Concepts and Languages for Privacy-Preserving Attribute-Based Authentication[☆,☆☆]

Jan Camenisch[a,*], Maria Dubovitskaya[a,b], Robert R. Enderlein[a,b,**], Anja Lehmann[a], Gregory Neven[a], Christian Paquin[c], Franz-Stefan Preiss[a]

[a] *IBM Research – Zurich, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland*
[b] *Department of Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland*
[c] *Microsoft Research Redmond, One Microsoft Way, Redmond, WA 98052, United States*

Abstract

Existing cryptographic realizations of privacy-friendly authentication mechanisms such as anonymous credentials, minimal disclosure tokens, self-blindable credentials, and group signatures vary largely in the features they offer and in how these features are realized. Some features such as revocation or de-anonymization even require the combination of several cryptographic protocols. The variety and complexity of the cryptographic protocols hinder the understanding and hence the adoption of these mechanisms in practical applications. They also make it almost impossible to change the underlying cryptographic algorithms once the application has been designed. In this paper, we aim to overcome these issues and simplify both the design and deployment of privacy-friendly authentication mechanisms. We define and unify the concepts and features of privacy-preserving attribute-based credentials (Privacy-ABCs), provide a language framework in XML schema, and present the API of a Privacy-ABC system that supports all the features we describe. Our language framework and API enable application developers to use Privacy-ABCs with all their features without having to consider the specifics of the underlying cryptographic algorithms—similar to as they do today for digital signatures, where they do not need to worry about the particulars of the RSA and DSA algorithms either.

Keywords: Authentication, privacy, data-minimization, anonymous credentials, digital credentials

[☆] This is an accepted author manuscript posted for scholarly purposes. The final version is available at `http://dx.doi.org/10.1016/j.jisa.2014.03.004`.

[☆☆] This is an extended version of a paper that appeared at the 3rd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management (IFIP IDMAN 2013). The original publication is available at `link.springer.com`.

[*] Authors sorted alphabetically, all authors contributed equally.

[**] Corresponding author.

Email addresses: `jca at zurich dot ibm dot com` (Jan Camenisch), `mdu at zurich dot ibm dot com` (Maria Dubovitskaya), `enr at zurich dot ibm dot com` (Robert R. Enderlein), `anj at zurich dot ibm dot com` (Anja Lehmann), `nev at zurich dot ibm dot com` (Gregory Neven), `cpaquin at microsoft dot com` (Christian Paquin), `frp at zurich dot ibm dot com` (Franz-Stefan Preiss)

1. Introduction

More and more transactions in our daily life are performed electronically and the security of these transactions is an important concern. Strong authentication and according authorization based on certified attributes of the requester is paramount for protecting critical information and infrastructures online.

Most existing techniques for transferring trusted user attributes cause privacy concerns. In systems where an online identity provider creates access tokens on demand, such as SAML, OpenID, or WS-Federation, the identity provider can impersonate its users and can track a user's moves online. Systems with offline token creation, such as X.509 certificates and some WS-Trust profiles, force the user to reveal more attributes than strictly needed (as otherwise the issuer's signature cannot be verified) and make her online transactions linkable across different websites.

These drawbacks can be overcome with privacy-preserving authentication mechanisms based on advanced cryptographic primitives such as anonymous credentials, minimal disclosure tokens, self-blindable credentials, or group signatures [19, 12, 24, 28, 7, 58]. In these schemes, users obtain certified credentials for their attributes from trusted issuers and later derive, without further assistance from any issuer, unlinkable tokens that reveal only the required attribute information yet remain verifiable under the issuer's public key. Well-known examples include Brands' scheme [12] and Camenisch-Lysyanskaya's scheme [27], which have been implemented in Microsoft's U-Prove [57] and IBM's Identity Mixer [40], respectively. Both implementations are freely available and efficient enough for practical use, yet the real-world adoption is slower than one may hope.

The various schemes described in the literature offer a large variety of features. Similar features are often referred to by different names or are realized with different cryptographic mechanisms. Many of the features such as credential revocation, efficient attribute encoding, or anonymity lifting even require a combination of several cryptographic protocols. This makes these technologies very difficult to understand, compare, and use.

We overcome these difficulties by providing unified definitions of the concepts and features of the different privacy-preserving authentication mechanisms. We will refer to this unification as *privacy-preserving attribute-based credentials* or *Privacy-ABCs*. Our definitions abstract away from the concrete cryptographic realizations but are carefully crafted so that they can be instantiated with different cryptographic protocols—or a combination of them. To enable the use and integration of Privacy-ABCs in authentication and authorization systems, we further present a cryptography-agnostic language framework and application programming interface (API) with well-documented data formats for credentials, policies, and claims. All languages are specified in XML schema and separate the abstract functionality expected from the underlying cryptographic mechanisms from the opaque containers for the cryptographic data itself. The API is based on the reference implementation of the ABC4Trust project and enables an easy integration of a Privacy-ABC system into existing applications. Our languages and API allow application developers to employ Privacy-ABCs without having to think about their cryptographic realization, similarly to how common cryptographic primitives such as encryption and signatures are used today: the application layer calls out to the cryptography through standardized interfaces; the concrete chosen algorithm is at most an initialization parameter.

The language described in this paper has been implemented in the ABC4Trust project [1] and will be made available as part of a reference implementation of a Privacy-ABC system which will include a number of cryptographic solutions. The full language description and schema are available as a project deliverable [16, 17].

2. Related work

Our work builds on the credential-based authentication requirements language (CARL) recently proposed by Camenisch et al. [30]. CARL allows a service provider (verifier) to specify which attributes a user needs to present, and by which issuer these attributes need to be certified, in order to get access. Compared to our work, CARL defines only a small part of a Privacy-ABC system, namely the presentation policy, but does not consider how these attributes are transmitted nor how credentials are issued or revoked. Bichsel et al. [8] have extended CARL to cover the transmission of certified attributes. The current version of the U-Prove protocol [57] natively provides a subset of features of our framework; the other features need to be added through extension points.

Privacy-ABCs can be used to realize a privacy-respecting form of attribute-based access control. Traditional attribute-based access control [9, 61, 59], however, does not see attributes as grouped together in a credential or token. Thus our framework allows one to realize more specific and more precise access control policies. Also, role-based access control [36, 54] (RBAC) can be seen as a special case of our attribute-based setting by encoding the user's roles as attributes. Recent work [43] extended RBAC with privacy-preserving authentication for the particular case of role and location attributes.

Bonatti and Samarati [9] also propose a language for specifying access control rules based on "credentials". The language focuses on credential ownership and does not allow for more advanced requirements such as for example revealing of attributes, signing statements, or inspection. The same is true for the languages proposed by Ardagna et al. [3] and by Winsborough et al. [60]. However, the latter allows one to impose attribute properties on credentials and its extension by Li et al. [46] supports revealing of attributes. The Auth-SL language [53] focuses on multi-factor authentication and enables the policy author to specify restrictions on the properties of the authentication mechanisms themselves, but not on attributes of individual users.

The language by Ardagna et al. [2] can also be considered as a predecessor to our language in the sense that it focuses on anonymous credential systems and some of the advanced features. However, it considers only the presentation phase and is less expressive than ours, for instance, it cannot express statements involving attributes from different credentials.

VeryIDX [50] is a system to prevent identity theft by permitting the use of certain identity attributes only in combination with other identity attributes. So-called verification policies specify which attributes have to be presented together. However, these policies are introduced only conceptually without any details on exact expressivity, syntax, or semantics.

Several logic-based and technology-neutral approaches to distributed access control have been proposed [4, 6, 38, 45]. However, none of these have been designed with Privacy-ABCs in mind. In particular, they do not support selective disclosure of attributes, proving predicates over attributes, or attribute inspection.

Summarized, our language framework is the first that covers the whole life-cycle of Privacy-ABCs and also the first one unifying the full spectrum of their features.

3. Example Scenario

In this section we describe a scenario using Privacy-ABCs. We will refer back to this scenario when we describe the concepts and features of Privacy-ABCs in Section 4, when we describe the language framework in Section 5, and the API in Section 6.

To illustrate the usage of Privacy-ABCs we consider the following scenario. The Republic of Utopia issues electronic identity cards to all of its citizens, containing their name, date of birth, and the state in which they reside. These electronic identities are used for many applications, such as interactions with government and businesses. It is therefore crucial that any card that is reported lost or stolen will be quickly revoked.

All citizens of Utopia may sign up for one free digital membership card to the library of their state. To obtain a library card, the applicant must present her valid identity card and reveal her state of residence, but otherwise remains anonymous during the issuance of the library card.

The state library has a privacy-friendly online interface for borrowing both digital and paper books. Readers can log in to the library website to anonymously browse and borrow books using their library card based on Privacy-ABCs. Hardcopy books will be delivered in anonymous numbered mailboxes at the post office; digital books are simply delivered electronically. If paper books are returned late or damaged, however, the library must be able to identify the reader to impose an appropriate fine. Repeated negligence can even lead to exclusion from borrowing further paper books—but borrowing digital books always remains possible. Moreover, the library occasionally offers special conditions to readers of targeted age groups, e.g., longer rental periods for readers under the age of twenty-six.

4. Concepts and Features

Figure 1 gives an overview of the entities involved in Privacy-ABC systems and the interactions between them. The interactions are named according to their purpose. Depending on the technical realizations, these interactions will be realized differently and might occur multiple times using different protocols (we consider sending a single message also a protocol). These entities are *users*, *issuers*, *verifiers*, *inspectors* and *revocation authorities*.

In our scenario, the users are the citizens of Utopia. The government of the Republic of Utopia acts as an issuer for the digital identity cards, while the state libraries act as issuers for the library cards. Both the government and the library act as their own

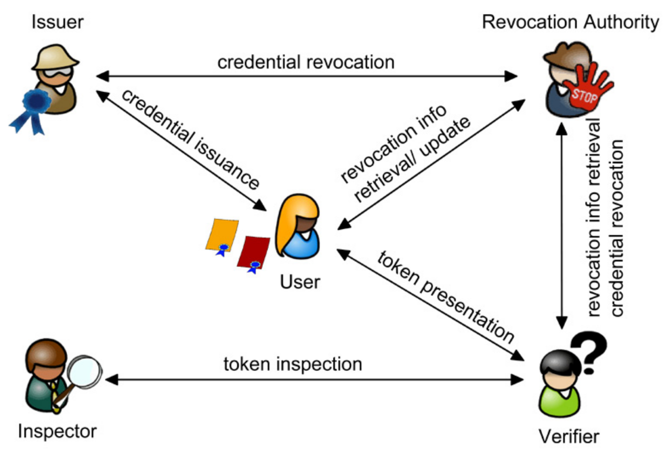

Figure 1: Entities and the interactions between them.

revocation authorities. (In fact, the roles of issuer and revocation authority can often be assumed by the same entity.) The online book-borrowing service is the verifier. A set of trusted arbitrators is appointed to confirm when books are brought back late or damaged and, if this is the case, to reveal the identity of the negligent reader.

Each issuer generates a secret issuance key and publishes the *issuer parameters* that include the corresponding public verification key. Similarly, each inspector generates a private decryption key and publishes the *inspector parameters* that include the corresponding public encryption key, and each revocation authority generates and publishes its *revocation authority parameters*. We assume that all entities have means to retrieve the parameters of the issuers, revocation authorities, and inspectors. Users get issued credentials by issuers via the *credential issuance* protocol. A credential contains attributes that its issuer vouches for with respect to the user. A credential can also specify one or more revocation authorities who are able to revoke the credential if necessary for some reason (e.g., loss or theft). To issue a credential that is revocable, the user and/or the issuer might need to interact with the revocation authority prior to or during the issuance protocol. Using her credentials, a user can form a presentation token that contains a subset of the certified attributes, provided that the corresponding credentials have not been revoked. This process might require the user to retrieve information from the revocation authority. Additionally, some of the attributes can be encoded in the presentation token so that they can only be retrieved by an inspector. The user can attach inspection grounds specifying under which conditions the inspector should reveal these attributes. Receiving a presentation token from a user, a verifier checks whether the presentation token is valid with respect to the relevant issuer parameters and inspector parameters and the latest revocation information (thus, the verifier will interact with the revocation authority). If the verification succeeds, the verifier will be convinced that the attributes contained in the presentation token are vouched for by the corresponding issuers. Finally, if a presentation token contains attributes that can only be retrieved by an inspector and the inspection grounds are met, the verifier can interact with the inspector to learn these attributes.

Informally, a secure realization of a Privacy-ABC system guarantees that (1) users can only generate a valid presentation token if they were indeed issued the corresponding credentials that have not been revoked, (2) that attributes encoded in the presentation token for an inspector can indeed be retrieved by that inspector, and (3) that the presentation tokens do not reveal any further information about the users other than the attributes contained in them.

We now provide a brief explanation of the main features supported by Privacy-ABCs, with a focus on the ones that were not modeled so far in existing identity frameworks. See also Table 1 for an overview of these features.

Feature	**Description**
Regular pseudonym	A "public key" derived from a user's secret key, but so that a user can generate many unlinkable regular pseudonyms from a given secret key.
Scope-exclusive pseudonym	A pseudonym that is unique for a given scope string and secret key, so that one user secret can only be used to generate a single scope-exclusive pseudonym for each scope string.
Credential	A list of attribute/value pairs certified by an issuer.
+ Carried-over attributes	Attributes in a credential that are copied from other credentials during issuance, and that the issuer certifies without seeing their values.
+ Key binding	Including a user secret key as an attribute in a credential. Pseudonyms are always bound to a user secret key.
+ Issuer-driven revocation	Allows for credentials to be revoked globally.
Presentation	Proving the possession of pseudonyms or credentials while revealing only a subset of the attributes in the proof. It is also possible to perform proofs of equality among attributes and user secret keys.
+ Key binding	Prove that credentials used in the presentation are bound to the same user secret key.
+ Predicate	Prove that an attribute is smaller, not equal, or greater than a constant or another attribute.
+ Inspection	During a presentation, additionally encrypt an attribute under the public key of an inspector and prove that the correct attribute was encrypted. The inspector is trusted to decrypt the ciphertext only under specific, mutually agreed circumstances.
+ Verifier-driven revocation	Allows for revoking a specific combination of attributes for certain transactions.

Table 1: Glossary of the main features supported by Privacy-ABCs.

4.1. Pseudonyms

Each user can generate a secret key. However, unlike traditional public-key authentication schemes, there is no single public key corresponding to the secret key. Rather, the user can generate as many public keys as she wishes. These public keys are called *pseudonyms* in Privacy-ABCs. Pseudonyms [19, 47] are cryptographically unlinkable, meaning that given two different pseudonyms, one cannot tell whether they were generated from the same or from different secret keys. By generating a different pseudonym for every verifier, users can thus be known under different unlinkable pseudonyms to different sites, yet use the same secret key to authenticate to all of them.

While it is sufficient for users to generate a single secret key, they can also have multiple secret keys. A secret key can be generated by a piece of trusted hardware (e.g., a smart card) that stores the key and uses it in computations (e.g., to generate pseudonyms), but that never reveals the key. The key is thereby *bound* to the hardware, in the sense that it can only be used in combination with the hardware.

There are situations, however, where the possibility to generate an unlimited number of unlinkable pseudonyms is undesirable. For example, in an online opinion poll, users should not be able to bias the result by entering multiple votes under different pseudonyms. In such situations, the verifier can request a special pseudonym called a *scope-exclusive pseudonym*, which is unique for a given user secret key and a given *scope string* [39]. Scope-exclusive pseudonyms for different scope strings remain unlinkable. By using the URL of the opinion poll as the scope string, for example, the verifier can ensure that each user can only register a single pseudonym to vote, but users remain unlinkable across different polls. In our example scenario, scope-exclusive pseudonyms can be used to ensure that each citizen of Utopia can only obtain a single library card for the library of his own state.

4.2. Credentials and Key Binding

A *credential* is a certified container of attributes issued by an issuer to a user. Formally, an *attribute* is described by the *attribute type* that determines the semantics of the attribute (e.g., first name) and the *attribute value* that determines its contents (e.g., "John"). By issuing a credential, the issuer vouches for the correctness of the contained attributes with respect to the user. The *credential specification* lists the attribute types that are encoded in a credential. A credential specification can be created by the issuer, or by an external authority so that multiple issuers can issue credentials according to the same specification. The credential specification must be published and distributed over a trusted channel. How exactly this is done goes beyond the scope of our language framework; the specification could for example be digitally signed by its creator.

Optionally, a credential can be *bound* to a user's secret key, i.e., it cannot be used without knowing the secret key [47]. We call this option *key binding*. It is somewhat analogous to traditional public-key certificates, where the certificate contains the CA's signature on the user's public key, but unlike traditional public-key certificates, a Privacy-ABC is not bound to a unique public key: it is only bound to a unique secret key. A user can derive as many pseudonyms as she wishes from this secret key and (optionally) show that they were derived from the same secret key that underlies the credential. In our scenario, both the identity and the library card are credentials with key binding.

4.3. Presentation

To authenticate to a verifier, the user first obtains the *presentation policy* that describes which credentials the user must present and which information from these credentials she must reveal. If the user possesses the necessary credentials, she can derive from these credentials a *presentation token* that satisfies the presentation policy. The presentation token can be verified using the issuer parameters of all credentials underlying the presentation token.

Presentation tokens derived from Privacy-ABCs only reveal the attributes that were explicitly requested by the presentation policy – all the other attributes contained in the credentials remain hidden. Moreover, presentation tokens are cryptographically unlinkable (meaning no collusion of issuers and verifiers can tell whether two presentation tokens were generated by the same user or by different users) and untraceable (meaning that no such collusion can correlate a presentation token to the issuance of the underlying credentials). This is exactly what is required in our scenario when citizens want to borrow books. Of course, presentation tokens are only as unlinkable as the information they intentionally reveal.

Rather than requesting and revealing full attribute values, presentation policies and tokens can also request and reveal *predicates* over one or more issued attributes. For example, a token could reveal that the name on the user's credit card matches that on her driver's license, without revealing the name. In our scenario, a presentation token can reveal that the borrower's date of birth is before April 1st, 1988 (making her eligible as a reader under twenty-six), without revealing her exact date of birth. Finally, the policy can also request that certain credentials and pseudonyms are bound to the same user secret key.

4.4. Issuance

In the simplest setting, an issuer knows all attribute values to be issued and simply embeds them into a credential.

Privacy-ABCs also support advanced issuance features where attributes are blindly "carried over" from existing credentials, without the issuer becoming privy to their values. Similarly, the issuer can blindly issue self-claimed attribute values (i.e., not certified by an existing credential), carry over the secret key to which a credential is bound, or assign a uniformly random value to an attribute such that the issuer cannot see it and the user cannot bias it [12, 27]. In our scenario, identity cards and library cards must be issued with advanced issuance: the identity card must include the secret key of the user; and the library card must carry over the full name and secret key of the requestor from the identity card.

Advanced issuance is an interactive protocol between the user and the issuer. In the first move, the issuer provides the user with an *issuance policy* that consists of a presentation policy specifying which pseudonyms and/or existing credentials the user must present, and of a *credential template* specifying which attributes or secret keys of the newly issued credential will be generated at random or carried over from credentials or pseudonyms in the presentation policy. In response, the user sends an *issuance token* containing a presentation token that satisfies the issuance policy. Then the (possibly multi-round) cryptographic issuance protocol ensues, at the end of which the user obtains the new credential.

4.5. Inspection

Absolute user anonymity in online services easily leads to abuses such as spam, harassment, or fraud. Privacy-ABCs provide the option to add accountability for misbehaving users through a feature called *inspection* [13, 31]. Here, a presentation token contains one or more credential attributes that are encrypted under the public key of a trusted *inspector*. The verifier can check that the correct attribute values were encrypted, but cannot see their actual values. The *inspection grounds* describe the circumstances under which the verifier may call upon the inspector to recover the actual attribute values. The inspector is trusted to collaborate only when the inspection grounds have been met; verifiers cannot change the inspection grounds after receiving a presentation token, as the grounds are cryptographically tied to the token.

The presentation policy specifies which attributes from which credentials have to be encrypted under which inspector parameters and which inspection grounds. In our library example, the library website includes as part of the presentation policy for borrowing paper books that the full name as stated on the library card must be encrypted to one of the trusted arbitrators, with as inspection grounds the late or damaged return of the book. In case this happens, the library hands the presentation token as well as any the evidence of the abuse to the arbitrator, who will then decrypt the reader's name.

4.6. Revocation

Credentials may need to be revoked for several reasons: the credential and the related user secrets may have been compromised, the user may have lost her right to carry a credential, or some of her attribute values may have changed. In such cases, credentials need to be revoked globally and we call this *issuer-driven revocation*. The Utopian identity cards fall under this category.

Sometimes credentials may be revoked only for specific contexts. For example, a hooligan may see his digital identity card revoked for accessing sport stadiums, but may still use it for all other purposes; or the Utopian state library may wish to deny lending paper book to borrowers who fail to properly return their books. We call this *verifier-driven revocation*.

Revocation for Privacy-ABCs is cryptographically more complicated than for classical certificates, but many mechanisms with varying efficiency [44] exist [26, 49, 11, 21, 48]. Bar a few exceptions, all of them can be used for both issuer-driven and verifier-driven revocation.

We describe revocation in a generic mechanism-agnostic way and consider credentials to be revoked by dedicated *revocation authorities*. They are separate entities in general, but may be under the control of the issuer or verifier in particular settings. The revocation authority publishes static *revocation authority parameters* and periodically publishes the most recent *revocation information*. When creating presentation tokens, users prove that their credentials have not been revoked, possibly using *non-revocation evidence* that they fetch and update from the revocation authority. The revocation authority to be used is specified in the issuer parameters for issuer-driven revocation and in the presentation policy for verifier-driven revocation. When a credential is subject to issuer-driven revocation, a presentation token related to this credential must always contain a proof that the presented credential has not been revoked. Issuer-driven revocation is performed based on the *revocation handle*, which is a dedicated unique attribute embedded in a credential. Verifier-driven revocation can be performed based on any combination of attribute values, possibly even from different credentials. This allows the revocation authority for example to exclude certain combinations of first names and last names to be used in a presentation token.

4.7. Cryptographic Realization

At the core of a Privacy-ABC system is a signature scheme with efficient protocols to interactively sign a set of messages and to prove possession of signatures. The signatures can then act as Privacy-ABC credentials, with the attributes making up the set of messages. The protocols are needed for a number of reasons. First, users cannot simply reveal the signature to a verifier as part of a presentation token, because then different presentations become linkable. Instead, the user will perform a zero-knowledge proof of knowledge of a valid signature so that the privacy properties of a Privacy-ABC system can be achieved. Second, a signature scheme needs to support the signing of a set of messages so that attributes can be selective disclosed in presentation tokens. Third, features such as key binding can be realized by including users' secret keys as an attribute into the credentials. To prevent the issuer from learning the user secret during issuance, however, the issuer must be able to sign messages without learning some of them.

While interactive issuance protocols and zero-knowledge proofs can be built for any signature scheme using generic techniques, this approach is not efficient and would hardly lead to practically useful Privacy-ABC systems. Fortunately, there exist a few signature schemes that allow for efficient issuance and prove protocols. These include the Camenisch-Lysyanskaya [27] and the Brands [12] signature schemes, on which Identity Mixer and U-Prove are built on, respectively, and some schemes that employ bilinear maps [28, 5]. Thus, an issued credential is realized as a signature on the user's attributes.

Ordinary pseudonyms can be realized with cryptographic commitments [51, 33] to a user's secret key; scope-exclusive pseudonyms can be realized as the output of a verifiable random function [34, 20] applied to the secret key as seed and the scope string as input. As mentioned above, key binding can be realized by designating one attribute as a user secret key (which of course must never be revealed). Inspection can be obtained through verifiable encryption [31] of the inspectable attributes. Revocation of Privacy-ABCs can be done through signed revocation lists [48], through dynamic accumulators [26, 49, 21], or through efficient updates of short-lived credentials [22]. The "glue" binding all primitives together in a presentation token is provided by zero-knowledge proofs of knowledge of discrete logarithms [52, 23] made non-interactive through the Fiat-Shamir transform [37]. These proofs are also used for equality predicates over attributes; inequality predicates are proved with range proofs [10, 14].

ABC4Trust provides an implementation [17] that combines all these cryptographic building blocks and thus supports all the concepts presented in this section. The ABC4Trust implementation is an extension of IBM's Identity Mixer [40] that also integrates U-Prove [57] signatures. U-Prove [57] natively supports a subset of the features of the ABC4Trust framework; the other features need to be added through extension points.

There are a few other cryptographic primitives that realize some of the concepts discussed in this paper. For instance group signatures [13] and identity escrow schemes [42] can be seen as attribute-less credentials with inspection and key binding. Similarly, direct anonymous attestation [18] can also be seen as attribute-less credentials with key-binding and revocation. Anonymous attribute tokens [29] are credentials that support attributes.

5. Language Framework

Given the multitude of distributed entities involved in a full-fledged Privacy-ABC system, the communication formats that are used between these entities must be specified and standardized.

None of the existing format standards for identity management protocols such as SAML, WS-Trust, or OpenID support all Privacy-ABCs' features. Although most of them can be extended to support a subset of these features, we define for the sake of simplicity and completeness a dedicated language framework which addresses all unique Privacy-ABC features. Our languages can be integrated into existing identity management systems.

In this section we introduce our framework covering the full life-cycle of Privacy-ABCs, including setup, issuance, presentation, revocation, and inspection. As the main purpose of our data artifacts is to be processed and generated by automated policy and credential handling mechanisms, we define all artifacts in XML schema notation, although one could also create a profile using a different encoding such as Abstract Syntax Notation One (ASN.1) [41] or JavaScript Object Notation (JSON) [32].

The XML artifacts formally describe and orchestrate the underlying cryptographic mechanisms and provide opaque containers for carrying the cryptographic data. Whenever appropriate, our formats also support user-friendly textual names or descriptions which allow to show a descriptive version of the XML artifacts to a user and to involve her in the issuance or presentation process if necessary.

For didactic purposes we describe the different artifacts realizing the concepts from Section 4 by means of examples for our scenario from Section 3. For the sake of space and readability, these examples do not illustrate all features described in the previous section; we refer the reader to [16, 17] for the full specification. In what follows, we explicitly distinguish between user attributes (as contained in a credential) and XML attributes (as defined by XML schema) whenever they could be confused.

5.1. Credential Specification

Recall that the credential specification describes the common structure and possible features of credentials. Remember that the Republic of Utopia issues electronic identity cards to its citizens containing their full name, state, and date of birth. Note that libraries and other verifiers may target different age groups in different policies, so hard-coding dedicated "over twenty-six" attributes would not be very sensible. Utopia may issue Privacy-ABCs according to the credential specification shown in Figure 2.

The XML attribute **KeyBinding** indicates whether credentials adhering to this specification must be bound to a secret key. The XML attribute **Revocable** being set to *"true"* indicates that the credentials will be subject to issuer-driven revocation and hence must contain a special revocation handle attribute. The assigned revocation authority is specified in the issuer parameters.

```
 1 <CredentialSpecification KeyBinding="true" Revocable="true">
 2   <SpecificationUID> urn:creds:id </SpecificationUID>
 3   <AttributeDescriptions MaxLength="256">
 4     <AttributeDescription Type="urn:creds:id:name" DataType="xs:string" Encoding="xenc:sha256">
 5       <FriendlyAttributeName lang="EN"> Full Name </FriendlyAttributeName>
 6     </AttributeDescription>
 7     <AttributeDescription Type="urn:creds:id:state" DataType="xs:string" Encoding="xenc:sha256"/>
 8     <AttributeDescription Type="urn:creds:id:bdate" DataType="xs:date" Encoding="date:unix:signed"/>
 9     <AttributeDescription Type="urn:revocationhandle" DataType="xs:integer" Encoding="integer:unsigned" />
10   </AttributeDescriptions>
11 </CredentialSpecification>
```

Figure 2: Credential specification of the identity card.

To encode user attribute values in a Privacy-ABC, they must be mapped to integers of a limited length. The maximal length depends on the security parameter (basically, it is the bit length of exponents in the group) and is indicated by the **MaxLength** XML attribute (Line 3), here 256 bits. In our example, electronic identity cards contain a person's full name, state, and date of birth. The XML attributes **Type**, **DataType**, and **Encoding** respectively contain the unique identifier for the user attribute type, for the data type, and for the encoding algorithm that specifies how the value is to be mapped to an integer of the correct size (Lines 4,7,8,9). Attributes that may have values longer than **MaxLength** have to be hashed, as is done here for the name using SHA-256. The specification can also define human-readable names for the user attributes in different languages (Line 5).

5.2. Issuer, Revocation, and System Parameters

The government of Utopia acts as issuer and revocation authority for the identity cards. It generates an issuance key pair and publishes the issuer parameters, and generates and publishes the revocation authority parameters, which are illustrated in Figure 3.

The **ParametersUID** element assigns unique identifiers for the issuer and revocation authority parameters. The issuer parameters additionally specify the chosen cryptographic Privacy-ABC and hash algorithm, the maximal number of attributes that credentials issued under these issuer parameters may have, the parameter identifier of the system parameters that shall be used, and the parameters identifier of the revocation authority that will manage the issuer-driven revocation. The **CryptoParams** contain cryptographic algorithm-specific information about the public key.

The revocation authority parameters can be used for both issuer- and verifier-driven revocation. They specify a unique identifier for the parameters, the cryptographic revocation mechanisms, and references to the network endpoints where the most recent revocation information and non-revocation evidence can be fetched.

The system parameters fix some cryptographic parameters that are needed by the Privacy-ABC system as a whole, such as the overall security level and the groups that are to be used with the pseudonyms. Every party in the Privacy-ABC system must use the same system parameters to ensure compatibility. Any trusted issuer can create fresh system parameters, but ideally system parameters should be standardized.

```
<IssuerParameters>
  <ParametersUID> urn:utopia:id:issuer </ParametersUID>
  <AlgorithmID> urn:com:microsoft:uprove </AlgorithmID>
  <SystemParametersUID> urn:utopia:id:system </SystemParametersUID>
  <MaximalNumberOfAttributes> 4 </MaximalNumberOfAttributes>
  <HashAlgorithm> xenc:sha256 </HashAlgorithm>
  <CryptoParams> ... </CryptoParams>
  <RevocationParametersUID> urn:utopia:id:ra </RevocationParametersUID>
</IssuerParameters>
```

```
<RevocationAuthorityParameters>
  <ParametersUID> urn:utopia:id:ra </ParametersUID>
  <RevocationMechanism> urn:privacy−abc:accumulators:cl </RevocationMechanism>
  <RevocationInfoReference ReferenceType="url"> https:utopia.gov/id/revauth/revinfo
    </RevocationInfoReference>
  <NonRevocationEvidenceReference ReferenceType="url"> https:utopia.gov/id/revauth/nrevevidence
      </NonRevocationEvidenceReference>
  <CryptoParams> ... </CryptoParams>
</RevocationAuthorityParameters>
```

```
<SystemParameters>
  <ParametersUID> urn:utopia:id:system </ParametersUID>
  <CryptoParams> ... </CryptoParams>
</SystemParameters>
```

Figure 3: Issuer, revocation authority, and system parameters.

5.3. Presentation Policy with Basic Features

Assume that a user already possesses an identity card from the Republic of Utopia issued according to the credential specification depicted in Figure 2. To get her free library card the user must present her valid identity card and reveal (only) the state attribute certified by the card. This results in the presentation policy depicted in Figure 4.

```
<PresentationPolicy PolicyUID="libcard">
  <Message>
    <Nonce> bkQydHBQWDR4TUZzbXJKYUM= </Nonce>
  </Message>
  <Pseudonym Alias="nym" Scope="urn:library:issuance" Exclusive="true"/>
  <Credential Alias="id" SameKeyBindingAs="nym">
    <CredentialSpecAlternatives>
      <CredentialSpecUID> urn:creds:id </CredentialSpecUID>
    </CredentialSpecAlternatives>
    <IssuerAlternatives>
      <IssuerParametersUID> urn:utopia:id:issuer </IssuerParametersUID>
    </IssuerAlternatives>
    <DisclosedAttribute AttributeType= "urn:creds:id:state"/>
  </Credential>
</PresentationPolicy>
```

Figure 4: Presentation policy for an identity card.

We now go through the preceding presentation policy and describe how the different features of Privacy-ABCs can be realized with our language. We first focus on the basic features and describe extended concepts such as inspection and revocation in our second example.

Signing Messages. A presentation token can optionally sign a message. The message to be signed is specified in the policy (Figure 4, Lines 2–4). It can include a nonce, any application-specific message, and a human-readable name and/or description of the policy. The nonce will be used to prevent replay attacks, i.e. to ensure freshness of the presentation token, and for cryptographic evidence generation. Thus, when making use of the nonce, the presentation policy is not static anymore, but needs to be completed with a fresh nonce element for every request.

Pseudonyms. The optional **Pseudonym** element (Figure 4, Line 5) indicates that the presentation token must contain a pseudonym. A pseudonym can be presented by itself or in relation with a credential if key binding is used (which we discuss later).

The associated XML attribute **Exclusive** indicates that a scope-exclusive pseudonym must be created, with the scope string given by the XML attribute **Scope**. This ensures that each user can create only a single pseudonym satisfying this policy, so that the registration service can prevent the same user from obtaining multiple library cards. Setting **Exclusive** to *"false"* would allow an ordinary pseudonym to be presented. The **Pseudonym** element has an optional boolean XML attribute **Established**, not illustrated in the example, which, when set to *"true"*, requires the user to re-authenticate under a previously established pseudonym. The presentation policy can request multiple pseudonyms, e.g., to verify that different pseudonyms actually belong to the same user.

Credentials and Selective Disclosure. For each credential that the user is requested to present, the policy contains a **Credential** element (Figure 4, Lines 6–14), which describes the credential to present in detail. In particular, disjunctive lists of the accepted credential specifications and issuer parameters can be specified via **CredentialSpecAlternatives** and **IssuerAlternatives** elements, respectively (Figure 4, Lines 7-9 and 10–12). The credential element also indicates all attributes that must be disclosed by the user via **DisclosedAttribute** elements (Figure 4, Line 13). The XML attribute **Alias** assigns the credential an alias so that it can be referred to from other places in the policy, e.g., from the attribute predicates.

Key Binding. If present, the **SameKeyBindingAs** attribute of a **Credential** or **Pseudonym** element (Figure 4, Line 6), contains an alias referring either to another Pseudonym element within this policy, or to a Credential element for a credential with key binding. This indicates that the current pseudonym or credential and the referred pseudonym or credential have to be bound to the same key. In our preceding example, the policy requests that the identity card and the presented pseudonym must belong to the same secret key.

Issuance Policy. To support the advanced features described in Section 4, we propose a dedicated *issuance policy*. A library card contains the applicant's name and is bound to the same secret key as the identity card. So the identity card must not only be presented, but also used as a source to carry over the name and the secret key to the library card. The library shouldn't learn either of these during the issuance process. Altogether, to issue library cards the state library creates the issuance policy depicted in Figure 5. It contains the presentation policy from Figure 4 and the credential template that is described in detail below.

```
 1 <IssuancePolicy>
 2   <PresentationPolicy PolicyUID="libcard"> ... </PresentationPolicy>
 3   <CredentialTemplate SameKeyBindingAs="id">
 4     <CredentialSpecUID> urn:utopia:lib </CredentialSpecUID>
 5     <IssuerParametersUID> urn:utopia:lib:issuer </IssuerParametersUID>
 6     <UnknownAttributes>
 7       <CarriedOverAttribute TargetAttributeType= "urn:utopia:lib:name">
 8         <SourceCredentialInfo Alias="id" AttributeType="urn:creds:id:name"/>
 9       </CarriedOverAttribute>
10     </UnknownAttributes>
11   </CredentialTemplate>
12 </IssuancePolicy>
```

Figure 5: Issuance policy for a library card. The presentation policy on Line 2 is depicted in Figure 4.

Credential Template. A credential template describes the relation of the new credential to the existing credentials that were requested in the presentation policy. The credential template (Figure 5, Lines 3–11) must first state the unique identifier of the credential specification and issuer parameters of the newly issued credential (notice that here those are different than the identifiers of the credential specification and issuer parameters of the credential that is presented). The optional XML attribute **SameKeyBindingAs** further specifies that the new credential will be bound to the same secret key as a credential or pseudonym in the presentation policy, in this case the identity card.

Within the **UnknownAttributes** element (Figure 5, Lines 6–10) it is specified which user attributes of the new credential will be carried over from existing credentials in the presentation token. The **SourceCredentialInfo** element (Figure 5, Line 8) indicates the credential and the user attribute of which the value will be carried over.

Although this is not illustrated in our example, an attribute value can also be specified to be chosen jointly at random by the issuer and the user. This is achieved by setting the optional XML attribute **JointlyRandom** to *"true"*.

5.4. Presentation and Issuance Token.

A *presentation token* consists of the *presentation token description*, containing the mechanism-agnostic description of the revealed information, and the *cryptographic evidence*, containing opaque values from the specific cryptography that "implements" the token description. The presentation token description roughly uses the same syntax as a presentation policy. An *issuance token* is a special presentation token that satisfies the stated presentation policy, but that contains additional cryptographic information required by the credential template.

The main difference to the presentation and issuance policy is that in the returned token a **Pseudonym** (if requested in the policy) now also contains a **PseudonymValue** (Figure 6, Line 6). Similarly, the **DisclosedAttribute** elements (Figure 6, Lines 10–12) in a token now also contain the actual user attribute values. Finally, all data from the cryptographic implementation of the presentation token and the advanced issuance features are grouped together in the **CryptoEvidence** element (Figure 6, Line 17). This data includes, e.g., proof that the contained identity card is not revoked by the issuer and that it is bound bound to the same secret key as the pseudonym.

```
<IssuanceToken>
  <IssuanceTokenDescription>
    <PresentationTokenDescription PolicyUID ="libcard" >
      <Message> ... </Message>
      <Pseudonym Alias="nym" Scope="urn:library:issuance" Exclusive="true" />
        <PseudonymValue> MER2VXISHI=</PseudonymValue>
      </Pseudonym>
      <Credential Alias="id" SameKeyBindingAs="nym" >
          ...
        <DisclosedAttribute AttributeType="urn:creds:id:state" >
          <AttributeValue> Nirvana </AttributeValue>
        </DisclosedAttribute>
      </Credential>
    </PresentationTokenDescription>
    <CredentialTemplate SameKeyBindingAs="id" > ... </CredentialTemplate>
  </IssuanceTokenDescription>
  <CryptoEvidence> ... </CryptoEvidence>
</IssuanceToken>
```

Figure 6: Issuance token for obtaining the library card.

5.5. Presentation Policy with Extended Features

Recall that the state library has a privacy-friendly online interface for borrowing books, but that it wants to identify readers who don't properly return their books and potentially ban them for borrowing more paper books. Also recall that the library has a special program for young readers. Altogether, for borrowing books under the "young-reader"-conditions, users have to satisfy the presentation policy depicted in Figure 7.

A presentation policy that is used for plain presentation (i.e., not within an issuance policy) can consist of multiple policy alternatives, each wrapped in a separate **PresentationPolicy** element (Figure 7, Lines 2–34 and 35–63). The returned presentation token must satisfy (at least) one of the specified policies.

The example presentation policy requires two **Credential** elements, for the library and for the identity card, which must belong to the same secret key as indicated by the XML attribute **SameKeyBindingAs**.

Attribute Predicates. No user attributes of the identity card have to be revealed, but the **AttributePredicate** element (Figure 7, Lines 30–33) specifies that the date of birth must be after April 1st, 1988, i.e., that the reader is younger than twenty-six. Supported predicate functions include equality, inequality, greater-than and less-than tests for most basic data types, as well as membership of a list of values. The arguments of the predicate function may be credential attributes (referred to by the credential alias and the attribute type) or constant values. See [16, 17] for an exhaustive list of supported predicates and data types and note that an attribute's encoding as defined in the credential specification has implications on which predicates can be used for it and whether it is inspectable [16, Sec. 4.2.1].

Inspection. To be able to nevertheless reveal the name of an anonymous borrower and to impose a fine when a book is returned late or damaged, the library can make use of inspection. The **DisclosedAttribute** element for the user attribute *"...:name"* contains

```
1  <PresentationPolicyAlternatives>
2    <PresentationPolicy PolicyUID= "young−reader" >
3      <Message> ... </Message>
4      <Credential Alias="libcard" SameKeyBindingAs="id" >
5        <CredentialSpecAlternatives>
6          <CredentialSpecUID> urn:utopia:lib </CredentialSpecUID>
7        </CredentialSpecAlternatives>
8        <IssuerAlternatives>
9          <IssuerParametersUID> urn:utopia:lib:issuer </IssuerParametersUID>
10       </IssuerAlternatives>
11       <DisclosedAttribute AttributeType= "urn:utopia:lib:name" >
12         <InspectorAlternatives>
13           <InspectorParametersUID> urn:lib:arbitrator </InspectorParametersUID>
14         </InspectorAlternatives>
15         <InspectionGrounds> Late return or damage. </InspectionGrounds>
16       </DisclosedAttribute>
17     </Credential>
18     <Credential Alias="id" >
19       <CredentialSpecAlternatives>
20         <CredentialSpecUID> urn:creds:id </CredentialSpecUID>
21       </CredentialSpecAlternatives>
22       <IssuerAlternatives>
23         <IssuerParametersUID> urn:utopia:id:issuer </IssuerParametersUID>
24       </IssuerAlternatives>
25     </Credential>
26     <VerifierDrivenRevocation>
27       <RevocationParametersUID> urn:lib:blacklist </RevocationParametersUID>
28       <Attribute CredentialAlias ="libcard" AttributeType="urn:utopia:lib:name" />
29     </VerifierDrivenRevocation>
30     <AttributePredicate Function= "...:date−greater−than" >
31       <Attribute CredentialAlias ="id" AttributeType= "urn:creds:id:bdate" />
32       <ConstantValue> 1988−04−01 </ConstantValue>
33     </AttributePredicate>
34   </PresentationPolicy>
35   <PresentationPolicy PolicyUID= "regular−reader" >
```

Lines 36–62 are identical to lines 3–29 (i.e., without the AttributePredicate element).

```
63   </PresentationPolicy>
64 </PresentationPolicyAlternatives>
```

Figure 7: Presentation policy for borrowing books.

InspectorParametersUID and **InspectionGrounds** child elements, indicating that the attribute value must not be disclosed to the verifier, but to the specified inspector with the specified inspection grounds. The former child element specifies the inspector's public key under which the value must be encrypted, in this case belonging to a designated arbiter within the library. The latter element specifies the circumstances under which the attribute value may be revealed by the arbiter. Our language also provides a data artifact for inspector parameters, which we omit here for space reasons.

Issuer-Driven Revocation. When the presentation policy requests a credential that is subject to issuer-driven revocation (as defined in the credential specification), the credential must be proved to be valid with respect to the most recent revocation information. However, a policy can also require the use of a particular past version of the revocation information. In the latter case, the element **IssuerParametersUID** has an extra XML attribute **RevocationInformationUID** specifying the identifier of the specific revocation information. The specification of the referenced **RevocationInformation** is given in [16]. Presentation tokens can accordingly state the validity of credentials with respect to a particular version by using a **RevocationInformationUID** XML element in the corresponding Credential element.

Verifier-Driven Revocation. If customers return borrowed books late or damaged, they are excluded from borrowing further paper books, but they are still allowed to use the library's online services. In our example, this is handled by a **VerifierDrivenRevocation** element (Figure 7, Lines 26–29), which specifies that the user attribute *"...:name"* of the library card must be checked against the most recent revocation information from the revocation authority *"urn:lib:blacklist"*. Revocation can also be based on a combination of user attributes from different credentials, in which case there will be multiple **Attribute** child elements per **VerifierDrivenRevocation**. The presentation policy can also contain multiple **VerifierDrivenRevocation** elements for one or several credentials, the returned presentation token must then prove its non-revoked status for *all* of them.

5.6. Interaction with the User Interface

During a presentation, the user can potentially satisfy the presentation policy alternatives in many ways. In order to allow the user to choose which presentation policy he wishes to satisfy, to choose how to satisfy the chosen policy (e.g., if he has multiple credentials of one type), and to check what he reveals by doing so, the Privacy-ABC framework generates a **UiPresentationArguments** object and hands it over to the application, which in turn will probably want to forward it to some sort of user interface. The framework then expects an object of type **UiPresentationReturn** with the user's choice. There are similar objects **UiIssuanceArguments** and **UiIssuanceReturn** for issuance. Standardizing the format of these objects is less critical than the other described in the remained of this section as they remain confined to the user's machine; we show here one possible embodiment of these objects.

We designed the **UiPresentationArguments** object (Figure 8) such that the complexity of the user interface is minimized: (1) it contains enough information so that the application does not have to query additional data from the Privacy-ABC framework, and (2) it contains some redundant information so that it does not need to do complex

```
<UiPresentationArguments>
  <data>
    <credentialSpecification id="urn:utopia:lib">...</credentialSpecification>
    <credentialSpecification id="urn:creds:id">...</credentialSpecification>
    <issuer id="urn:utopia:lib:issuer">...</issuer>
    <issuer id="urn:utopia:id:issuer">...</issuer>
    <inspector id="urn:lib:arbitrator">...</inspector>
    <revocationAuthority id="urn:utopia:id:ra">...</revocationAuthority>
    <credentialDescription id="urn:utopia:lib:74bddfb3−6886−43ac−83f8−ca3b72ad050d">...</credentialDescription>
    <credentialDescription id="urn:creds:id:14f22b9d−06e0−4110−a8d9−b1a922462cd1">...</credentialDescription>
  </data>
  <tokenCandidatePerPolicy policyId="0">
    <policy>...</policy>
    <tokenCandidate candidateId="0">
      <tokenDescription>...</tokenDescription>
      <credential ref="urn:utopia:lib:74bddfb3−6886−43ac−83f8−ca3b72ad050d" />
      <credential ref="urn:creds:id:14f22b9d−06e0−4110−a8d9−b1a922462cd1" />
      <revealedFact>
        <description lang="EN">You prove that urn:creds:id:bdate from credential urn:creds:id
            is greater than 1988−04−01 (26 years ago).</description>
      </revealedFact>
      <revealedFact>
        <description lang="EN">You prove that ‘Full Name’ from credential ‘Library Card’
            is not revoked by the verifier urn:lib:blacklist.</description>
      </revealedFact>
      <revealedFact>
        <description lang="EN">You prove that urn:creds:id is not revoked by urn:utopia:id:ra.</description>
      </revealedFact>
      <inspectableAttribute>
        <credential ref="urn:utopia:lib:74bddfb3−6886−43ac−83f8−ca3b72ad050d" />
        <attributeType>urn:utopia:lib:name</attributeType>
        <inspectionGrounds>Late return or damage.</inspectionGrounds>
        <inspectorAlternative ref="urn:lib:arbitrator" />
      </inspectableAttribute>
    </tokenCandidate>
  </tokenCandidatePerPolicy>
  <tokenCandidatePerPolicy policyId="1">...</tokenCandidatePerPolicy>
</UiPresentationArguments>
```

Figure 8: Message sent to the User Interface for Presentation.

```
<UiPresentationReturn>
  <chosenPolicy>0</chosenPolicy>
  <chosenPresentationToken>0</chosenPresentationToken>
  <chosenInspectors>urn:lib:arbitrator</chosenInspectors>
</UiPresentationReturn>
```

Figure 9: Response from the User Interface for Presentation.

```
1  <UiIssuanceArguments>
2    <data>
3      ...
4      <pseudonym id="nym:urn:library:issuance:965999d1−25e9−49e5−8db6−ad8ae9705807">...</pseudonym>
5      ...
6    </data>
7    <tokenCandidate candidateId="0">
8      ...
9      <pseudonymCandidate candidateId="0">
10       <pseudonym ref="nym:urn:library:issuance:965999d1−25e9−49e5−8db6−ad8ae9705807" />
11     </pseudonymCandidate>
12     ...
13     </tokenCandidate>
14   <issuancePolicy>...</issuancePolicy>
15 </UiIssuanceArguments>
```

Figure 10: Message sent to the User Interface for Issuance.

```
<UiIssuanceReturn>
  <chosenIssuanceToken>0</chosenIssuanceToken>
  <chosenPseudonymList>0</chosenPseudonymList>
  <metadataToChange>
    <entry>
        <key>nym:urn:library:issuance:965999d1−25e9−49e5−8db6−ad8ae9705807</key>
      <value>I used this to obtain my library card.</value>
    </entry>
  </metadataToChange>
</UiIssuanceReturn>
```

Figure 11: Response from the User Interface for Issuance.

parsing of the policy to figure out what exactly is being revealed. It consists of two parts: the first part is a **data** element, which lists all parameters and similar objects that are referred to in the second part: a list of all credential specifications (Lines 3–4), summaries of all issuer parameters (Lines 5–6), summaries of all inspector parameters (Line 7), summaries of all revocation authorities (Line 8), credential descriptions (Lines 9–10), and pseudonym descriptions (not shown for this example, but see Line 4 of Figure 10). The second part consists of a list of **tokenCandidatePerPolicy** elements, which in turn comprise a presentation policy (Line 13) and a list of **tokenCandidate** showing all possible alternatives to satisfy the policy. The latter consists of a partially filled out presentation token description (Line 15); the list of credentials that will be presented (Lines 16–17); all possible alternative lists of pseudonyms that are compatible with the presented credentials and that satisfy the policy (not shown in this example, but see Lines 9–11 in Figure 10), here the Privacy-ABC framework will tentatively create new pseudonyms each time and include those in the list, these pseudonyms are then only saved if the user actually selects them for inclusion in the presentation token; a list of facts that will be revealed as part of the presentation (Lines 18–28), such as equality between attributes, predicates over the attributes, revocation checks—the friendly names of credentials, attributes, and parameters are used whenever available; the list of attributes that are revealed (not shown in this example), including attributes that are proven to be equal to a revealed attribute; and the list of inspectable attributes (Lines 29–34) with a choice of possible inspectors (Line 33).

The **UiPresentationReturn** object (Figure 9) indicates which policy (Line 2), which presentation token within that policy (Line 3), and which inspector for each of the inspectable attributes (Line 4) the user chose. Not shown in this example, but also part of the **UiPresentationReturn** is the list of pseudonyms the user wishes to chose, and whether the user wishes to change the metadata of any of the stored pseudonyms (we show examples of those in Figure 11).

The **UiIssuanceArguments** object (Figure 10) is similar to the **UiPresentationArguments** element. Since there is only one issuance policy per issuance transaction, we removed the **tokenCandidatePerPolicy** element; instead the **tokenCandidate** elements (Line 7) and **issuancePolicy** element (Line 14) are direct children of the root element.

The **UiIssuanceReturn** object (Figure 11) is similar to the **UiPresentationReturn** object. It indicates which presentation token within the policy (Line 2), which inspectors (not shown in this example), and which list of pseudonyms (Line 3) were chosen. In this example, the user has also chosen to associate new metadata to the pseudonym (Lines 4–9).

6. API of a Privacy-ABC System

In this section, we present the architecture of a Privacy-ABC system and the application programming interface (API) of the Privacy-ABC framework in that system. The latter contains the methods that are minimally necessary to implement all the features provided by the language framework described in the previous section. We illustrate the API with sample code that realize the library scenario from Section 3. In this document we consider a discussion of an implementation out of scope.

The API is based on the reference implementation of a Privacy-ABC framework that we created within the EU project ABC4Trust [16, 17, 1]; the source code of that implementation is available at https://github.com/p2abcengine/p2abcengine. However, with respect to the API of the reference implementation, we omitted some convenience methods, simplified the error reporting mechanism, and adjusted the signature of some methods for didactic reasons. The API exposes technology-agnostic methods for generating cryptographic parameters and keys, importing these parameters, generating and verifying presentation tokens, issuing credentials, inspecting tokens, and revoking credentials or attributes. We provide a summary of the API in Table 2.

6.1. Architecture of a Privacy-ABC system

For an easy integration of Privacy-ABCs in various applications, we consider a Privacy-ABC framework consisting of a mechanism-independent Privacy-ABC engine layer on top of the core cryptographic engine layer. This Privacy-ABC engine contains all the mechanism-agnostic components of a Privacy-ABC system and processes the data formats that we presented in Section 5. It is invoked by the application layer and calls out to the cryptographic engine to obtain the mechanism-specific cryptographic data. The Privacy-ABC framework also contains trusted storage containing public authenticated data such as issuer parameters, credential specifications, inspector parameters, and revocation authority parameters, as well as secure storage containing secret keys and users' credentials.

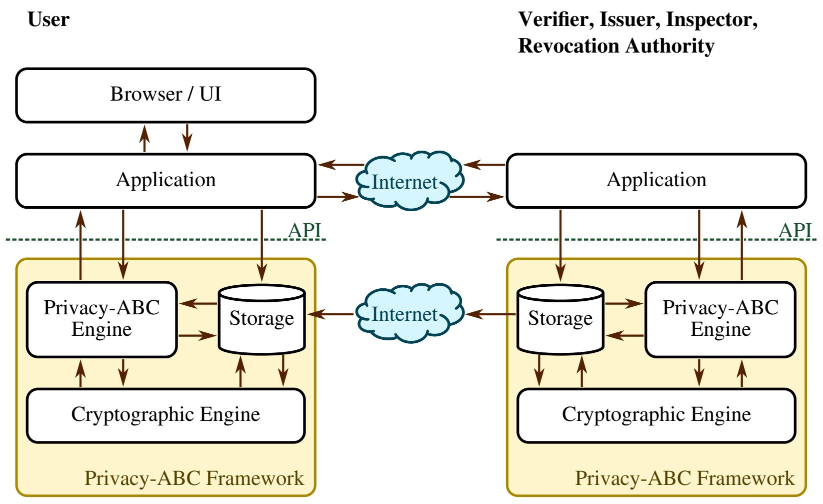

Figure 12: Architecture of a Privacy-ABC system.

Function name	**Input**	**Output**	**Who?***
Parameter generation			
generateSystemParameters	int *securityLevel*	SystemParameters	Is
generateIssuerParameters	URI *id*, SystemParameters, URI *technology*, int *maximalNumberOfAttributes*, *@Nullable* URI *rev.Auth.Id*	IssuerParameters	Is
generateInspectorParameters	URI *id*, SystemParameters, URI *technology*	InspectorParameters	In
generateRev.Auth.Parameters	URI *id*, SystemParameters, URI *technology*, URI *revocationInfoLocation*, URI *nonRevocationEvidenceLocation*, URI *nonRevocationUpdateLocation*	RevocationAuthorityParameters	RA
generateUserSecretKey	SystemParameters	URI *id*	U
Storage of parameters			
storeCredentialSpecification	CredentialSpecification	boolean *success*	U, V, Is
storeSystemParameters	SystemParameters	boolean *success*	U, V, Is, In, RA
storeIssuerParameters	IssuerParameters	boolean *success*	U, V, Is
storeInspectorParameters	InspectorParameters	boolean *success*	U, V, Is, In
storeRev.Auth.Parameters	RevocationAuthorityParameters	boolean *success*	U, V, Is, RA
Issuance			
initIssuanceProtocol	IssuancePolicy, List<Attribute> *issuerSpecifiedAttributes*	IssuanceMessage, boolean *isLastMessage*	Is
issuanceProtocolStep	IssuanceMessage	IssuanceMessage, boolean *isLastMessage*	Is
extractIssuanceToken	IssuanceMessage	IssuanceToken	Is
issuanceProtocolStep	IssuanceMessage	IssuanceMessage, CredentialDescription, *@Nullable* UiIssuanceArguments	U
issuanceProtocolStep	UiIssuanceReturn	IssuanceMessage	U
Presentation			
createIdentitySelectorArguments	PresentationPolicyAlternatives	*@Nullable* UiPresentationArguments	U
createPresentationToken	UiPresentationReturn	PresentationToken	U
verifyTokenAgainstPolicy	PresentationToken, PresentationPolicyAlternatives	boolean	V
Inspection and revocation			
inspect	PresentationToken, URI *credentialAlias*, URI *attributeType*	Attribute	In
inspect	IssuanceToken, URI *credentialAlias*,URI *attributeType*	Attribute	In
revoke	URI *rev.Auth.Id*, List<Attribute> *toRevoke*	—	RA

Table 2: Summary of the application programming interface (API) of our Privacy-ABC system. * U = user, V = verifier, Is = issuer, In = inspector, RA = revocation authority.

Programmers that wish to use our Privacy-ABC framework need to code the application layer sitting on top of that framework. This application layer communicates with the framework via the API described in this section. This application should also be able to display some information to the end users via a dedicated user interface (which can for example be a page displayed in a web browser): for example during *identity selection*, the user must be asked to chose how to satisfy a given policy given all possible alternatives (or to abort the protocol) and see what information he reveals in each case. See Figure 12.

6.2. Setup

To equip all parties in a Privacy-ABC system with the necessary key material, the API provides methods for generating public and/or private cryptographic parameters. Additionally, there is a method that generates system parameters. The Privacy-ABC framework stores the private parameters in the trusted storage of the corresponding party. The generated public parameters must then be authentically sent to the other parties in the system.

Each party in the system is responsible for keeping its key storage up-to-date. Once it obtains a public parameter from a party and after it verified the authenticity of that parameter, it must import the parameter in its key storage with the dedicated API methods. The framework may refuse to store parameters or credential specifications that have the same UID as another object in the store, or it may decide to overwrite the old object. The Privacy-ABC framework can retrieve the different parameters and credential specifications by their UID. The corresponding operations to retrieve the keys and parameters from the key storage are used internally only, and are therefore not exposed by the API described here.

See Figure 13 for the application layer setup code for our library scenario of Section 3.

System parameters. When generating system parameters, a security level must be provided. Reasonable values for the security level are values between 80 and 256. System parameters define security parameters (e.g., size of secrets, size of moduli, size of group orders, prime probability), the range of values the attributes can take, and the cryptographic parameters for the pseudonyms. The choice of parameters will be such that the overall system has an equivalent security level as a symmetric cryptosystem with securityLevel bits [56].

To ensure interoperability, every user, issuer, inspector, and revocation authority in the system must use the same system parameters for generating their cryptographic keys and parameters. To achieve this, for example, a trusted authority such as a standardization body (or in our scenario, the Utopian government) could generate and publish system parameters for various security levels, which are then used by all parties.

Issuer parameters. When generating issuer parameters, one must specify the maximal number of attributes that can appear in credential specifications that are used in conjunction with these issuer parameters. That number is needed because it might influence the length of the issuer parameters.

Utopia government (issuer and revocation authority):

```
int securityLevel = 128; // Security equivalent to 128−bit symmetric cipher.
SystemParameters sp = utopia.generateSystemParameters(securityLevel);

int maxNumAttributesUt = 4; // Identity card has 4 named attributes (plus the user's secret key).
IssuerParameters utIp = utopia.generateIssuerParameters("urn:utopia:id:issuer", sp,
    "urn:com:microsoft:uprove", maxNumAttributesUt, "urn:utopia:id:ra");

URI raLocationUt = "https://api.ra.id.utopia/";
RevocationAuthorityParameters utRap = utopia.generateRevocationAuthorityParameters("urn:utopia:id:ra",
    sp, "urn:privacy−abe:accumulators:cl", raLocationUt + "rinfo", raLocationUt + "nre", raLocationUt + "nru");

CredentialSpecification utCs = "..."; // Specification of identity card. See Figure 2.
utopia.storeCredentialSpecification(utCs);

// Distribute sp, utRap, utIp, utCs to all other entities
```

Arbitrator for Library (inspector):

```
InspectorParameters arbInsp = arbitrator.generateInspectorParameters("urn:lib:arbitrator", sp,
    "urn:inspection:CS03");

arbitrator.storeSystemParameters(sp);

// Distribute arbInsp to all other entities
```

Library (issuer, revocation authority, and verifier):

```
int maxNumAttributesLib = 1; // Library card has 1 named attribute (plus the user's secret key).
IssuerParameters libIp = library.generateIssuerParameters("urn:utopia:lib:issuer", sp,
    "urn:com:ibm:idemix", maxNumAttributesLib, null);

URI raLocationLib = "https://api.blacklist.lib.utopia/";
RevocationAuthorityParameters libRap = library.generateRevocationAuthorityParameters("urn:lib:blacklist",
    sp, "urn:privacy−abe:NFHF09", raLocationLib + "rinfo", raLocationLib + "nre", raLocationLib + "nru");

CredentialSpecification libCs = "..."; // Specificaion of library card.
library.storeCredentialSpecification(libCs);

library.storeSystemParameters(sp);
library.storeIssuerParameters(utIp);
library.storeRevocationAuthorityParameters(utRap);
library.storeCredentialSpecification(utCs);
library.storeInspectorParameters(arbInsp);

// Distribute libIp, libRap to all other entities
```

User:

```
user.generateUserSecretKey(sp);

user.storeSystemParameters(sp);
user.storeIssuerParameters(utIp);
user.storeIssuerParameters(libIp);
user.storeRevocationAuthorityParameters(utRap);
user.storeRevocationAuthorityParameters(libRap);
user.storeCredentialSpecification(utCs);
user.storeCredentialSpecification(libCs);
user.storeInspectorParameters(arbInsp);
```

Figure 13: Application layer code for the setup phase of our library scenario.

Revocation authority parameters. When generating revocation authority parameters, one must specify the location where the Privacy-ABC framework of the users and verifiers can obtain the latest revocation information, the location where the framework of issuers can obtain the initial non-revocation evidence of newly issued credentials, and the location where the framework of users can obtain updates to their non-revocation evidence.

User secret keys. Our reference implementation also supports the storage of private keys on external devices such as smartcards. However, for the sake of brevity, we do not describe the corresponding API methods here. We note that a user may generate multiple keys by calling this method multiple times.

6.3. Issuance

Generally speaking, issuance is an interactive multi-round protocol between a user and an issuer, at the end of which the user obtains a credential. In fact, issuance can be seen as a special case of a standard resource request, where the resource is a new credential that the user wants to obtain. Thus, to handle such a credential request, the Privacy-ABC framework might invoke the same components and procedures as in the presentation scenario described later. However, depending on the scenario, the issuance transaction involves additional components to handle the case where the user wishes to (blindly) carry over her attributes or her secret key from one of her existing credentials to the new credential.

To start an issuance transaction, the user first authenticates towards the issuer. The exact details of such authentication are outside the scope of this document (for example, it can be done using traditional means such as username and password). The user also indicates the credential type she wishes to obtain. The issuer triggers the issuance of a credential through the API only after having received that initial message from the user. As described in Section 4.4, there are two variants of issuance: *simple issuance* and *advanced* issuance, where the latter applies if attributes or a key need to be carried over from existing credentials.

6.3.1. Simple issuance

In the simple issuance variant, an issuer issues the user a credential that is unrelated to any existing credentials or pseudonyms already owned by the user. In such a setting, the issuer first sends to the Privacy-ABC framework the set of attributes that shall be certified in the new credential, and with an issuance policy that merely contains the identifiers of the credential specification and the issuer parameters of the credential that is to be issued (see Line 9 in Figure 14), thus initiating the cryptographic issuance protocol. The framework outputs an issuance message containing cryptographic data (the format of the data is specific to the technology of the credential to be issued) and a reference that uniquely identifies the instance of the corresponding issuance protocol. The returned issuance message is then sent by the issuer to the user.

Upon receiving an issuance message, both the user and subsequently the issuer pass the message to their Privacy-ABC framework (Line 24 and Line 18). If the framework outputs an issuance message (Line 25–27), that message is sent to the other party until

Issuer (Library):

```
IssuancePolicy issuancePolicy = "..."; // Policy containing only a credential template.
CredentialSpecification utCsNoKeyBinding = "..."; // Specification for non−key−bound identity card.
                                                  // See Figure 2, but with KeyBinding = false.
List<Attribute> issuerSetAttributes = new ArrayList<Attribute>();
isserSetAttributes.add(utCsNoKeyBinding.getAttribute("∗name").setValue("Arthur Dent"));
isserSetAttributes.add(utCsNoKeyBinding.getAttribute("∗state").setValue("Nirvana"));
isserSetAttributes.add(utCsNoKeyBinding.getAttribute("∗bdate").setValue("1992−10−01"));
isserSetAttributes.add(utCsNoKeyBinding.getAttribute("∗revocationhandle").setValue("47"));
Pair<IssuanceMessage, Boolean> ret = library.initIssuanceProtocol(issuancePolicy, issuerSetAttributes);
while(true) {
  IssuanceMessage imToUser = ret.first /∗ issuanceMessage ∗/;
  // Send imToUser to the user over a secure channel.
  if (ret.second /∗ lastMessage ∗/ == true) {
    break;
  }
  IssuanceMessage imFromUser;
  // Receive the issuance message imFromUser from the user over a secure channel.
  ret = library.issuanceProtocolStep(imFromUser);
}
```

User:

```
while(true) {
  IssuanceMessage imFromIssuer, imToIssuer;
  // Receive the issuanceMessage imFromIssuer from the issuer over a secure channel.
  Triple<IssuanceMessage, CredentialDescription, UiIssuanceArguments> ret;
  ret = user.issuanceProtocolStep(imFromIssuer);
  imToIssuer = ret.first /∗ issuanceMessage ∗/;
  if (imToIssuer != null) {
    // Send imToIssuer to the issuer over a secure channel.
  } else {
    CredentialDescription descOfIssuedCredential = ret.second /∗ credentialDescription ∗/;
    // The description of the issued credential is stored in descOfIssuedCredential:
    // optionally display a success message to the user.
    break;
  }
}
```

Figure 14: Application layer code for the simple issuance protocol. Here we show how to issue an identity card that is *not* bound to the user's secret key. To properly realize the library scenario, advanced issuance is required instead of simple issuance.

Issuer (Library):

```
IssuancePolicy issuancePolicy = "..."; // Policy for issuing a library card based on an identity card. See Figure 5.
List<Attribute> issuerSetAttributes = Collections.emptyList();
Pair<IssuanceMessage, Boolean> ret = library.initIssuanceProtocol(issuancePolicy, issuerSetAttributes);
while(true) {
  IssuanceMessage imToUser = ret.first /* issuanceMessage */;
  // Send imToUser to the user over a secure channel.
  if (ret.second /* lastMessage */ == true) {
    break;
  }
  IssuanceMessage imFromUser;
  // Receive the issuance message imFromUser from the user over a secure channel.
  IssuanceToken it = library.extractIssuanceToken(imFromUser);
  if(it != null) {
    // Optionally, perform additional checks on the user's issuance token:
    // check that the pseudonym has never been seen before (and abort if a card was already issued to that pseudonym).
  }
  ret = library.issuanceProtocolStep(imFromUser);
}
```

User:

```
while(true) {
  IssuanceMessage imFromIssuer, imToIssuer;
  // Receive the issuanceMessage imFromIssuer from the issuer over a secure channel.
  Triple<IssuanceMessage, CredentialDescription, UiIssuanceArguments> ret;
  ret = user.issuanceProtocolStep(imFromIssuer);
  if (ret.third /* uiIssuanceArguments */ != null) {
    UiIssuanceReturn uir;
    // Ask the user to choose which credentials to use by invoking the user interface with
    // ret.third /* uiIssuanceArguments */. The user's choice is recorded in uir.
    imToIssuer = user.issuanceProtocolStep(uir);
  } else {
    imToIssuer = ret.first /* issuanceMessage */;
  }
  if (imToIssuer != null) {
    // Send imToIssuer to the issuer over a secure channel.
  } else {
    CredentialDescription descOfIssuedCredential = ret.second /* credentialDescription */;
    // The description of the issued credential is stored in descOfIssuedCredential:
    // optionally display a success message to the user.
    break;
  }
}
```

Figure 15: Application layer code for the advanced issuance protocol of an Utopian library card. Similar code can be used for the issuance of the Utopian identity card.

the user's framework finally outputs a credential (Line 28–29). At the end of a successful issuance protocol, the user's Privacy-ABC framework stores the new credential in her local credential store and returns the description of the credential to the user (Line 29–32).

In our library scenario, there is no simple issuance: both the identity card and the library card need to carry over the secret key of the user (and the library card also needs to carry over the full name of the user). In Figure 14 we show an application layer code for simple issuance for a simple identity card that is *not* bound to the user's secret key—the credential specification for that simple identity card is almost the same as for the regular identity card (see Figure 2), except that **KeyBinding** is set to false.

6.3.2. Advanced issuance

In the advanced issuance variant, the information embedded in the newly issued credential can be blindly carried over from existing credentials and pseudonyms that are already owned by the user. To this end, the issuance protocol is preceded by the generation and verification of an issuance token, which is generated on the basis of an issuance policy sent to the user. More precisely, the issuer triggers an advanced issuance transaction by invoking the Privacy-ABC framework with an issuance policy and the set of known user attributes that shall be certified in the new credential (see Line 3 in Figure 15). The issuance policy must require the user to present at least one credential or one pseudonym, otherwise simple issuance is performed. The framework may simply wrap the issuance policy into an issuance message. The returned issuance message must then be sent to the user (see Line 6).

The user in turn invokes his framework with the received message (see Line 23). The user's Privacy-ABC framework recognizes that this is an advanced issuance scenario, and subsequently starts preparing an issuance token. This process is similar to the generation of a presentation token described below in that the framework outputs an object of type **UiIssuanceArguments** for the user to perform an *identity selection*, and then records the user's response in an object of type **UiIssuanceReturn** (see Lines 25–27). Finally, based on the user's choice, her Privacy-ABC framework generates an issuance token (see Line 28), which includes additional cryptographic data needed for the subsequent issuance protocol. The issuance token is wrapped in an issuance message, which the user then forwards to the issuer (see Line 33).

If the issuer wants to perform additional checks on the contents of the issuance token description before starting the actual issuance protocol—for example because the issuer wants to check that the value of a scope-exclusive pseudonym was not yet registered in a given database—he can ask the framework to unwrap the token from the incoming message (see Line 12). As for simple issuance, the issuer then sends the incoming issuance message from the user to the framework (see Line 17). The issuer's Privacy-ABC framework then verifies the issuance token contained in the message with respect to the issuance policy (using similar methods as for the verification of a presentation token described below). If the verification succeeds, the cryptographic issuance protocol is started. The method outputs an issuance message containing cryptographic data depending on the technology of the credential to issue. The issuer then sends the returned issuance message to the user (see Line 6).

Whenever the user or the issuer receive an issuance message, they send it to their framework (see Line 23 and Line 17). The framework then outputs either another issuance message that must be sent to the other party, or an indication of the completion of the protocol. At the end of the protocol, the user's Privacy-ABC framework stores the obtained credential and returns a description of that credential to the user (see Lines 34–38).

In Figure 15 we show the application layer code for the advanced issuance protocol of the Utopian library card. Very similar code can be used for issuance of identity cards. Finally, we note that this code also works for simple issuance, the only differences being that the user will not be asked to do identity selection, the issuer won't see any issuance token, and the number of exchanged messages will be lower.

6.4. Presentation

Verifier (Library):

```
1  PresentationPolicyAlternatives ppa = "..."; // Policy for borrowing books. See Figure 7.
2  // Send the presentation policy alternatives ppa to the user.
```

User:

```
3  // Receive the presentation policy alternatives ppa from the verifier.
4  UiPresentationArguments upa = user.createIdentitySelectorArguments(ppa);
5  UiPresentationReturn upr;
6  // Ask user to choose which credentials to use by invoking the user
7  // interface with upa. The user's choice is recorded in upr.
8  PresentationToken pt = user.createPresentationToken(upr);
9  // Send pt to the verifier over a secure channel.
```

Verifier (Library):

```
10 // Receive the presentation token pt from the user.
11 boolean ok = library.verifyTokenAgainstPolicy(pt, ppa);
12 if(ok == true) {
13   // Show page for borrowing books and store pt for future reference in case the book was not properly returned.
14 }
```

Figure 16: Application layer code for the presentation protocol for accessing the Utopian library catalogue.

The process of presentation is triggered when the application on the user's side contacts a verifier to request access to a resource. Having received the request, the verifier responds with one or more presentation policies, which are aggregated in a **PresentationPolicyAlternatives** object (see Line 1–2 in Figure 16). Recall that a presentation policy defines what information a user has to reveal to the verifier in order to gain access to the requested resource. For example, it describes which credentials from which trusted issuers are required, which attributes from those credentials have to be revealed, or which predicates the attributes have to fulfill.

Upon receiving the policy, the application on the user's side invokes the Privacy-ABC framework with the received presentation policy alternatives (Line 4). The Privacy-ABC framework then determines whether the user has the necessary credentials and pseudonyms (recall that the Privacy-ABC framework stores all of the user's credentials and pseudonyms) to create one or more tokens that satisfies the policy. This method returns an object of type **UiPresentationArguments** which describes all the possible combinations of the user's credentials and pseudonyms that satisfy the policy, and which we described in Section 5.6. The user's application layer then performs an *identity selection* (see Lines 6–7), which allows the user to choose her preferred combination of credentials and pseudonyms (if there are several ways in which the policy can be satisfied) and to obtain the user's consent in revealing her personal data. The user's choice is recorded in an object of type **UiPresentationReturn**. This object is then passed to the Privacy-ABC framework (see Line 8), which generates and returns a presentation token according the user's choice. Afterwards, the presentation token is sent to the verifier.

After the verifier receives the presentation token from the user, he passes it to the framework to verify whether the statements made in the presentation token satisfy the corresponding presentation policy alternatives (Line 11). The token verification is done in two steps. First, it is determined whether the statements made in the presentation token description logically satisfy the required statements in the corresponding presentation policy. Second, the validity of the cryptographic evidence for the given token description is verified. If both checks succeed, the presentation token is stored in a dedicated token store, which allows the verifier to recognize established pseudonyms.

6.5. Inspection

Inspector (Arbitrator):

```
1 PresentationToken pt = ...; // The token that was presented by the delinquent book borrower.
2 // Parse the token and check that the event specified in the inspection grounds has occurred.
3 Attribute a = arbitrator.inspect(pt, "libcard", "urn:utopia:lib:name");
4 // Send attribute a to the library.
```

Revocation authority (Library):

```
5 // Receive attribute a from the inspector.
6 library.revoke("urn:lib:blacklist", Collections.singletonList(a));
7 // The updated revocation information will be picked up by the users' and verifiers' Privacy−ABC framework.
```

Figure 17: Application layer code for inspection and verifier-driven revocation of the name of a delinquent book borrower.

As described in Section 4.5, the anonymity that is usually provided by Privacy-ABCs can be lifted through inspection if the policy allows it. In particular, if a policy mandates attributes to be inspectable, the user prepares his presentation tokens in a special way: the inspectable attributes are not revealed to the verifier, but are verifiably encrypted in the token under the public key of a trusted inspector and inseparabely tied to some inspection grounds.

In case the event specified in the inspection grounds occurs, the inspection requestor (e.g., the verifier) contacts the inspector to request the de-anonymization of a presentation or issuance token. To do that, it sends the token with the (non-cryptographic) evidence that the inspection grounds are fulfilled to the inspector. If the latter determines by means of the evidence that these grounds are indeed fulfilled, he decrypts the relevant attribute with the the Privacy-ABC framework (see Line 3 in Figure 17).

6.6. Revocation

As discussed in Section 4.6, credentials may need to be revoked either globally (issuer-driven revocation) or for a specific context (verifier-driven revocation).

To revoke a credential globally, the revocation authority passes the credential's revocation handle to the Privacy-ABC framework. For verifier-driven revocation, a conjunction of attribute values can be revoked (that is, all credentials that contain the combination of attribute values specified in the list are revoked) by passing them to the framework (see Line 6 in Figure 17). The revocation authority typically knows the attribute values to revoke because they were either revealed in a former presentation token, or were decrypted by an inspector.

Verifiers must obtain the latest revocation information from the revocation authority in order to correctly detect revoked credentials. This update process must be handled transparently by a verifier's Privacy-ABC framework.

Issuers have to contact their revocation authority during issuance in order to obtain a fresh revocation handle. This is also handled internally by the framework.

Similarly, users have to keep the non-revocation evidence of their credentials up-to-date. The Privacy-ABC framework of a user should allow her to configure whether to contact the revocation authority only shortly prior to presenting a credential, or whether to perform proactive updates at regular intervals. The latter approach has the advantage that presentation is faster and that the revocation authority is not involved each single time a user wants to present her credential(s). Depending on the revocation technology, these updates may even fully preserve the anonymity of the user.

7. Security Discussion

For our framework to be useful, the various parties need to be given security guarantees: a verifier wants to be sure that if a presentation token verifies then all the statements made in it are indeed supported by the issuer and have not been revoked. Furthermore, the verifier wants to be sure that inspection will succeed when needed. The issuer wants to have similar guarantees with respect to issuance tokens. The user wants assurance that her privacy is maintained, i.e., that no more information is leaked than what she willingly released in a presentation token.

Formally proving that such guarantees are fulfilled, provided the underlying cryptography is sound, is far beyond the scope of this paper and is the topic of future work. Likewise, we do not go into detail here on the complex trust relations between the different participants in a Privacy-ABC system, or on which particular privacy or security threats can arise when some of these participants collude. Since presentation policies can always ask to reveal much more information than strictly necessary, one could also consider adding yet another authority to the system to approve "reasonable" policies. Finally, in order to deploy a Privacy-ABC system, one also needs a public-key infrastructure (PKI) to certify issuer parameters, revocation authority parameters, and inspectors' public keys.

For the individual cryptographic building blocks, security proofs are given in the cryptographic literature, and Camenisch and Lysyanskaya give a proof for their credential system [24]. Thus, the first step in proving security of our language framework is to provide precise security notions that capture the high-level security properties stated above and to prove that the composition of the cryptographic building blocks as imposed by our languages achieves those. Next, one would have to show that the mapping of our languages to the concrete cryptographic realizations is sound. Only then one could attempt to formally prove that the security guarantees as stated above are fulfilled.

8. Conclusion

We presented a language framework enabling a unified deployment of Privacy-ABC technologies, in particular, of U-Prove and Identity Mixer. Our framework improves upon the state of the art [57, 30] by covering the entire life-cycle of Privacy-ABCs, including issuance, presentation, inspection, and revocation, and by supporting advanced features such as pseudonyms and key binding. The framework offers a set of abstract concepts that make it possible for application developers to set up a Privacy-ABC infrastructure and to author policies without having to deal with the intricacies of the underlying cryptography.

In an upcoming companion paper, we demonstrate the soundness of our languages by providing a formal semantics that specifies the effects of issuing, presenting, verifying, inspecting, and revoking credentials on the user's credential portfolio and on the knowledge states of the involved parties. A complete description of our framework including the language description as well as the formal semantics is available as a technical report [15].

The proposed language framework has been implemented as part of the ABC4Trust project, where it will be rolled out in two pilot projects. Preliminary tests indicate that our language framework adds a noticeable but reasonable overhead to the cryptographic routines, comparable to the overhead incurred by, for example, XML Signature [55] with respect to the underlying signing algorithm.

Our language framework supports a number of different authentication mechanisms including the mentioned privacy-preserving ones but also standard mechanisms such as X.509. However, most of them will not support the full set of features but we are currently working on a protocol framework that allows the combination of different cryptographic mechanisms to address this.

Acknowledgements

The authors thank Bart De Decker, Lan Nguyen, and the anonymous referees of the conference and journal papers for their valuable comments. The research leading to these results was supported by the European Commission under Grant Agreement 257782 ABC4Trust.

References

[1] Attribute-based Credentials for Trust (ABC4Trust) EU project. `https://abc4trust.eu/`

[2] C. A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio. Exploiting cryptography for privacy-enhanced access control. *J. of Comput. Secur.*, 18(1), 2010.

[3] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. *J. Comput. Secur.*, 16(4), 2008.

[4] A. W. Appel and E. W. Felten. Proof-carrying authentication. *ACM CCS 1999*.

[5] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In *SCN 2006*, vol. 4116 of *LNCS*.

[6] K. D. Bowers, L. Bauer, D. Garg, F. Pfenning, and M. K. Reiter. Consumable credentials in linear-logic-based access-control systems. *NDSS 2007*.

[7] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO 2009*, vol. 5677 of *LNCS*.

[8] P. Bichsel, J. Camenisch, and F.-S. Preiss. A comprehensive framework enabling data-minimizing authentication. In *ACM DIM 2011*.

[9] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *J. Comput. Secur.*, 10(3), 2002.

[10] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*.

[11] S. Brands, L. Demuynck, and B. De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. In *ACISP 07*, vol. 4586 of *LNCS*.

[12] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. MIT Press, 2000.

[13] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT '91*, vol. 547 of *LNCS*.

[14] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT 2008*, vol. 5350 of *LNCS*.

[15] J. Camenisch, M. Dubovitskaya, A. Lehmann, G. Neven, C. Paquin, and F.-S. Preiss. A language framework for privacy-preserving attribute-based authentication. Technical Report RZ3818, IBM, 2012.

[16] J. Camenisch, I. Krontiris, A. Lehmann, G. Neven, C. Paquin, K. Rannenberg, and H. Zwingelberg. H2.1 – ABC4Trust Architecture for Developers. ABC4Trust heartbeat H2.1, 2011.

[17] P. Bichsel, J. Camenisch, M. Dubovitskaya, R. R. Enderlein, I. Krontiris, A. Lehmann, G. Neven, J. Nielsen, C. Paquin, F.-S. Preiss, K. Rannenberg, M. Stausholm, and H. Zwingelberg. H2.2 – ABC4Trust Architecture for Developers. ABC4Trust heartbeat H2.2, 2013.

[18] E.F. Brickell, J. Camenisch, L. Chen. Direct anonymous attestation. In *ACM CCS 2004*, ACM press.

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

[20] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing accountability and privacy using e-cash. In *SCN 06*, vol. 4116 of *LNCS*.

[21] J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *PKC 2009*, vol. 5443 of *LNCS*.

[22] J. Camenisch, M. Kohlweiss, and C. Soriente. Solving revocation with efficient update of anonymous credentials. In *SCN 10*, vol. 6280 of *LNCS*.

[23] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized Schnorr proofs. In Antoine Joux, editor, *EUROCRYPT 2009*, vol. 5479 of *LNCS*.

[24] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001*, vol. 2045 of *LNCS*.

[25] J. Camenisch and A. Lysyanskaya. An identity escrow scheme with appointed verifiers. In *CRYPTO 2001*, vol. 2139 of *LNCS*.

[26] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002*, vol. 2442 of *LNCS*.

[27] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN 02*, vol. 2576 of *LNCS*.

[28] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, vol. 3152 of *LNCS*.

[29] J. Camenisch, G. Neven, and M. Rückert. Fully anonymous attribute tokens from lattices. In *SCN 2012*, vol. 7485 of LNCS, Springer Verlag.

[30] J. Camenisch, S. Mödersheim, G. Neven, F.-S. Preiss, and D. Sommer. A card requirements language enabling privacy-preserving access control. In *SACMAT 2010*.

[31] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*, vol. 2729 of *LNCS*.

[32] D. Crockford. The application/json media type for JavaScript Object Notation (JSON). Internet Engineering Taskforce (IETF) RFC 4627, 2006.

[33] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, vol. 2501 of *LNCS*.

[34] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *PKC 2005*, vol. 3386 of *LNCS*.

[35] J. R. Douceur. The Sybil attack. In *IPTPS 2002*, vol. 2429 of *LNCS*.

[36] D. Ferraiolo and R. Kuhn. Role-based access control. *NIST-NCSC 1992*.

[37] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, vol. 263 of *LNCS*.

[38] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. A linear logic of authorization and knowledge. *ESORICS 2006*.

[39] IBM Research Zurich Security Team. Specification of the identity mixer cryptographic library. Technical Report RZ3730, IBM, 2010.

[40] Identity Mixer. `http://idemix.wordpress.com/`.

[41] International Telecommunication Union. Abstract syntax notation one (ASN.1). ITU-T recommendation X.680, 2008.

[42] J. Killian and E. Petrank. Identity Escrow. In *CRYPTO 1998*, vol. 1462 of LNCS, Springer Verlag.

[43] M. Kirkpatrick, G. Ghinita, and E. Bertino. Privacy-preserving enforcement of spatially aware RBAC. In *IEEE Trans. on Dependable and Secure Computing*, 99(PrePrints), 2011.

[44] J. Lapon, M. Kohlweiss, B. De Decker, and V. Naessens. Analysis of Revocation Strategies for Anonymous *Idemix* Credentials. *IFIP CMS 2011*.

[45] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM TISSEC*, 6(1), 2003.

[46] J. Li, N. Li, and W. Winsborough. Automated trust negotiation using cryptographic credentials. *ACM CCS 2005*.

[47] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *SAC 1999*, vol. 1758 of *LNCS*.

[48] T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In *PKC 2009*, vol. 5443 of *LNCS*.

[49] L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA 2005*, vol. 3376 of *LNCS*.

[50] F. Paci, N. Shang, K. Steuer Jr., R. Fernando, E. Bertino. VeryIDX - A privacy preserving digital identity management system for mobile devices. *Mobile Data Management 2009*.

[51] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*, vol. 576 of *LNCS*.

[52] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[53] A. Squicciarini, A. Bhargav-Spantzel, E. Bertino, and A. Czeksis. Auth-SL – A system for the specification and enforcement of quality-based authentication policies. In *ICICS 2007*.

[54] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Comput.*, 29(2), 1996.

[55] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance comparison of security mechanisms for grid services. GRID 2004.

[56] N. Smart. ECRYPT II yearly report on algorithms and keysizes, revision 1.0. `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`, 2012.

[57] Microsoft U-Prove. `http://www.microsoft.com/uprove`.

[58] E. R. Verheul. Self-Blindable Credential Certificates from the Weil Pairing. ASIACRYPT 2001.

[59] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. *ACM FMSE 2004*.

[60] W. Winsborough, K. Seamons, and V. Jones. Automated trust negotiation. *DISCEX 2000*.

[61] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005.