generation

On input $spar_{\mathsf{s}}$, IKGen behaves as follows:

- It first computes κ_p according to [Blu13] for security level κ, and
- then chooses a prime p of length κ_p and a prime q such that $q \geq 2^{\ell_m}$ and $q|p-1$.
- The algorithm picks a random generator g of the unique subgroup of order q in $\mathbb{Z}_p^*$, and
- then chooses random values $y_i \in_R \mathbb{Z}_q$ and sets $g_i = g^{y_i}$ for $i = 0, \ldots, L$.

The algorithm outputs:

$$(ipk, isk) = ((g, p, q, g_0, g_1, \ldots, g_L), y_0) \,.$$

7.4.3 Key Verification

KeyVf outputs accept, if and only if $p \in \mathbb{P} \cap 1\|\{0,1\}^{\kappa_p - 1}$, $q \in \mathbb{P}$, $q|(p-1)$, $q \geq 2^{\ell_m}$, g has order q and $y_i^q = 1$ for all i.

7.4.4 Signature Issuance

Let H be a secure cryptographic hash function with output domain $\mathbb{Z}_q$. Figure 7.2 describes the Sign protocol for attributes $\vec{a} = (a_1, \ldots, a_L)$

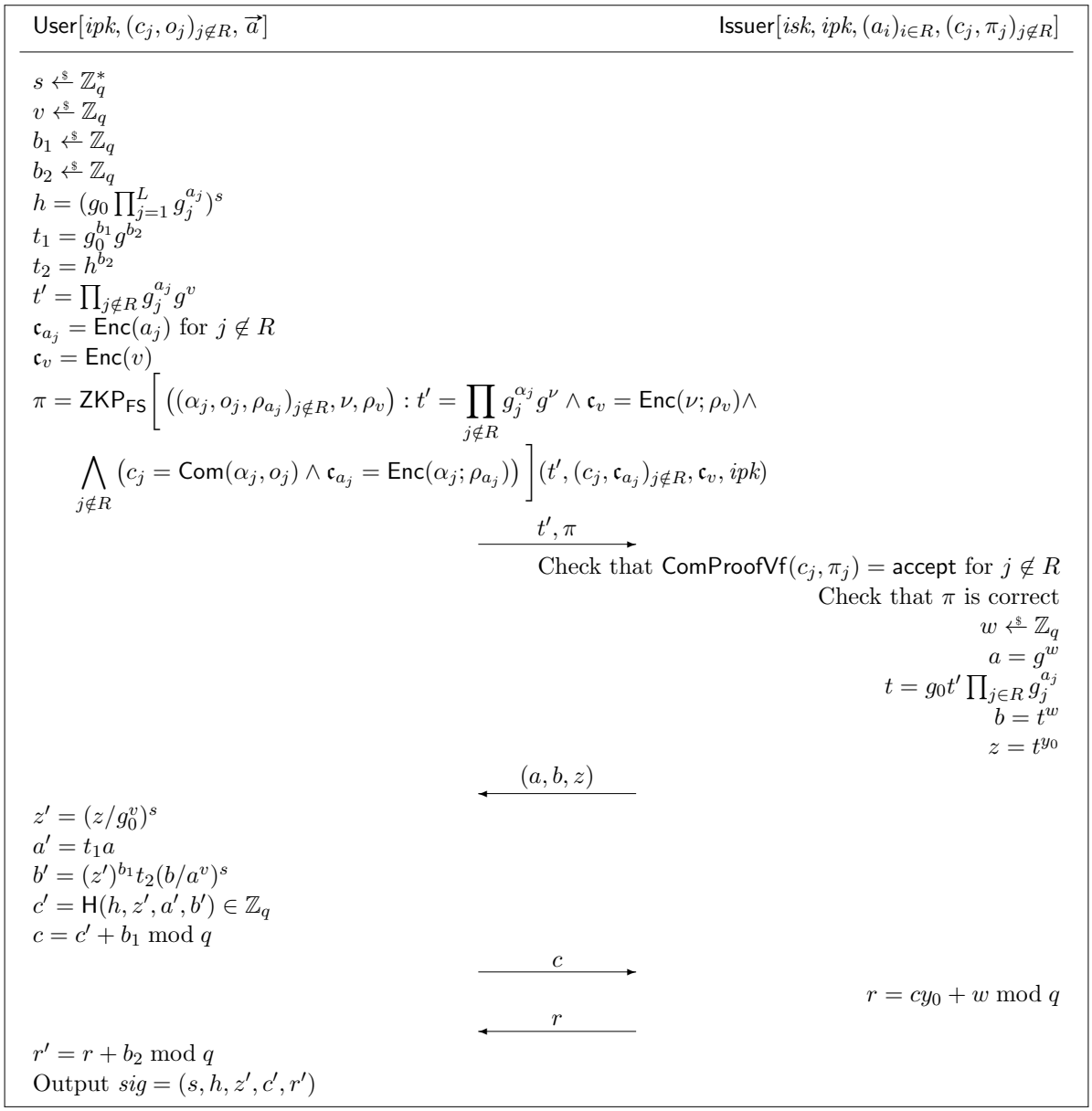

Figure 7.2: Issuance of a signature for attributes $(a_1, \ldots, a_L)$. In the zero-knowledge proof the user acts as the prover. If any of the checks or proofs fails, the protocol aborts.

7.4.5 Signature Verification

The correctness of a signature $sig = (s, h, z', c', r')$ on attributes $\vec{a} = (a_1, \ldots, a_L)$ under an issuer public key ipk can be verified as follows:

$\mathsf{SigVf}(sig, \vec{a}, ipk)$ outputs accept if the following two equations hold:

$$h \stackrel{?}{=} (g_0 \prod_{i=1}^{L} g_i^{a_i})^s$$
$$c' \stackrel{?}{=} \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'} (z')^{-c'})$$

and reject otherwise.

7.4.6 Signature Presentation Token Generation

On inputs $(ipk, sig, \vec{a}, R, (c_j, o_j)_{j\in C}, M)$, $\mathsf{SignTokenGen}$ behaves as follows:

- In a first step $\mathsf{SignTokenGen}$ encrypts all unrevealed attributes and commitment openings under the public key specified in $spar_{\mathtt{s}}$. That is, it computes:

$$\begin{aligned} \mathfrak{c}_{a_j} &= \mathsf{Enc}(a_j) \quad \forall j \notin R\,, \\ \mathfrak{c}_{o_j} &= \mathsf{Enc}(o_j) \quad \forall j \in C\,, \\ \mathfrak{c}_s &= \mathsf{Enc}(s)\,. \end{aligned}$$

  Let r_{a_j}, r_{o_j}, r_a be the random coins being used in these encryptions.

- $\mathsf{SignTokenGen}$ then generates the following signature proof of knowledge:

$$\begin{aligned} \pi \xleftarrow{\$} \mathsf{ZKP}_{\mathsf{FS}}\Big[&((\alpha_j, \rho_{a_j})_{j\notin R}, (o_i, \rho_{o_i})_{i\in C}, \sigma, \rho_s) : h = \Big(g_0 \prod_{j\in R} g_i^{a_i} \prod_{j\notin R} g_i^{\alpha_j}\Big)^{\sigma} \ \wedge \\ &\bigwedge_{j\notin R} \mathfrak{c}_{a_j} = \mathsf{Enc}(\alpha_j; \rho_{a_j}) \ \wedge \ \bigwedge_{i\in C} \Big(c_i = \mathsf{Com}(\alpha_i; o_i) \ \wedge \ \mathfrak{c}_{o_i} = \mathsf{Enc}(o_i; \rho_{o_i})\Big) \ \wedge \\ &\mathfrak{c}_s = \mathsf{Enc}(\sigma; \rho_s)\Big]((a_i)_{i\in R}, h, z', c', r', \mathfrak{c}_s, (c_j, \mathfrak{c}_{o_j})_{j\in C}, (\mathfrak{c}_{a_j})_{j\notin R}, spar_{\mathtt{s}}, ipk, M) \end{aligned}$$

The algorithm outputs:

$$spt = (h, z', c', r', \mathfrak{c}_s, (\mathfrak{c}_{a_j})_{j\notin R}, (\mathfrak{c}_{o_j})_{j\in C}, \pi)\,.$$

7.4.7 Signature Presentation Token Verification

$\mathsf{SignTokenVf}$ outputs accept, if and only if spt has the correct form, the proof π contained in the signature presentation token spt verifies correctly and the following equation holds:

$$c' \stackrel{?}{=} \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'} (z')^{-c'})$$

7.5 A Revocation Scheme Based on the Nakanishi et al. Scheme

In the following we sketch a revocation scheme satisfying the properties defined in § 4.4. The high level idea is taken from Nakanishi et al. [NFHF09]: the revocation authority signs intervals of unrevoked attributes, and at presentation time a user shows that his attribute lies in one of these intervals without revealing which one. For this approach to work we assume that the attribute space is given by an integer interval as is the case for the scheme presented earlier. If this is not the case, one first has to define a 1 to 1 mapping from the attribute space to some integer interval, and then additionally prove that this mapping was done correctly for the specific attribute.

The following description of the required algorithms is kept at an abstract level. While a trivial solution especially for the presentation protocol can be realized using standard techniques to prove partial knowledge of a witness [CDS94], an efficient solution based on pairings follows directly from the work of Nakanishi et al. [NFHF09].

7.5.1 System Parameter Generation

On input $spar_{\mathtt{g}}$, $\mathsf{SPGen}_{\mathtt{r}}$ behaves as follows:

- It defines $\mathcal{RS} = \pm\{0,1\}^{\ell_m}$, and

- sets $spar'_{\mathtt{r}} = (\mathsf{SignGen}, \mathsf{Sign}, \mathsf{SignVf})$, where the algorithms define a signature scheme which is unforgeable under chosen message attacks for a message space containing $[-2^{\ell_m}, 2^{\ell_m}] \supseteq \mathcal{RS}$. For the trivial solution mentioned before this could, e.g., be instantiated with CL-signatures.

The algorithm outputs:

$$spar_{\mathtt{r}} = (spar_{\mathtt{g}}, \mathcal{RS}, spar'_{\mathtt{r}}) .$$

7.5.2 Revocation Setup

On input $spar_{\mathtt{r}}$, RKGen behaves as follows:

- It generates a signing/verification key for the specified signature scheme, i.e., it computes $(rsk, rpk) \xleftarrow{\$} \mathsf{SignGen}(1^\kappa)$.
- It then defines $r_{\min} = -2^{\ell_m}$ and $r_{\max} = 2^{\ell_m}$, such that for all $a \in \mathcal{AS}(1^\kappa)$ it holds that $r_{\min} < a < r_{\max}$.
- It sets $\sigma_{0,0} = \mathsf{Sign}(rsk, (0, r_{\min}, r_{\max}))$ as well as $\vec{r}_0 = (r_{\min}, r_{\max})$ and $RI = (0, \{\sigma_{0,0}\}, \vec{r}_0)$.

The algorithm outputs

$$(rsk, rpk, RI) .$$

7.5.3 Attribute Revocation

On input (rsk, RI, a), where $RI = (t, \{\sigma_{t,j}\}_{j=0}^{t}, \vec{r})$ and $\vec{r}_t = (r_{t,0}, \ldots, r_{t,t+1})$, Revoke behaves as follows:

- It first determines j' such that $r_{t,j'} < a < r_{t,j'+1}$, and
- defines

$$\vec{r}_{t+1} = (r_{t+1,0}, \ldots, r_{t+1,t+2}) = (r_{t,0}, \ldots, r_{t,j'}, a, r_{t,j'+1}, \ldots, r_{t,t+1}) .$$

- For all $j = 0, \ldots, t+1$ it then computes

$$\sigma_{t+1,j} = \mathsf{Sign}(rsk, (t+1, r_{t+1,j}, r_{t+1,j+1})) .$$

The algorithm outputs

$$RI = (t+1, \{\sigma_{t+1,j}\}_{j=0}^{t+1}, \vec{r}_{t+1}) .$$

7.5.4 Revocation Token Generation

On input (a, c, o, RI, rpk), where $RI = (t, \{\sigma_{t,j}\}_{j=0}^{t}, \vec{r})$, $\mathsf{RevTokenGen}$ behaves as follows:

- It computes $\mathfrak{c}_a = \mathsf{Enc}(a)$ and $\mathfrak{c}_o = \mathsf{Enc}(o)$, with random coins r_a and r_o, respectively.
- It then computes a non-interactive zero-knowledge proof of knowledge for the following statement:

$$\pi' \xleftarrow{\$} \mathsf{ZKP}_{\mathsf{FS}}\Big[(\alpha, o, \iota) \ : \ c = \mathsf{Com}(\alpha, o) \ \wedge \ \sigma_{t,\iota} = \mathsf{Sign}(t, r_\iota, r_{\iota+1}) \wedge \mathfrak{c}_a = \mathsf{Enc}(\alpha, \rho_\alpha) \ \wedge \ \mathfrak{c}_o = \mathsf{Enc}(o, \rho_o) \ \wedge \ r_\iota < \alpha < r_{\iota+1}\Big](spar_{\mathtt{r}}, c, RI, rpk, \mathfrak{c}_o, \mathfrak{c}_a) .$$

The algorithm outputs

$$rt = (\mathfrak{c}_a, \mathfrak{c}_o, \pi') .$$

Note that the index ι remains hidden during the presentation, and thus no information about the attribute is leaked during the proof.

7.5.5 Revocation Token Verification

The algorithm RevTokenVf outputs accept, if and only if the following is satisfied:

- rt has the correct form,
- all $\sigma_{t,j}$ are valid signatures for $0 \leq j \leq t+1$, and
- the contained proof verifies correctly.

7.6 A Pseudonym Scheme

In the following we describe a realization of a pseudonym extension, satisfying the properties defined in § 4.5.

7.6.1 System Parameter Generation

Before being able to specify this algorithm, we need to recap the following definition:

Definition 7.6.1 (DDH Assumption)**.** *Let $\mathcal{G}$ be a cyclic group of prime order q. The* DDH assumption *holds for $\mathcal{G}$, if the following two distributions are computationally indistinguishable:*

$$\{(g, g^x, g^y, g^{xy})\} \sim \{(g, g^x, g^y, g^z)\},$$

where $g \xleftarrow{\$} \mathcal{G}$ and $x, y, z \xleftarrow{\$} \mathbb{Z}_q$.

On input $spar_{\mathsf{g}}$, $\mathsf{SPGen}_{\mathsf{p}}$ now behaves as follows:

- It first chooses a group $\mathcal{G}$ of prime order $q \in \mathbb{P} \cap \{0,1\}^{\ell_m}$, such that the DDH-assumption holds for $\mathcal{G}$.
- Furthermore, it chooses $\mathsf{H} \xleftarrow{\$} \mathcal{H}_\kappa$, where $\mathcal{H}_\kappa$ is a family of collision resistant hash functions for security parameter κ with co-domain $\mathcal{G}$.
- It sets $spar'_{\mathsf{p}} = (\mathcal{G}, \mathsf{H}, q)$.

The algorithm outputs

$$spar_{\mathsf{p}} = (spar_{\mathsf{g}}, spar'_{\mathsf{p}}).$$

7.6.2 User Key Generation

On input $spar_{\mathsf{p}}$, a user secret key is chosen as $usk \xleftarrow{\$} \mathbb{Z}_q$.

7.6.3 Pseudonym Generation

On input $(usk, scope)$, NymGen outputs

$$nym = (\mathsf{H}(scope))^{usk}.$$

7.6.4 Pseudonym Presentation Generation

On input $(usk, c, o, scope)$, $\mathsf{NymPres}$ behaves as follows:

- It first computes $nym \leftarrow \mathsf{NymGen}(usk, scope)$.
- It then computes $\mathfrak{c}_o = \mathsf{Enc}(o)$ and $\mathfrak{c}_{usk} = \mathsf{Enc}(usk)$. Let the associated random coins be given by r_o and r_{usk}, respectively.
- Next, the algorithm computes the following non-interactive zero-knowledge proof of knowledge:

$$\pi' \xleftarrow{\$} \mathsf{ZKP}_{\mathsf{FS}}\big[(u, o, \rho_o, \rho_{usk}) \ : \ nym = \mathsf{H}(scope)^u \ \wedge \ c = \mathsf{Com}(u, o) \ \wedge \\ \mathfrak{c}_{usk} = \mathsf{Enc}(u; \rho_{usk}) \ \wedge \ \mathfrak{c}_o = \mathsf{Enc}(o; \rho_o)\big]\left(spar_{\mathfrak{p}}, c, \mathfrak{c}_{usk}, \mathfrak{c}_o, scope, nym\right) .$$

The algorithm outputs

$$(nym, \pi) = (nym, (\mathfrak{c}_o, \mathfrak{c}_{usk}, \pi')) .$$

7.6.5 Pseudonym Verification

This algorithm outputs accept, if and only if the input has the correct form and the contained proof verifies correctly.

Chapter 8

Security of Instantiation

8.1 Security Proofs of Commitment Scheme

Lemma 8.1.1. *Under the strong RSA assumption, the commitment scheme described in § 7.2 is perfectly correct, computationally binding, and statistically hiding according to Definitions 4.2.1, 4.2.2, and 4.2.3.*

We omit the proof here and refer to the original work of Pedersen [Ped91] and Fujisaki et al. [DF02, FO97].

Lemma 8.1.2. *Under the strong RSA assumption and the quadratic residuosity assumption, the commitment scheme described in § 7.2 is opening extractable according to Definition 4.2.4 in the random oracle model.*

Proof. We have to show correctness, soundness, zero-knowledge and extractability, cf. Definitions 2.2.1 and 2.2.2.

Correctness follows directly from the correctness of the non-interactive zero knowledge proof in 7.2.

For the zero-knowledge property, the simulator first computes $\mathfrak{c}_o$ and $\mathfrak{c}_m$ as encryptions to 0, and then outputs whatever the standard simulator for the Fiat-Shamir heuristic outputs on input $c, \mathfrak{c}_o, \mathfrak{c}_m, ck, spar_{\mathsf{c}}$, cf., e.g., [BPW12]. By the semantic security of the encryption scheme and the zero-knowledge property of the Fiat-Shamir heuristic, the claim follows.

Soundness follows from the fact that the Fiat-Shamir heuristic is sound for the given relation in the random oracle model [PS96].

Following the "encryption to the sky" paradigm, extractability can now be seen as follows. The faked parameters are computed by E_1 just as the original ones, except that the decryption key of Enc is stored in τ, which obviously makes real and faked parameters indistinguishable. Now, given a commitment c and a proof π generated by the adversary, E_2 decrypts $\mathfrak{c}_o, \mathfrak{c}_m$, obtaining o' and m', respectively. Together with the correctness of the encryption scheme, the soundness property guarantees that the extracted witnesses are the same as those for the commitment c with overwhelming probability. The claim now follows immediately. □

8.2 Security Proofs of PABS-Scheme Based on CL Signatures

Lemma 8.2.1. *Assuming that CL-signatures (cf. § 2.2.3) are existentially UF-CMA secure according to Definition 2.2.5, the* PABS*-scheme described in § 7.3 is signature unforgeable according to Definition 4.3.2 in the random oracle model.*

Proof. Let Ω denote the probability space as in Definition 4.3.2. It then holds that:

$$\begin{aligned}
&\Pr\Big[\mathsf{SignTokenVf}(ipk, spt, (a_i)_{i\in R}, (c_j)_{j\in C}, M) = \mathsf{accept} \ \wedge \ \Big(\mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} \ \vee \\
&\quad \exists k \in C : \mathsf{ComOpenVf}(a_k, c_k, o_k) = \mathsf{reject} \ \vee \ \big(((a_i)_{i\in R}, (c_j)_{j\in C}, M) \notin \mathcal{L}_{\mathrm{spt}} \ \wedge \ \vec{a} \notin \mathcal{L}_{\mathrm{iss}}\big)\Big) : \Omega\Big] \\
\leq\ &(i) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ \mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} : \Omega\Big] + \\
&(ii) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ \exists k \in C : \mathsf{ComOpenVf}(a_k, c_k, o_k) = \mathsf{reject} : \Omega\Big] + \\
&(iii) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ ((a_i)_{i\in R}, (c_j)_{j\in C}, M) \notin \mathcal{L}_{\mathrm{spt}} \ \wedge \ \vec{a} \notin \mathcal{L}_{\mathrm{iss}} : \Omega\Big].
\end{aligned}$$

We now show that each of the terms is negligible as a function of the security parameter.

For (i), note that signature presentation tokens are in particular non-interactive zero-knowledge proofs for $(a_i)_{i\notin R}$ with $a_i \in \pm\{0,1\}^{\ell_m}$, $2^{\ell_e-1} < e < 2^{\ell_e}$ and v' such that:

$$Z \prod_{i\in R} R_i^{-a_i} = A'^e \prod_{j\notin R} R_j^{a_j} S^{v'} ,$$

and thus these values can be extracted from the adversary with overwhelming probability. Furthermore, $\mathsf{SignTokenVf}$ checks that all revealed attributes are elements of $\pm\{0,1\}^{\ell_m}$ as well. Together, this guarantees that

$$Z \equiv A'^e \prod_{i=1}^{L} R_i^{a_i} S^{v'} \pmod{n} \ \wedge \ 2^{\ell_e-1} < e < 2^{\ell_e} \ \wedge \ \bigwedge_{i=1}^{L} a_i \in \pm\{0,1\}^{\ell_m} ,$$

which is what SignVf tests.

For (ii), note that signature presentation tokens are in particular non-interactive zero-knowledge proofs for valid openings of $(c_j)_{j\in C}$. Therefore, the adversary only has a negligible chance of outputting commitments and a valid signature presentation token, although not knowing the content and openings of at least one commitment.

For (iii), we first modify the game the adversary has to win through a number of game hops, such that the winning probability of the adversary does not change by more than a negligible amount. We then show that if the adversary can win the last game of this sequence, then he can break the UF-CMA property of CL-signatures.

Game 1: This is the original game specified in (iii).

Game 2: In this game, we additionally require that the extracted signature passes SigVf, and that valid openings of all commitments can be extracted. From (i) and (ii) it follows immediately that the winning probability of the adversary only changes negligibly.

Game 3: Requests to $\mathcal{O}^{\mathtt{user}}$ are now simulated. That is, `obtain` requests are simply answered with a *cid*, and $(cid, \vec{a}, \varepsilon)$ is added to $\mathcal{L}_{\mathrm{iss}}$. Upon receiving a `token` request, A' is chosen randomly in $\langle S\rangle$, and $\mathfrak{c}_{v'}, \mathfrak{c}_e$ are computed as encryptions to zero. The required non-interactive zero-knowledge proof is then simulated programming the random oracle. By the CPA-security of the encryption scheme and the zero-knowledge property of the deployed proof, this game is indistinguishable from the previous game for every adversary.

Game 4: The proof contained in the issuer public key is now simulated. The indistinguishability follows from the zero-knowledge property of the deployed proof.

We now sketch how an adversary A winning this game can be used to construct an adversary B that forges CL-signatures. Given a public CL-key as input, B fakes the required proof and forwards it as input to A. Calls to $\mathcal{O}^{\mathtt{user}}$ are answered by B as described in Game 3. Furthermore, for calls to $\mathcal{O}^{\mathtt{issuer}}$, B first extracts the values inside the commitments, and requests a signature from the CL-signing oracle. Using this signature, it then runs the issuance protocol with A. All inputs provided to the adversary now have exactly the same distribution as in Game 4.

When A now outputs a presentation token, B extracts the contained signature and attributes, and outputs this as a CL-forgery. It is easy to see that B wins whenever A wins, which can happen with at most negligible probability under the assumed UF-CMA security of CL-signatures.

Now, the claim of the theorem follows immediately from $(i) - (iii)$. □

Corollary 8.2.2. *Under the Strong RSA assumption (cf. Definition 2.2.6), the* PABS*-scheme described in § 7.3 is signature unforgeable according to Definition 4.3.2.*

Proof. This follows directly from Lemma 8.2.1 and the fact that the CL-signature scheme is existentially UF-CMA under the Strong RSA assumption [CL02]. □

Lemma 8.2.3. *The* PABS*-scheme described in § 7.3 satisfies key correctness according to Definition 4.3.3 in the random oracle model.*

Proof. This follows immediately from the soundness of the Fiat-Shamir heuristic [PS96] for the proof contained in *ipk*. □

Lemma 8.2.4. *Assuming that the underlying commitment scheme is hiding, the* PABS*-scheme described in § 7.3 is user private according to Definition 4.3.4.*

Proof. We prove this theorem through a sequence of games played with the issuance adversary $\mathsf{A}_{\mathtt{iss}} = \mathsf{A}_{1,\mathtt{iss}}, \mathsf{A}_{2,\mathtt{iss}}$ and presentation adversary $\mathsf{A}_{\mathtt{pres}} = \mathsf{A}_{1,\mathtt{pres}}, \mathsf{A}_{2,\mathtt{pres}}$. We first describe the simulators $\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}$ and $\mathsf{S}_{\mathtt{pres}}$.

$\mathsf{S}_{\mathtt{params}}$ The parameters are computed just as in the original $\mathsf{SPGen}_{\mathtt{s}}$, and the trapdoor is just the empty string.

$\mathsf{S}_{\mathtt{iss}}$ Acting as the user, the simulator computes all encryptions as encryptions to 0, and computes t as S^r for $r \xleftarrow{\$} [0, n]$. The simulator then uses the standard simulator for the Fiat-Shamir heuristic to produce a valid π_1, by programming the random oracle accordingly. Apart from that, it follows the protocol.

$\mathsf{S}_{\mathtt{pres}}$ The simulator computes $A' \xleftarrow{\$} QR_n, \mathfrak{c}_{a_j} = \mathsf{Enc}(0)$ for all $j \notin R, \mathfrak{c}_{o_j} = \mathsf{Enc}(0)$ for all $j \in C$, $\mathfrak{c}_e = \mathsf{Enc}(0)$, $\mathfrak{c}_{v'} = \mathsf{Enc}(0)$ and sets π' to simulates π' by programming the random oracle accordingly. Finally the simulator outputs $spt = (A', \mathfrak{c}_e, \mathfrak{c}_{v'}, (\mathfrak{c}_{a_j})_{j \notin R}, (\mathfrak{c}_{o_j})_{j \in C}, \pi')$.

Note that the first condition of Definition 4.3.4 is trivially satisfied as the simulated parameters are just the same as the real ones.

BlindIssuance. From $\mathsf{KeyVf}(ipk) = \mathsf{accept}$ we get that t unconditionally hides the unrevealed attributes, in both the real and the simulated world. Furthermore, by the IND-CPA security of the used encryption scheme, none of the computed ciphertexts leaks information about the contained values to the adversary in either world. Now, from the zero-knowledge property of the used non-interactive zero-knowledge proof of knowledge, we can infer that replacing the honest user by $\mathsf{S}_{\mathtt{iss}}$ cannot be detected by the adversary.

PtPrivacy. Similar to above, the distributions of signature presentation tokens computed in the real and the simulated world are computationally indistinguishable by the IND-CPA security of the used encryption scheme, the zero-knowledge property of the used non-interactive zero-knowledge proof of knowledge, and the fact that A' is statistically close to the uniform in the real world because of the well-formedness of *ipk*. □

8.3 Security Proofs of PABS-Scheme Based on Brands' Signatures

Lemma 8.3.1. *Assuming that UProve-signatures (cf. § 2.2.3) are existentially UF-CMA secure according to Definition 2.2.5, the* PABS*-scheme described in § 7.4 is signature unforgeable according to Definition 4.3.2 in the random oracle model.*

Proof. We proceed similar to the proof of Lemma 8.2.1, and first observe that, for Ω denoting the probability space as in Definition 4.3.2, we have that:

$$\begin{aligned}
&\Pr\Big[\mathsf{SignTokenVf}(ipk, spt, (a_i)_{i\in R}, (c_j)_{j\in C}, M) = \mathsf{accept} \ \wedge \ \Big(\mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} \ \vee \\
&\quad \exists k \in C : \mathsf{ComOpenVf}(a_k, c_k, o_k) = \mathsf{reject} \ \vee \ \big(((a_i)_{i\in R}, (c_j)_{j\in C}, M) \notin \mathcal{L}_{\text{spt}} \ \wedge \ \vec{a} \notin \mathcal{L}_{\text{iss}}\big)\Big) : \Omega\Big] \\
\leq\ &(i) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ \mathsf{SigVf}(sig, \vec{a}, ipk) = \mathsf{reject} : \Omega\Big] + \\
&(ii) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ \exists k \in C : \mathsf{ComOpenVf}(a_k, c_k, o_k) = \mathsf{reject} : \Omega\Big] + \\
&(iii) \quad \Pr\Big[\mathsf{SignTokenVf}(\dots) = \mathsf{accept} \ \wedge \ ((a_i)_{i\in R}, (c_j)_{j\in C}, M) \notin \mathcal{L}_{\text{spt}} \ \wedge \ \vec{a} \notin \mathcal{L}_{\text{iss}} : \Omega\Big].
\end{aligned}$$

We now show that each of the terms is negligible as a function of the security parameter.

For (i), note that signature presentation tokens are in particular non-interactive zero-knowledge proofs for $(a_i)_{i\notin R}$ with $a_i \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q$ such that:

$$h = \left(g_0 \prod_{i=1}^{L} g_i^{a_i}\right)^s,$$

and thus these values can be extracted from the adversary with overwhelming probability. Furthermore, $\mathsf{SignTokenVf}$ checks that $c' = \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'}(z')^{-c'})$. Together this is exactly what SignVf tests for.

For (ii), note that signature presentation tokens are in particular non-interactive zero-knowledge proofs for valid openings of $(c_j)_{j\in C}$. Therefore, the adversary only has a negligible chance of outputting commitments and a valid signature presentation token, although not knowing the content and openings of at least one commitment.

For (iii), we first modify the game the adversary has to win through a number of game hops, such that the winning probability of the adversary does not change by more than a negligible amount. We then show that if the adversary can win the last game of this sequence, then he can break the UF-CMA property of UProve-signatures which is hard by assumption.

Game 1: This is the original game specified in (iii).

Game 2: In this game, we additionally require that the extracted signature passes SigVf, and that valid openings of all commitments can be extracted. From (i) and (ii) it follows immediately that the winning probability of the adversary only changes negligibly.

Game 3: Requests to $\mathcal{O}^{\texttt{user}}$ are now simulated. That is, `obtain` requests are simply answered with a *cid*, and $(cid, \vec{a}, \varepsilon)$ is added to $\mathcal{L}_{\text{iss}}$. Upon receiving the first `token` request for some *cid*, c', r', z' are chosen uniformly at random in their domains, h is computed honestly, $a' = h^{r'}(z')^{-c'}$ and $b' = g^{r'} g_0^{-c'}$, and calls to H on input (h, z', a', b') are not forwarded to the external oracle any more, but answered with c'. In subsequent presentation requests for the same *cid*, those values are re-used. All ciphertexts are computed as encryptions to

zero. The required non-interactive zero-knowledge proof is then simulated programming the random oracle. By the CPA-security of the encryption scheme, the zero-knowledge property of the deployed proof, and the fact that the internal change of H will cause a collision with the real oracle for H only with negligible probability, this game is indistinguishable from the previous game for every adversary.

We now sketch how an adversary A winning this game can be used to construct an adversary B that forges UProve-signatures. Given a public UProve-key as input, B forwards it to A together with (self-chosen) system parameters. Calls to $\mathcal{O}^{\mathtt{user}}$ are treated as described in Game 3. Furthermore, for calls to $\mathcal{O}^{\mathtt{issuer}}$, B first extracts the values inside the commitments, sends them to the signing oracle, and then acts as a forwarder for the remaining messages of the protocol.

When A now outputs a presentation token, B extracts the contained s and the attributes $(a_i)_{i=1}^{L}$ and outputs the message $(a_i)_{i=1}^{L}$ and the signature (s, h, z', c', r'). It is easy to see that B wins whenever A wins, which can only happen with negligible probability assuming that UProve-signatures are weakly UF-CMA.

Now, the claim of the theorem follows immediately from $(i) - (iii)$. □

Lemma 8.3.2. *The* PABS*-scheme described in § 7.4 satisfies key correctness according to Definition 4.3.3.*

Proof. It is easy to see that the scheme is even perfectly key correct, i.e., the required negligibly function is equal to 0, as the subgroup of g and all g_i is unique, and thus there exist all required discrete logarithms. All the other properties of *ipk* follow trivially. □

Lemma 8.3.3. *Assuming that the underlying commitment scheme is hiding, the* PABS*-scheme described in § 7.4 is weakly user private according to Definition 4.3.5.*

Proof. We next describe the required simulators $\mathsf{S}_{\mathtt{params}}, \mathsf{S}_{\mathtt{iss}}$ and $\mathsf{S}_{\mathtt{pres}}$.

$\mathsf{S}_{\mathtt{params}}$ The parameters are computed just as in the original $\mathsf{SPGen}_{\mathtt{s}}$, and the trapdoor is just the empty string.

$\mathsf{S}_{\mathtt{iss}}$ Acting as the user, the simulator computes all encryptions as encryptions to 0, and computes t' as g^v for $v \xleftarrow{\$} \mathbb{Z}_q$. The simulator then uses the standard simulator for the Fiat-Shamir heuristic to produce a valid π, by programming the random oracle accordingly. Upon receiving the tuple (a, b, z), the simulator sends to the adversary a randomly chosen element $c \in \mathbb{Z}_q$.

$\mathsf{S}_{\mathtt{pres}}$ The simulator computes $\mathfrak{c}_{a_j} = \mathsf{Enc}(0)$ for all $j \notin R, \mathfrak{c}_{o_j} = \mathsf{Enc}(0)$ for all $j \in C$, and $\mathfrak{c}_s = \mathsf{Enc}(0)$. The simulator then checks whether an entry $(id_{sig}, z', c', r', h) \in \mathcal{L}$ exists, where $\mathcal{L}$ is an initially empty list, and id_{sig} is the pointer to the signature for which a presentation is to be simulated. If not, it chooses r', c', z', h uniformly at random in their respective domains, and programs H such that $c' = \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'}(z')^{-c'})$, and adds $(id_{sig}, z', c', r', h)$ to $\mathcal{L}$. Finally, the simulator fakes the proof π by programming the random oracle accordingly, and outputs $spt = (h, z', c', r', \mathfrak{c}_s, (\mathfrak{c}_{a_j})_{j \notin R}, (\mathfrak{c}_{o_j})_{j \in C}, \pi)$.

We now sketch why the given simulators satisfy the definition. The details of the proof are straightforward and hence omitted.

BlindIssuance. From $\mathsf{KeyVf}(ipk) = \mathsf{accept}$ we get that t' unconditionally hides the unrevealed attributes, in both the real and the simulated world. Furthermore, by the IND-CPA security of the used encryption scheme, none of the computed ciphertexts leaks information about the contained values to the adversary in either world. The user's second message c is uniformly

random in both worlds. Now, from the zero-knowledge property of the used non-interactive zero-knowledge proof of knowledge, we can infer that replacing the honest user by $\mathsf{S}_{\mathtt{iss}}$ cannot be detected by the adversary.

PtPrivacy. The values h, z', c', r' are constant for all presentation tokens derived from the same signature in both, the real and the ideal world. Apart from that, the distributions of the ciphertexts are indistinguishable by the IND-CPA security of the used encryption scheme, and the proofs contained in *spt* are indistinguishable by the zero-knowledge property of the used non-interactive zero-knowledge proofs. □

8.4 Security Proofs of Revocation Scheme

Lemma 8.4.1. *The revocation scheme described in § 7.5 is correct according to Definition 4.4.1.*

Proof. Let $RI = (t, \{\sigma_{t,j}\}_{j=0}^{t}, \vec{r}_t)$, and let the user want to show that some unrevoked a has not yet been revoked. Then by construction there exists a j such that $r_{t,j} < a < r_{t,j+1}$, as $\vec{r}_t$ only contains revoked attributes and $r_{\min}, r_{\max}$. Thus, the user can construct the required non-interactive zero-knowledge proof of knowledge. Correctness now follows from the correctness of this proof. □

Lemma 8.4.2. *Assuming that the deployed signature scheme is strongly unforgeable under chosen message attacks, then the revocation scheme described in § 7.5 is sound according to Definition 4.4.2 in the random oracle model.*

Proof. We prove the lemma by showing that a prover can only convince the verifier with negligible probability for each of the three given winning conditions.

Case a By construction, a revocation token rt is a zero-knowledge proof of knowledge of (α, o) such that $c = \mathsf{Com}(\alpha, o)$. It thus follows from the soundness of the deployed proof that $\mathsf{RevTokenVf}(rt, c, RI_{\mathsf{A}}, rpk) = \mathsf{accept} \wedge \mathsf{ComOpenVf}(c, a, o) = \mathsf{reject}$ can only happen with negligible probability.

Case b Assume that $\mathsf{RevTokenVf}(rt, c, RI_{\mathsf{A}}, rpk) = \mathsf{accept} \wedge \nexists\ (RI_{\mathsf{A}}, epoch, a') \in \mathcal{L}$. That is, the adversary outputs (RI_{A}, rt, c) such that the verifier accepts although the given revocation information was never generated by the revocation authority. Then the following algorithm D can be used to break the UF-CMA property of the underlying signature scheme. On input a signature verification key *rpk*, D behaves as follows:

- It computes $(spar_{\mathtt{r}}, \tau_{\mathtt{r}}) \xleftarrow{\$} \mathsf{E}_1^{\mathtt{r}}(1^\kappa, \lambda)$.
- It requests a signature $\sigma_{0,0}$ on message $(0, -2^{\ell_m}, 2^{\ell_m})$, and defines the initial revocation information as $RI_0 = (0, \{\sigma_{0,0}\}, (-2^{\ell_m}, 2^{\ell_m}))$.
- It then calls A on inputs $(rpk, RI_0, spar_{\mathtt{r}})$.
- For every call $(\mathtt{revoke}, a)$ to $\mathcal{O}^{\mathtt{revoke}}$, D computes the updated revocation information by requesting the required signatures from the signing oracle, thereby storing all tuples $(t, r_j, r_{j+1}, \sigma_{t,j})$.
- When A outputs (RI_{A}, rt, c) where $RI_{\mathsf{A}} = (t, \{\sigma_{t,j}\}_{j=0}^{t}, \vec{r})$ and all signatures verify correctly, D searches for a signature $\sigma_{t,k}$ for which no tuple $(t, r_k, r_{k+1}, \sigma_{t,k})$ has been recorded, and outputs message (t, r_k, r_{k+1}) together with the signature $\sigma_{t,k}$. If no such tuple exists or some signature does not verify, D outputs $\perp$.

It is easy to see that D does not output $\perp$ with more than negligible probability if A succeeds with more than negligible probability. Furthermore, one can see that D breaks the strong UF-CMA property of the signature scheme whenever it does not output $\perp$.

Case c Assume that $\mathsf{RevTokenVf}(rt, c, RI_{\mathsf{A}}, rpk) = \mathsf{accept} \wedge (\exists\ (RI_{\mathsf{A}}, epoch, a'), (RI', epoch', a) \in \mathcal{L} : epoch' \leq epoch)$. Then, by construction of the revocation information, we have that a is contained in $\vec{r}_{epoch}$. Now, as rt is in particular a proof that a lies between two subsequent entries of $\vec{r}_{epoch}$, this situation can only happen with negligible probability because of the soundness of the deployed zero-knowledge proof of knowledge.

The claim now follows immediately. □

Lemma 8.4.3. *The revocation scheme described in § 7.5 is private according to Definition 4.4.3, if the used encryption scheme is IND-CPA secure and the commitment scheme is hiding.*

Proof. As revocation tokens are zero-knowledge, they do not leak any information about the attribute, and the claim follows immediately. □

8.5 Security Proofs of Pseudonym Scheme

Lemma 8.5.1. *The pseudonym system described in § 7.6 is correct according to Definition 4.5.1.*

Proof. It is easy to see that $\Pr[nym \neq \mathsf{NymGen}(usk, scope) : \Omega]$ and $\Pr[\mathsf{NymVf}(spar_{\mathsf{p}}, c, o, scope) : \Omega = \mathsf{reject}]$ are both negligible, where Ω denotes the probability space from Definition 4.5.1. Indeed, the former probability is 0, as $\mathsf{NymPres}$ internally executes NymGen to obtain *nym*. The second probability is negligible by the correctness of the used non-interactive zero-knowledge proof of knowledge. The claim of the lemma now follows immediately. □

Lemma 8.5.2. *The pseudonym system described in § 7.6 is key extractable according to Definition 4.5.2 in the random oracle model.*

Proof. We have to show correctness, soundness and extractability, cf. Definition 2.2.2.

Correctness follows directly from the correctness of the Σ-protocol underlying the non-interactive zero-knowledge proof of knowledge, and soundness follows from the fact that the Fiat-Shamir heuristic is sound for the given relation in the random oracle model [PS96].

Finally, extractability is shown similar to the proof of Lemma 8.1.2. Again, the faked parameters are computed by E_1 just as the original ones, except that the decryption key of Enc is stored in τ. Now, given a c, *scope*, *nym* and π generated by the adversary, E_2 decrypts $\mathfrak{c}_o, \mathfrak{c}_{usk}$, obtaining o' and usk', respectively. Together with the correctness of the encryption scheme, the soundness property guarantees that the extracted witnesses are the same as those for c and *nym* with overwhelming probability. The claim now follows immediately. □

Lemma 8.5.3. *The pseudonym system described in § 7.6 is collision resistant according to Definition 4.5.3.*

Proof. We first note that because of $\mathcal{G}$ being a group of prime order q, every element different from 1 has order q as well.

Assume the adversary outputs *scope* such that $\mathsf{H}(scope) \neq 1$, but $\mathsf{H}(scope)^{usk_1} = \mathsf{H}(scope)^{usk_2}$. We then get that $\mathsf{H}(scope)^{usk_1 - usk_2} = 1$, and thus $usk_1 = usk_2$. On the other hand, if $\mathsf{H}(scope) = 1$ with more than negligible probability, the adversary broke the preimage resistance of the hash function, contradicting the security assumptions on H. □

Lemma 8.5.4. *Under the DDH assumption for $\mathcal{G}$ and the quadratic residuosity assumption, the pseudonym system described in § 7.6 is unlinkable according to Definition 4.5.4 in the random oracle model.*

Proof. We define a series of games, where each game is indistinguishable from the previous one.

Game 0: This is the original experiment as described in Figure 4.4.

Game 1: For this game, in $\mathcal{O}_0$ and $\mathcal{O}_1$, we replace all encryptions in $\mathsf{NymPres}$ by encryptions to 0, and simulate all the non-interactive zero-knowledge proofs using the standard simulator for the underlying Σ-protocols and programming the random oracle accordingly.

Using a hybrid argument, it is easy to see that the two games are indistinguishable by the zero-knowledge property of the deployed proof.

Game 2: Instead of computing c^* as a commitment to usk_b, in this game we set $(c^*, o^*) \xleftarrow{\$} \mathsf{Com}(0)$. Also, all commitments computed by $\mathcal{O}_0$ and $\mathcal{O}_1$ are computed as commitments to 0.

Again using a hybrid argument, one can see that this experiment is indistinguishable from Game 1 by the hiding property of the commitment scheme.

Game 3: Instead of computing nym^* as honestly, we compute it using a fresh and random usk^* instead.

Assume that there exists an algorithm D that can distinguish Games 2 and 3 with more than negligible probability. We then describe an algorithm D' breaking the DDH-assumption in $\mathcal{G}$, leading to a contradiction. Without loss of generality, we therefore assume that every call to a user oracle $\mathcal{O}_i$ is preceded by a call to the random oracle for the same scope, and that the random oracle is queried at most once for the each input.

On input (g, g^x, g^y, g^z) where either $z = xy$ or z is random, D' now behaves as follows, where q is a polynomial upper bound on the number of queries to the random oracle D does:

- In general, every call to the random oracle for *scope* is answered by $g^{r_{scope}}$, where $r_{scope} \xleftarrow{\$} \mathbb{Z}_q$, and the oracle stores $(scope, r_{scope})$. However, D' further randomly chooses two queries to the random oracle for some $scope'$ and $scope''$ which are answered with g and g^y, respectively.
- All queries to $\mathcal{O}_{1-b}$ are answered for an honestly computed pseudonym *nym* for some fixed and random usk_{1-b}.
- The pseudonyms for $\mathcal{O}_b$ are computed as follows: if the given $scope \neq scope'$, the simulated user oracle sets $nym_{scope} = (g^x)^{r_{scope}}$. For $scope'$, the pseudonym is set to g^x.
- The challenge pseudonym nym^* is defined as g^z if $scope^* = scope''$.
- If $scope'$ was never sent to $\mathcal{O}_b$, of if $scope^* \neq scope''$, D aborts and outputs a random bit. Otherwise, D' outputs 0 if D guesses the challenge bit correctly, and 1 otherwise.

It is easy to see that D' perfectly simulates the experiment of Game 2 if its input was a DDH-triple, and that of Game 3 if this was not the case. Thus, if the success probability of D was significantly different for these two games, then D' would break the DDH-assumption.

The claim of the lemma now follows immediately, as nym^*, c^* and π^*_{nym} are independent of the challenge bit b. □

Chapter 9

Conclusion

We provided security definitions, a modular construction, and secure instantiations of a privacy-enhancing attribute-based credential system. Our framework encompasses a rich feature set including multi-attribute credentials, multi-credential presentations, key binding, pseudonyms, attribute equality proofs, revocation, and advanced credential issuance with carried-over attributes. Prior to our work, most of these features found provably secure instantiations in isolation, but their combination into a bigger PABC system was never proved secure, nor was it clear which security notions such a combination should try to achieve.

Even though we think that our feature set is rich enough to cover a wide range of use cases (cf. § 1.2), there are more features that can be added to our framework. Among the features identified by the ABC4Trust project [CDL+13], this includes *inspection*, where presentation tokens can be de-anonymized by trusted inspectors; *verifier-driven revocation*, where verifiers can exclude certain credentials based on local blacklists; and attribute predicates, allowing to prove for example greater-than relations between attributes. Adding delegation would probably require a major overhaul of the framework.

Bibliography

[ABC] ABC4Trust – Attribute-based Credentials for Trust. EU FP7 Project. http://www.abc4trust.eu.

[BCC+09] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 09*, volume 5677 of *LNCS*, pages 108–125. Springer, 2009.

[BCKL08] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-Signatures and Non-interactive Anonymous Credentials. In R. Canetti, editor, *TCC 08*, volume 4948 of *LNCS*, pages 356–374. Springer, 2008.

[BG92] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In E. F. Brickell, editor, *CRYPTO 92*, volume 740 of *LNCS*, pages 390–420. Springer, 1992.

[BL13a] F. Baldimtsi and A. Lysyanskaya. Anonymous Credentials Light. In *ACM CCS 13*, pages 1087–1098. ACM, 2013.

[BL13b] F. Baldimtsi and A. Lysyanskaya. On the Security of One-Witness Blind Signature Schemes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 13*, volume 8270 of *LNCS*, pages 82–99. Springer, 2013.

[Blu13] BlueKrypt. Keylength – Cryptographic Key Length Recommendation, September 2013. http://www.keylength.com/.

[Bou00] F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In B. Preneel, editor, *EUROCRYPT 00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

[BP97] N. Barić and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In W. Fumy, editor, *EUROCRYPT 97*, volume 1233 of *LNCS*, pages 480–494. Springer, 1997.

[BPW12] D. Bernhard, O. Pereira, and B. Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In X. Wang and K. Sako, editors, *ASIACRYPT 12*, volume 7658 of *LNCS*, pages 626–643. Springer, 2012.

[Bra93] S. Brands. An Efficient Off-line Electronic Cash System Based On The Representation Problem. Technical report, 1993.

[Bra97] S. Brands. Rapid Demonstration of Linear Relations Connected by Boolean Operators. In W. Fumy, editor, *EUROCRYPT 97*, volume 1233 of *LNCS*, pages 318–333. Springer, 1997.

[Bra99] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates – Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, 1999.

[CDL+13] J. Camenisch, M. Dubovitskaya, A. Lehmann, G. Neven, C. Paquin, and F.-S. Preiss. Concepts and Languages for Privacy-Preserving Attribute-Based Authentication. In S. Fischer-Hübner, E. de Leeuw, and C. Mitchell, editors, *IDMAN 13*, volume 396 of *IFIP Advances in Information and Communication Technology*, pages 34–52. Springer, 2013.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Y. Desmedt, editor, *CRYPTO 94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CFT98] A. Chan, Y. Frankel, and Y. Tsiounis. Easy Come - Easy Go Divisible Cash. In K. Nyberg, editor, *EUROCRYPT 98*, volume 1403 of *LNCS*, pages 561–575. Springer, 1998.

[CG08] J. Camenisch and T. Groß. Efficient Attributes for Anonymous Credentials. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM CCS 08*, pages 345–356. ACM, 2008.

[CH02] J. Camenisch and E. Van Herreweghen. Design and Implementation of the *idemix* Anonymous Credential System. In V. Atluri, editor, *ACM CCS 02*, pages 21–30. ACM, 2002.

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Cha85] D. Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[Cha08] M. Chase. *Efficient Non-Interactive Zero-Knowledge Proofs for Privacy Applications*. PhD thesis, Brown University, Providence, Rhode Island, United States, 2008.

[CKM+14] J. Camenisch, S. Krenn, A. Lehmann G. L. Mikkelsen, G. Neven, and M. Ø. Pedersen. D3.1: Scientific Comparison of ABC Protocols. Part I - Formal Treatment of Privacy-Enhancing Credential Systems. Project deliverable in ABC4Trust, 2014.

[CKY09] J. Camenisch, A. Kiayias, and M. Yung. On the Portability of Generalized Schnorr Proofs. In A. Joux, editor, *EUROCRYPT 09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.

[CL01] J. Camenisch and A. Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In B. Pfitzmann, editor, *EUROCRYPT 01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[CL02] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[CL04] J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In M. K. Franklin, editor, *CRYPTO 04*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[CM99] J. Camenisch and M. Michels. Proving in Zero-Knowledge that a Number is the Product of two Safe Primes. In J. Stern, editor, *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 107–122. Springer, 1999.

[CMZ13] M. Chase, S. Meiklejohn, and G. M. Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. eprint, 2013/516, 2013. http://eprint.iacr.org/2013/516.

[Cor11] Microsoft Corporation. Proof of Concept on integrating German Identity Scheme with U-Prove technology, 2011. http://www.microsoft.com/mscorp/twc/endtoendtrust/vision/eid.aspx.

[CS97] J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In B. Kaliski, editor, *CRYPTO 97*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[DF02] I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In Y. Zheng, editor, *ASIACRYPT 02*, volume 2501 of *LNCS*, pages 125–142. Springer, 2002.

[DMS04] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The Second-Generation Onion Router. In M. Blaze, editor, *USENIX 04*, pages 303–320, 2004.

[FO97] E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In B. S. Kaliski Jr., editor, *CRYPTO 97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[FS87] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *CRYPTO 86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[GGM13] C. Garman, M. Green, and I. Miers. Decentralized Anonymous Credentials. ePrint, 2013/622, 2013. http://eprint.iacr.org/2013/622.

[GS08] J. Groth and A. Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. In N. P. Smart, editor, *EUROCRYPT 08*, volume 4965 of *LNCS*, pages 415–432. Springer, 2008.

[GS12] J. Groth and A. Sahai. Efficient Noninteractive Proof Systems for Bilinear Groups. *SIAM Journal on Computing*, 41(5):1193–1232, 2012.

[HK09] D. Hofheinz and E. Kiltz. The Group of Signed Quadratic Residues and Applications. In S. Halevi, editor, *CRYPTO 09*, volume 5677 of *LNCS*, pages 637–653. Springer, 2009.

[IRM] IRMA – I Reveal My Attributes. Research Project. https://www.irmacard.org.

[Lip03] H. Lipmaa. On Diophantine Complexity and Statistical Zero Knowledge Arguments. In C.-S. Laih, editor, *ASIACRYPT 03*, volume 2894 of *LNCS*, pages 398–415. Springer, 2003.

[NFHF09] T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki. Revocable Group Signature Schemes with Constant Costs for Signing and Verifying. In S. Jarecki and G. Tsudik, editors, *PKC 09*, volume 5443 of *LNCS*, pages 463–480. Springer, 2009.

[NP13] L. Nguyen and C. Paquin. U-Prove Designated-Verifier Accumulator Revocation Extension. TechReport MSR-TR-2013-87, 2013.

[Pai99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In J. Stern, editor, *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[Paq13] C. Paquin. U-Prove Cryptographic Specification V1.1 (Revision 3), 2013.

[Ped91] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *CRYPTO 91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[PotEU01] European Parliament and Council of the European Union. Regulation (EC) No 45/2001. Official Journal of the European Union, 2001.

[PotEU09] European Parliament and Council of the European Union. Directive 2009/136/EC. Official Journal of the European Union, 2009.

[PS96] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In U. Maurer, editor, *EUROCRYPT 96*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

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

[Sch91] C. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch10] H. A. Schmidt. National strategy for trusted identities in cyberspace. Cyberwar Resources Guide, Item 163, 2010. http://www.projectcyw-d.org/resources/items/show/163.

[SCO+01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero Knowledge. In J. Kilian, editor, *CRYPTO 01*, volume 2139 of *LNCS*, pages 566–598. Springer, 2001.

[Tea10] IBM Research Zurich Security Team. Specification of the Identity Mixer Cryptographic Library. IBM Technical Report RZ 3730 (99740), 2010.

[Zav13] G. Zaverucha. U-Prove ID escrow extension. TechReport MSR-TR-2013-86, 2013.