

Chapter 2

An Architecture for Privacy-ABCs

Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Stephan Krenn, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Franz-Stefan Preiss, Kai Rannenberg, and Ahmad Sabouri

Abstract One of the main objectives of the ABC4Trust project was to define a common, unified architecture for Privacy-ABC systems to allow comparing their respective features and combining them into common platforms. The chapter presents an overview of features and concepts of Privacy-ABCs and introduces the architecture proposed by ABC4Trust, describing the layers and components as well as the high-level APIs. We also present the language framework of ABC4Trust through an example scenario. Furthermore, this chapter investigates integration of Privacy-ABCs with the existing Identity Management protocols and also analyses the required trust relationships in the ecosystem of Privacy-ABCs.

As we mentioned in the previous chapter, there are several implementations of Privacy-ABCs, based on different cryptographic primitives. Even though these schemes have similar features, they are realized with different cryptographic mechanisms and many times they are even called differently, making these technologies hard to understand and compare. Their differences and complexity also makes it difficult for application developers to use them in practice and it is almost impossible to switch between them once the application has been deployed.

The ABC4Trust architecture presented in this chapter aims to overcome these problems by defining an abstract interface to Privacy-ABCs, in such a way that they are independent from the concrete algorithms or cryptographic components used

Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Stephan Krenn, Anja Lehmann, Gregory Neven, and Franz-Stefan Preiss
IBM Research – Zurich, Switzerland, e-mail: {pbi, jca, md, enr, skr, anj, nev, frp}@zurich.ibm.com

Ioannis Krontiris, Kai Rannenberg, and Ahmad Sabouri
Chair of Mobile Business & Multilateral Security, Goethe University Frankfurt, Germany, e-mail: {kai.rannenberg, ahmad.sabouri}@m-chair.de

Christian Paquin
Microsoft Research, USA, e-mail: cpaquin@microsoft.com

underneath. The functional decomposition foresees possible architectural extensions to additional functional modules that may be desirable and feasible using future Privacy-ABC technologies or extensions of existing ones.

2.1 Concepts and Features of Privacy-ABCs

The predominant way to authenticate users on the Internet today is by usernames and passwords. When creating accounts, users often additionally have to provide a list of self-claimed attributes such as their name, address, or birth date. Only a few attributes, such as email addresses and credit card numbers, have some external mechanism to check their authenticity; all other attributes are essentially self-claimed.

Technical solutions such as the Security Assertion Markup Language (SAML), OpenID, or X.509 certificates let users authenticate and transfer trusted attributes, certified by issuers, to relying parties. Such technologies are slowly gaining momentum but present considerable security and privacy concerns. Briefly, either an online issuer unnecessarily exposes the issuance key to online attacks and learns the details of all transactions between users and relying parties, or the relying party learns more attributes than necessary, thereby becoming an attractive target for hackers.

Privacy-preserving attribute-based credentials (Privacy-ABCs) are a superior solution offering the best of both worlds: Issuers do not have to be involved during authentication, while users disclose only those attributes required by the relying parties and can do so without being easily traceable across their transactions.

2.1.1 *User Attributes*

We view a user's identity as a set of attributes. In most cases these attributes are information a party knows about a user. So, a user "identity" exists only in connection to a party. Because different parties know different things about the same user, every user has many different partial identities, possibly even multiple identities with each party he or she interacts with. To verify the authenticity of a user's attributes, a party (often called Relying Party (RP)) can either perform identity vetting on the attributes itself (for example, require the user to provide physical documents or take an exam) or rely on a specialized issuer whose identity-vetting procedures it trusts.

For example, in Figure 2.1, Alice has many different attributes, subsets of which make up Alice's different identities with the people and institutions she interacts with online. Alice should be able to manage these identities the same way she manages them in a paper-based world. Identities sharing a unique attribute can of course be linked; for example, her social security number can be linked across her healthcare-related identities, but her other identities should remain unlinkable.

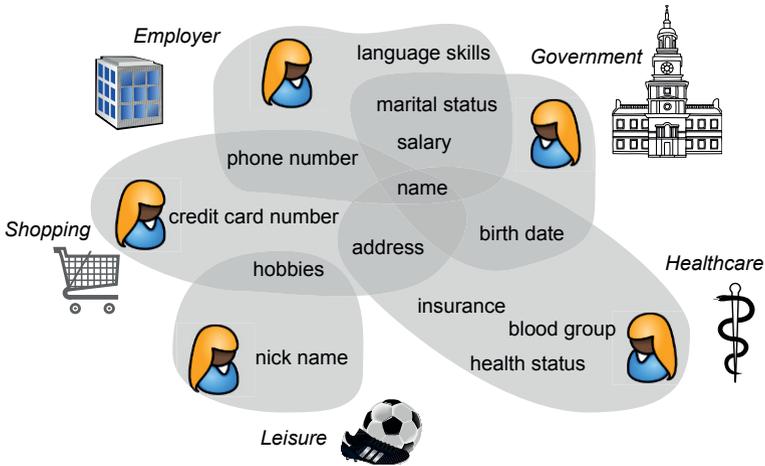


Fig. 2.1 Partial identities as subsets of attributes

Such user-centric identity management requires two basic mechanisms: one to transfer certified attributes from an issuer to a verifier, and one to authenticate (or re-authenticate) a user under a previously established identity. The former mechanism is essential to conduct trusted electronic transactions and requires cryptography. The latter mechanism can in principle be realized with a simple username and password, but this provides poor security guarantees. Indeed, passwords are well known to be vulnerable to password guessing, phishing, and social-engineering attacks. Their insecurity affects privacy, too. To alleviate these shortcomings, many service providers collect as much side information (for example, location or transaction history) about users as they can and analyze that data to detect suspicious behavior and potential breaches. So, a stronger cryptographic mechanism for authentication involving public-key cryptography seems advisable.

In our paper-based world, attribute transfer and authentication are often folded into one mechanism. For instance, a driver’s license transfers the attribute “I’m allowed to drive a car” from the issuer to any relying party and, via the photo on it, provides an authentication mechanism. When realizing attribute transfer and authentication for the digital world, mimicking the paper-based solutions, as often happens, isn’t enough. Instead, one must consider the very different environment: digital data is easily copied and virtually impossible to control once released. So, any digital realization must follow the principle of data minimization. When a user transfers an attribute from an issuer to a relying party, neither party should be able to learn any information that the transferred attribute hasn’t already revealed, even if the parties collaborated.

Of course, an identity management system adhering to these principles doesn’t eliminate all the digital world’s dangers. Communication and stored information should always be encrypted. Sensitive data should be accompanied with usage poli-

cies defining how to treat it, who can use it, for what purpose it's to be used, and when to delete it. We do not elaborate on these issues here; rather, we concentrate on the identity management mechanisms.

Roughly, existing solutions to transfer certified user attributes from an issuer to a relying party are either offline or online. Offline solutions involve the issuer only at the time of issuance. Online solutions also actively involve the issuer during attribute transfer.

2.1.2 Existing Solutions

The most prominent offline solution are X.509 v3 certificates with attribute extensions. Here, the issuer or certificate authority (CA) signs the user's public key together with his or her attributes and includes the signature in the certificate. Since all attributes are needed to verify the CA's signature, the user is forced to reveal all of the attributes in the certificate when transferring an attribute. Moreover, the user's public key acts as a unique identifier that follows the user across all of his or her online transactions.

In online solutions, the user first authenticates directly to the issuer. The issuer then creates a verifiable token for the specific set of attributes required by the relying party. Popular examples following this approach include SAML and WS-Federation, as well as the more lightweight OpenID. The advantage of this approach is that only the required attributes are revealed. However, the issuer learns which user authenticates to which relying party at which time. Although some protocols may optionally hide the user's identity from the verifier or hide the verifier's identity from the issuer, this doesn't help when verifiers and issuers compare their transaction logs. Moreover, with online solutions, the issuance key must be on a system that's permanently connected to the Internet. This considerably increases the issuer's vulnerability to intruders, thus endangering the entire system's security.

2.1.3 Basic Concepts of Privacy-ABCs

Privacy-ABCs are similar to the offline approach in terms of the overall functionality and provided security guarantees, while letting users control and separate their different partial identities.

Similarly to X.509 certificates, users' credentials in Privacy-ABC systems are essentially signatures by the issuer on the attribute values assigned to the user. Unlike X.509 certificates, however, the user can hide some of the attribute values while keeping the issuer's signature verifiable to a verifier.

This section provides a detailed explanation on the features supported by Privacy-ABCs, on the different involved entities, and on the type of interactions that they engage in.

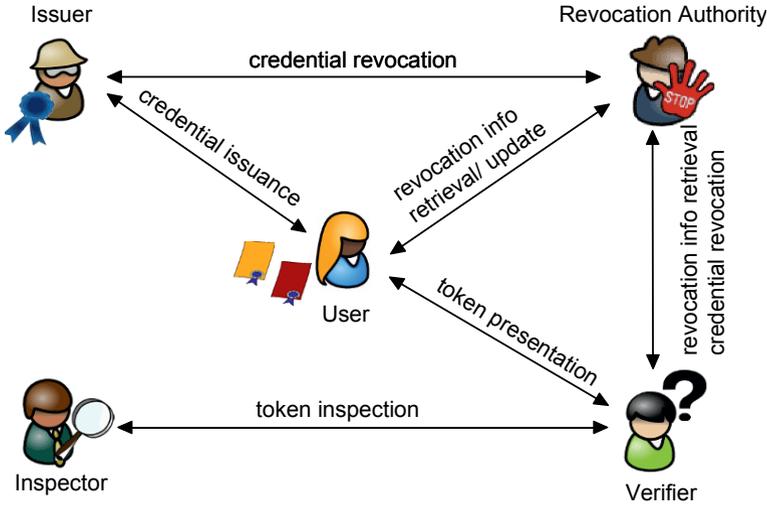


Fig. 2.2 Entities and interactions diagram

Figure 2.2 gives an overview of the different entities and the interactions between them.

- The *user* is at the center of the picture, collecting credentials from various issuers and controlling which information from which credentials she presents to which verifiers. The human user is represented by her user agent, a software component running either on a local device (e.g., on the user’s computer or mobile phone) or remotely on a trusted cloud service. The user may own special hardware tokens to which credentials can be bound to improve security. In the identity management literature, the user is sometimes referred to as the requestor or subject.
- An *issuer* issues credentials to users, thereby vouching for the correctness of the information contained in the credential with respect to the user to whom the credential is issued. Before issuing a credential, the issuer may have to authenticate the user, which it may do using Privacy-ABCs, using a different online mechanism (e.g., username and password), or using out-of-band communication (e.g., by requiring the user to physically present herself at the issuer’s office). In the identity management literature, the issuer is sometimes referred to as the identity (service) provider or attribute authority.
- A *verifier* protects access to a resource or service that it offers by imposing restrictions on the credentials that users must own and the information from these credentials that users must present in order to access the service. The verifier’s restrictions are described in its *presentation policy*. The user generates from her credentials a *presentation token* that contains the information required by the presentation policy and the supporting cryptographic evidence. In the identity management literature, the verifier is sometimes also referred to as the relying party, server, or service provider.

- A *revocation authority* is responsible for revoking issued credentials, so that these credentials can no longer be used to generate a presentation token. The use of a particular revocation authority may be imposed by the issuer, in which case the revoked credentials cannot be used with any verifier for any purpose, or by the verifier, in which case the effect of the revocation is local to the verifier and does not affect presentations with other verifiers. Both the user and the verifier must obtain the most recent revocation information from the revocation authority to generate, respectively verify, presentation tokens.
- An *inspector* is a trusted authority who can de-anonymize presentation tokens under specific circumstances. To make use of this feature, the verifier must specify in the presentation policy which inspector should be able to recover which attribute(s) under which circumstances. The user is therefore aware of the de-anonymization options when the token is generated and actively participates to make this possible; therefore the user can make a conscious decision based on her trust in the inspector.

In an actual deployment, some of the above roles may actually be fulfilled by the same entity, or split among many. For example, an issuer can at the same time play the role of revocation authority and/or inspector, or an issuer could later also be the verifier of tokens derived from credentials that it issued.

Moreover, some of the flows presented in this document could be adapted for particular deployments and scenarios. It is assumed that verifiers have a reliable way of obtaining the public information of issuers and revocation authorities needed to validate presentation tokens, for example, by certifying the information through a classical public-key infrastructure (PKI).

2.1.3.1 Credentials

A *credential* is a certified container of attributes issued by an issuer to a user. An attribute is described by the attribute type, that describes the meaning of the attribute (e.g., first name), and the attribute value, that gives 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 user can then later use her credentials to derive presentation tokens that reveal partial information about the encoded attributes to a verifier.

The *credential specification* specifies the list of attribute types that are encoded in a credential. Since Privacy-ABCs natively only support integers of limited size (typically 256 bits) as attribute values, the credential specification also specify the *encoding mechanism* that maps attribute values to their integer representation. Depending on the data type and the size of the attribute value, this encoding may involve a cryptographic hash function.

At setup, the issuer generates public *issuer parameters* and a secret *issuance key*. The issuer parameters are used by verifiers to verify the authenticity of presentation tokens. Trust management and distribution are out of scope of this specification; a standard PKI, e.g., using hierarchical certification authorities, can be used to ensure

that verifiers obtain authentic copies of the credential specifications and issuer parameters. Apart from cryptographic information, the issuer parameters also contain other meta-data such as the hash algorithm to use to create presentation tokens, as well as information for key binding, and revocation (see later). The issuer keeps the issuance key strictly secret and uses it only to issue credentials.

2.1.3.2 Presentation

To provide certified information to a verifier (for authentication or an access decision), the user uses one or more of her credentials to derive a *presentation token* and sends it to the verifier. A single presentation token can contain information from any number of credentials. The token can reveal a subset of the attribute values in the credentials (e.g., `IDcard.firstname = "John"`), prove that a value satisfies a certain predicate (e.g., `IDcard.birthdate < 1993/01/01`) or that two values satisfy a predicate (e.g., `IDcard.lastname = creditcard.lastname`). Apart from revealing information about credential attributes, the presentation token can optionally sign an application-specific message and/or a random nonce to guarantee freshness. Moreover, presentation tokens support a number of advanced features such as pseudonyms, key binding, inspection, and revocation that are described in more details below.

A verifier announces in its *presentation policy* which credentials from which issuers it accepts and which information from these credentials must be revealed in the presentation token. The verifier can cryptographically verify the authenticity of a received presentation token using the credential specifications and issuer parameters of all credentials involved in the token. The verifier must obtain the credential specifications and issuer parameters in a trusted manner, e.g., by using a traditional PKI to authenticate them or retrieving them from a trusted location.

The presentation token created in response to such a presentation policy consists of the presentation token description, containing a mechanism-agnostic description of the revealed information, and the presentation token evidence, containing opaque technology-specific cryptographic data in support of the token description. Presentation tokens based on Privacy-ABCs are cryptographically unlinkable and untraceable by default, meaning that verifiers cannot tell whether two presentation tokens were derived from the same or from different credentials, and that issuers cannot trace a presentation token back to the issuance of the underlying credentials. However, in what follows we will discuss additional mechanisms that, with the user's consent, introduce intentional linkability, or allow a dedicated third party to recover the user's identity.

Obviously, presentation tokens are only as unlinkable as the information that they intentionally reveal. For example, tokens that explicitly reveal a unique attribute (e.g., the user's social security number) are fully linkable. Moreover, pseudonyms and inspection can be used to purposely create linkability across presentation tokens (e.g., to maintain state across sessions by the same user) and create traceability of presentation tokens (e.g., for accountability reasons in case of abuse). Finally,

Privacy-ABCs have to be combined with anonymous communication channels (e.g., Tor onion routing) to avoid linkability in the “layers below”, e.g., by the IP addresses in the underlying communication channels or by the physical characteristics of the hardware device on which the tokens were generated.

2.1.3.3 Key Binding

Credentials can optionally be *bound* to a user’s secret key, i.e., a cryptographically strong random value that is generated by and known only to a particular user. The credential specification specifies whether the credentials issued according to this specification are to employ key binding or not. A presentation token derived from such a key-bound credential always contains an implicit proof of knowledge of the underlying secret key, so that the verifier can be sure that the secret key of the credential was involved in the creation of the presentation token.

Key-bound credentials can optionally be issued in such a way that the newly issued credential is bound to the same secret key as an existing credential already owned by the user — without the issuer learning the secret key in the process (see Section 2.1.3.6). A verifier can also optionally impose in its presentation policy that all key-bound credentials involved in the creation of the token must be bound to the same secret key.

Key binding can be used for several purposes. First, it can be used to prevent users from “pooling” their credentials, i.e., sharing their credentials with other users. In a presentation involving multiple credentials, the verifier can optionally insist that all credentials must be bound to the same user secret, so that credentials issued to different users cannot be used together. For this to work, users must be prevented from choosing the same secret key and from sharing their secret key with others. The former can be done by letting the secret be generated by a trusted hardware device such as a smartcard, through a joint generation between the issuer and user (see advanced issuance in Section 2.1.3.6), or by requiring the user to establish a scope-exclusive pseudonym at issuance and making sure that no two users have the same pseudonym (see Section 2.1.3.4). The latter can be enforced by making some highly valuable information or services accessible with the user secret alone, e.g., by protecting access to the user’s main e-government account through a pseudonym derived from the same secret key.

Second, by storing the user secret on a trusted hardware device such as a smartcard, the credentials can be bound to the device. That is, if the key cannot be extracted from the device, but the device does participate in the generation of presentation tokens, then credentials cannot be used without the physical presence of the device.

Finally, key binding can be used to prevent users from being “framed” by a malicious issuer, i.e., from being impersonated by the issuer towards a verifier. A malicious issuer can of course always generate all the credentials that she wants, but she can only do so for a user secret that is different from the real user’s secret. By letting users establish scope-exclusive pseudonyms at issuance and at presentation, the

user can later prove that a presentation token was generated using a different user secret than the one used at issuance. Some external mechanism must be in place to prevent the issuer from tampering with the list of issued pseudonyms, for example, by letting the user sign (digitally or on paper) the pseudonym and then checking this signature.

2.1.3.4 Pseudonyms

There are many situations where a controlled linkability of presentation tokens is actually desirable. For example, web services may want to maintain state information per user or user account to present a personalized interface, or conversation partners may want to be sure to continue a conversation thread with the same person that they started it with.

Privacy-ABCs have the concept of *pseudonyms* to obtain exactly such controlled linkability. If the secret key from Section 2.1.3.3 is seen as the equivalent of a user's secret key in a classical public-key authentication system, then a pseudonym is the equivalent of the user's public key. Just like a public key, it is derived from the user's secret key and can be given to a verifier so that the user can later re-authenticate by proving knowledge of the secret key underlying the pseudonym. Unlike public keys of which there is only one for every secret key, however, users can generate an unlimited number of unlinkable pseudonyms for a single secret key. Users can thus be known under different pseudonyms with different verifiers, yet authenticate to all of them using the same secret key.

To be able to re-authenticate under a previously established pseudonym, the user may need to store some additional information used in the generation of the pseudonym. To assist the user in keeping track of which pseudonym she used at which verifier, the verifier's presentation policy specifies a pseudonym scope, which is just a string that the user can use as a key to look up the appropriate pseudonym. The scope string could for example be the identity of the verifier or the URL of the web service that the user wants to access.

We distinguish between the following three types of pseudonyms:

- *Verifiable pseudonyms* are pseudonyms derived from an underlying secret key as described above, allowing the user to re-authenticate under the pseudonym by proving knowledge of the secret key. Presenting a verifiable pseudonym does not involve presenting any related credentials and is useful in login scenarios, e.g., to replace usernames/passwords.
- *Certified pseudonyms* are verifiable pseudonyms derived from a secret key that also is bound to an issued credential. By imposing same-key binding in the presentation policy and token, a single presentation token can therefore prove ownership of a credential and at the same time establish a pseudonym based on the same secret key. As an example, a student could create several personas on a school discussion board using its core student credential, presenting the pseudonym associated with each persona, and without the possibility of anyone else (including

a malicious issuer) to present a matching pseudonym to hijack's the user's identity.

- *Scope-exclusive pseudonyms* are certified pseudonyms that are guaranteed to be unique for a specific scope string and secret key. For normal (i.e., non-scope-exclusive) pseudonyms, nothing prevents a user from generating multiple unlinkable pseudonyms for the same scope string. For scope-exclusive pseudonyms, it is cryptographically impossible to do so. By imposing a scope-exclusive pseudonym to be established, a verifier can be sure that only a single pseudonym can be created for each credential or combination of credentials that are required in the presentation. This feature can be useful to implement a form of “consumption control” in situations (e.g., online petitions or one-time coupons) where users must remain anonymous to the verifier but should not be allowed to create multiple identities based on a single credential. Note that scope-exclusive pseudonyms for different scope strings are still unlinkable, just like normal verifiable pseudonyms.

2.1.3.5 Inspection

Absolute user anonymity in online services can easily lead to abuses such as spam, harassment, or fraud. Privacy-ABCs give verifiers the option to strike a trade-off between anonymity for honest users and accountability for misbehaving users through a feature called inspection.

An inspector is a dedicated entity, separate from the verifier, who can be asked to uncover one or more attributes of the user who created a presentation token, e.g., in case of abuse. The inspector must on one hand be trusted by the user not to uncover identities unnecessarily, and must on the other hand be trusted by the verifier to assist in the recovery when an abuse does occur.

Presentation tokens are fully anonymous by default, without possibility of inspection. To enable an inspector to trace a presentation token when necessary, the presentation policy must explicitly specify the identity of the inspector, which information the inspector must be able to recover, and under which circumstances the inspector can be asked to do so. The user then creates the presentation token in a particular way so that the verifier can check by himself, i.e., without help from the inspector, that the token could be inspected under the specified restrictions if necessary.

In more technical detail, the inspector first sets up a public encryption key and a secret decryption key; he makes the former publicly available but keeps the latter secret. The presentation policy specifies

- (a reference to) the inspector's public key,
- which attribute(s) from which credential(s) which inspector must be able to recover, and
- the *inspection grounds*, i.e., an arbitrary human- and/or machine-readable string describing the circumstances under which the token can be inspected.

The user then prepares the presentation token so that it contains encrypted versions of the requested attribute values under the respective public key of the suggested inspector, together with a verifiable cryptographic proof that the encryption contains the same attribute values as encoded in the user's credentials and certified by the issuer.

When the situation described in the inspection grounds arises, the inspection requester can ask for an inspection. Besides the verifier, other entities such as criminal prosecutors, courts or the user herself are also potential requesters for inspection. Usually the verifier holds the stored copy of the presentation token and will send it to the inspector for inspection, possibly together with some kind of evidence (e.g., transaction logs, inquiry of competent authority, court order) that the inspection grounds have been fulfilled. The inspection grounds are cryptographically tied to the presentation token, so the verifier cannot change these after having received the token. The inspector uses its secret key to decrypt the encrypted attribute values and returns the cleartext values to the inspection requestor.

De-anonymization of presentation tokens is probably the main use case for inspection, but it can also be used to reveal useful attribute values to third parties instead of to the verifier himself. For example, suppose the verifier is an online merchant wishing to accept credit card payments without running the risk of having the stored credit card data stolen by hackers. In that case, he can make the user encrypt her credit card number under the public key of the bank by specifying the bank as an inspector for the credit card number with "payment" as inspection grounds.

2.1.3.6 Credential Issuance

In the simplest setting, an issuer issues credentials to users "from scratch", i.e., without relation to any existing credentials already owned by the users. In this situation, the user typically has to convince the issuer through some out-of-band mechanism that she qualifies for a credential with certain attribute values, e.g., by showing up in person at the issuer's office and showing a physical piece of ID, or by bootstrapping from a government-issued electronic identity. Credential issuance is a multi-round interactive protocol between the issuer and the user. The attribute values can be specified by either parties, or jointly generated at random (i.e., the issuer can be ensured that an attribute value is chosen randomly and not chosen solely by user, but without the issuer learning the attribute value).

Privacy-ABCs also support a more advanced form of credential issuance where the information embedded in the newly issued credential is *carried over* from existing credentials already owned by the user, without the issuer being able to learn the carried-over information in the process. In particular, the newly issued credential can

1. carry over attribute values from an existing credential,
2. contain "self-claimed" attribute values, i.e., values chosen by the user himself,
3. be bound to the same secret key as an existing credential or verifiable pseudonym,

all without the issuer being able to learn the carried-over attribute values or secret key in the process.

Moreover, the issuer can insist that certain attributes be generated jointly at random, meaning that the attribute will be assigned a fresh random value. The issuer does not learn the value of the attribute, but at the same time the user cannot choose, or even bias, the value assigned to the attribute. This feature is for instance helpful to impose usage limitation of a credential. To this end, the issuer first embeds a jointly random value as serial number in the credential. A verifier requesting a token based on such a credential can require that its serial number attribute must be disclosed by the user, such that it can detect if the same credential is used multiple times. The jointly random attribute hereby ensures that the verifier and issuer cannot link the generated token and issued credential together, and the user can not cheat by biasing the serial number in the credential.

The issuer publishes or sends to the user an *issuance policy* consisting of a presentation policy and a *credential template*. The presentation policy expresses which existing credentials the user must possess in order to be issued a new credential, using the same concepts and format as the presentation policy for normal token presentation. The user prepares a special presentation token that fulfills the stated presentation policy, but that contains additional cryptographic information to enable carrying over attributes and user secrets. The credential template describes the relation of the new credential to the existing credentials used in the presentation token by specifying

- which attributes of the new credential will be assigned the same value as which attributes from which credential in the presentation token,
- whether the new credential will be bound to the same secret key as one of the credentials or pseudonyms in the presentation token, and if so, to which credential or pseudonym.

The user and issuer subsequently engage in a multi-round issuance protocol, at the end of which the user obtains the requested credential.

2.1.3.7 Revocation

No identification system is complete without an appropriate revocation mechanism. There can be many reasons to revoke a credential. For example, the credential and the related user or device 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. Moreover, credentials may be revoked for a restricted set of purposes. For example, a football hooligan's digital identity card could be blocked from accessing sport stadiums, but is still valid for voting or submitting tax declarations.

In classical public-key authentication systems, revocation usually works by letting either the issuer or a dedicated revocation authority publish the serial numbers of revoked certificates in a so-called certificate revocation list. The verifier then simply checks whether the serial number of a received certificate is on such a list or not.

The same approach does not work for Privacy-ABCs, however, as Privacy-ABCs should not have a unique fingerprint value that must be revealed at each presentation, as this would nullify the unlinkability of the presentation tokens. Nevertheless, there are cryptographically more advanced revocation mechanisms that provide the same functionality in a privacy-preserving way, i.e., without imposing a unique trace on the presentation tokens. We describe an abstract interface that covers all currently known revocation mechanisms.

Credentials are revoked by dedicated *revocation authorities*, which may be separate entities, or may also take the joint role of issuer or verifier. The revocation authority publishes its *revocation parameters* and regularly (e.g., at regular time intervals, or whenever a new credential is issued or revoked) publishes the most recent *revocation information* that verifiers use to make sure that the credentials used in a presentation token have not been revoked. The revocation parameters contain information where and how the verifiers can obtain the most recent revocation information. Depending on the revocation mechanism, the identifiers of revoked credentials may or may not be visible from the revocation information. It is important that verifiers obtain the most recent revocation information from the revocation authority directly, or that the revocation information is signed by the revocation authority if it is provided by the user together with the presentation token.

In order to prove that their credentials have not been revoked, users may have to maintain *non-revocation evidence* for each credential and for each revocation authority against which the credential must be checked. The first time that a user checks one of her credentials against a particular revocation authority, she obtains an initial non-revocation evidence. Later, depending on the revocation mechanism used, the user may have to obtain regular non-revocation evidence updates at each update of the revocation information. Also depending on the revocation mechanism, these evidence updates may be the same for all users/credentials or may be different for each user/credential. In the latter case, again depending on the mechanism, the users may fetch their updates from a public bulletin board or obtain their updates over a secure channel.

We distinguish between two types of revocation. Apart from a small list of exceptions, all revocation mechanisms can be used for either type of revocation.

- In *issuer-driven revocation*, the issuer specifies, as part of the issuer parameters, the revocation authority (and revocation parameters) to be used when verifying a presentation token involving a credential issued by his issuer parameters. Issuer-driven revocation is always global in scope, meaning that any presentation token must always be checked against the most recent revocation information from the specified revocation authority, and that the issuer denies any responsibility for revoked credentials. Issuer-driven revocation is typically used when credentials have been lost or compromised, or when the user is denied any further use of the credential. The revocation authority may be managed by or be the same entity as the issuer, or may be a separate entity. Issuer-driven revocation is performed through a *revocation handle*, a dedicated unique identifier that the issuer embeds as an attribute in each issued credential (but that should not be unnecessarily revealed in a presentation token). When the issuer, a verifier, or any third party

wants to revoke a credential, it must provide the revocation handle to the revocation authority. How the party requesting the revocation learns the revocation handle is out of the scope of this document; this could for example be done digitally by insisting in the presentation policy that the revocation handle be revealed to a trusted inspector, or physically by arresting the person and obtaining his or her identity card.

- *Verifier-driven revocation* essentially allows the verifier to “black-list” certain credentials and prevent them from being used for authentication. The verifier specifies as part of the presentation policy against which revocation authority or authorities (and revocation parameters) the presentation must additionally be checked, i.e., on top of any revocation authorities specified by the issuer in the issuer parameters. The effect of the revocation is local to those verifiers who explicitly specify the revocation authority in their presentation policies, and does not affect presentations with other verifiers. Verifier-driven revocation is mainly useful for purpose-specific revocation (e.g., a no-fly list for terrorists) or verifier-local revocation (e.g., a website excluding misbehaving users from its site). Note that if unlinkability of presentation tokens is not a requirement, the latter effect can also be obtained by using scope-exclusive pseudonyms. The revocation authority may be managed by or be the same entity as the verifier, or may be a separate entity. Verifier-driven revocation can be performed based on any attribute, not just based on the revocation handle as for issuer-driven revocation. It is up to the verifier and/or the revocation authority to choose an attribute that on the one hand is sufficiently identifying to avoid false positives (e.g., the user’s first name probably doesn’t suffice) and on the other hand will be known to the party likely to request the revocation of a credential. Verifier-driven revocation is essentially a black list of attribute values, banning all credentials with a blacklisted attribute value.

2.1.4 Security and Privacy Features

Privacy-ABCs are a combination of several cryptographic building blocks, including signatures, pseudonyms, zero-knowledge proofs, encryption, and revocation mechanisms. Properly defining the security and privacy guarantees offered by such an encompassing framework is not an easy task. On a scientific level, the ABC4Trust project has made great advances in this respect by creating the most comprehensive formal security notions of Privacy-ABCs so far [CKL⁺14]. In this section, we avoid technical details of cryptographic security notions, but rather give an intuitive description of the security and privacy features that application developers can expect when working with Privacy-ABCs.

Roughly, one could summarize the security and privacy features of Privacy-ABCs as security meaning that users cannot create valid presentation tokens without having the proper underlying credentials and keys, while privacy guarantees that presentation tokens do not reveal any more information than what was intentionally

disclosed. The various features of Privacy-ABCs deserve a more detailed discussion, which we give in the following.

2.1.4.1 Basic Presentation

The most basic security guarantee is that credentials in a Privacy-ABC system are unforgeable. This means that users, without access to an issuer's secret key, cannot create new credentials or change attribute values in the credentials they obtained from that issuer. Presentation tokens are unforgeable as well, in the sense that in order to create a valid presentation token that discloses a number of attribute values or proves a number of (in)equality predicates, the user must possess credentials that satisfy the disclosed criteria. Note that this unforgeability only holds as long as the verifier can obtain authentic copies of the issuers' public keys, e.g., by certifying issuers' keys using an external PKI.

Presentation tokens can optionally "sign" a message that can contain a nonce, the intended verifier's identity, or any application-provided content. The information in that message is immutable: without the necessary credentials to regenerate a complete presentation token, one cannot change the message signed by the presentation token. The nonce in the signed message can be used to prevent replay attacks, where an eavesdropper or cheating verifier reuses a presentation token generated by an honest user to re-authenticate to the same or to a different verifier. Including the verifier identity (e.g., its URL or public key) in the signed message prevents man-in-the-middle attacks where a cheating verifier relays presentation tokens from honest users to authenticate itself to a second verifier. The application layer on the user's side must check that the verifier identity included in the signed message matches the application's intended verifier.

In terms of privacy, presentation tokens only reveal the information that is explicitly disclosed by the token. This means for example that presentation tokens reveal no information at all about the values of hidden credential attributes. If the presentation token includes attribute predicates, the token reveals nothing beyond the proof of the predicate, and in particular does not reveal the exact value of the involved attributes. It also means that presentation tokens are unlinkable, in the sense that even a collusion of issuers and verifiers cannot tell whether two presentation tokens were created by the same user or by different users, and cannot trace the presentation back to the issuance of the credentials.

Of course, unlinkability is only guaranteed to the extent that neither the disclosed attributes themselves nor the communication layer introduce trivial correlations between a user's presentations. In particular, it is important that presentation takes place over an anonymous communication channel, e.g., using Tor onion routing, to avoid that the verifier can link visits by the same user through his IP address. Achieving unlinkability at the physical layer can be particularly hard: intrinsic hardware characteristics of the user's device such as clock skews may be exploitable as unique device fingerprints [KBC05].

2.1.4.2 Key Binding

A key-bound credential cannot be used in a presentation without knowledge of the user secret. If the user secret is generated and stored on a trusted hardware device such as a smartcard, this means that the creator of the presentation token must have access to the device at the time of presentation. The presentation policy can optionally insist that different key-bound credentials or pseudonyms are bound to the *same* secret key. In this case, the policy cannot be satisfied using credentials or pseudonyms that are bound to or derived from different keys; the presentation token does not leak any information about the value of the key, however.

2.1.4.3 Advanced Issuance

In an advance issuance protocol, the user essentially performs a presentation before proceeding with the issuance. The same security and privacy properties hold for the issuance token as for normal presentation. Additionally, the issuance can carry over attribute values and user secrets from credentials involved in the presentation. In this case, the issuer is guaranteed that the attribute values or key in the newly issued credential are equal to those of the original credentials used in the presentation, but he doesn't see the actual value. For self-claimed attribute values, there is no such guarantee; the issuer blindly signs any attribute value that the user chooses. Jointly random attributes are guaranteed to be truly random, meaning that the user cannot steer or bias the distribution in any way, but the issuer again doesn't see the actual value. The user always sees all attribute values in his credentials.

2.1.4.4 Pseudonyms

Verifiable and certified pseudonyms can be seen as public keys corresponding to a user's secret key, with the main difference that the user can generate arbitrarily many pseudonyms from a single user secret. Pseudonyms are unlinkable, in the sense that verifiers cannot tell whether two pseudonyms originated from the same user secret or from different user secrets. Knowledge of the underlying secret key is required to create a valid presentation token involving a pseudonym. An attacker therefore cannot successfully authenticate under a pseudonym that was established by an honest user. This also implies that two honest users with independent user secrets will never accidentally generate the same pseudonym (because otherwise an adversary could generate pseudonyms for his own user secret until he hits an already established pseudonym).

Scope-exclusive pseudonyms are unique per scope and per user secret. Meaning, for a given scope string and a given user secret, there is only one scope-exclusive pseudonym for which a valid presentation token can be generated. Scope exclusive pseudonyms are unlinkable in the sense that, without knowing the user secret,

one cannot tell whether two scope-exclusive pseudonyms for different scope strings were derived from the same or from different user secrets.

2.1.4.5 Inspection

Inspection allows the user to encrypt one or more attribute values under the public key of a trusted inspector. The encryption is secure against chosen-ciphertext attacks, meaning that the encrypted attribute values remain hidden even when the adversary can guess the encrypted value or can ask the inspector to inspect other presentation tokens. The user must encrypt his real attribute values for which he has valid credentials. Any attempt by the user to encrypt a different value, or to make the ciphertext undecryptable, will be detected by the verifier as an invalid presentation token. Finally, the inspection grounds are clear to the user at the time of presentation and are “signed” into the token, so that they cannot be modified afterwards. This prevents a malicious verifier from requesting a presentation token to be inspected based on different grounds than those that the user agreed with.

2.1.4.6 Revocation

When a credential is used in a presentation token with issuer-driven or verifier-driven revocation, the user merely proves that his revocation handle, respectively his combination of attribute values, was not revoked when the revocation authority published the stated revocation information. No other information about the value of the revocation handle or attributes is leaked. It is up to the verifier to check that the revocation information used in the presentation token is indeed the latest one as published by the revocation authority.

Revocation inherently opens up a subtle attack on user privacy by malicious revocation authorities. Namely, a cheating authority can always arbitrarily revoke valid credentials, just to test whether these credentials are involved in an ongoing presentation. The authority could even gradually “close in” on the user during subsequent presentations. External precautions must be taken to prevent such an attack, for example, by requiring that revocations must be logged on a public website or approved by an external auditor.

The communication pattern between users, issuers, and the revocation authority differs considerably for different revocation mechanisms. Some mechanisms follow a whitelist approach, where the revocation authority keeps track of valid revocation handles (attributes) and removes those of revoked credentials. These mechanisms usually require the revocation authority to be involved during credential issuance. Other revocation mechanisms use blacklists, where the revocation authority only keeps track of revoked values.

The revocation information may or may not hide the values of valid and revoked handles; this depends on the actual revocation mechanism that is used. Also depending on the mechanism, users may need to store non-revocation evidence with their

credentials and update it before using it in a presentation. Some mechanisms require individualized updates, meaning that the user has to identify himself towards the revocation authority during the update. If the update occurs right before the presentation, this is a potential privacy leak. It is therefore better to let users perform the update of their non-revocation evidence at regular time intervals, rather than during presentation.

2.2 Architecture Highlights

The architecture of ABC4Trust is defined by following a layered approach, where all Privacy-ABC related functionalities are grouped together in a layer called ABCE (ABC Engine). It provides simple interfaces towards the application layer, thereby abstracting the internal design and structure. More specifically, this means that we define all the technology-agnostic components of the ABCE layer, as well as the APIs they provide. The APIs can be divided into two categories: first the interfaces that the ABCE components offer to the upper layers (e.g. Application), and second the interfaces that the different components within the ABCE layer expose to each other.

Equally important in the architecture is the specification of the data artefacts exchanged between the implicated actors, in such a way that the underlying differences of concrete Privacy-ABCs are abstracted away through the definition of formats that can convey information independently from the mechanism-specific cryptographic data. So the ABC4Trust architecture emphasizes on the XML based specification of the corresponding messages exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials.

The way that the ABC4Trust architecture is designed offers several benefits and facilitates their integration in today's applications. More specifically:

- The API defined by the architecture enables application developers to integrate Privacy-ABCs in their applications without having to think about their cryptographic realization.
- Application developers can implement their own UI for the interaction of the users with Privacy-ABCs, since it is considered to be an independent component and can be replaced and adapted according to the needs of different platforms.
- Users are able to benefit from different Privacy-ABC technologies at the same time on the same hardware and software platforms.
- Service providers and IdSPs are able to adopt whichever Privacy-ABC technology best suits their needs and switch among them with a minimal effort spent on adjusting their infrastructures. In this way, lock-in to specific technologies can be avoided.

The architecture described in this chapter has been implemented by the ABC4Trust project, it has been tested in the deployment of the two pilots and has been published

as a reference implementation. Chapter 9 describes in more details the technical implementation of the architecture.

The architecture allows for deployments in actual production environments and in several application areas. Two specific cases are the ABC4Trust pilots, described in Chapters 6 and 7, where (1) minimal disclosure of identifying information when accessing resources, (2) and the anonymous feedback to a community by accredited members were in focus. Furthermore, some other application scenarios that could benefit from the ABC4Trust architecture are discussed in Chapter 10.

2.3 Architectural Design

Following standard design principles, our architecture uses a layered approach, where related functionalities are grouped into a common layer that provides simple interfaces towards other layers and components, thereby abstracting the internal design and structure. As mentioned in Chapter 1, the architecture focuses on the technology-independent components for Privacy-ABC systems, grouped in the ABCE layer, which can be integrated in various application and deployment scenarios. That is, we do not propose a concrete application-level deployment but provide generic interfaces to the ABCE layer that allow for a flexible integration. Note that we aim at an architecture that is capable of supporting all the privacy-enhancing features of Privacy-ABCs, but at the same time is not exclusive to those, i.e., it is also generic enough to support standard ABC technologies such as X.509 certificates.

The Privacy-ABC architecture defines for each entity the necessary components to operate with attribute-based credentials and to support the various features introduced in Section 2.1. A simplified overview of this architecture is depicted in Figure 2.3.

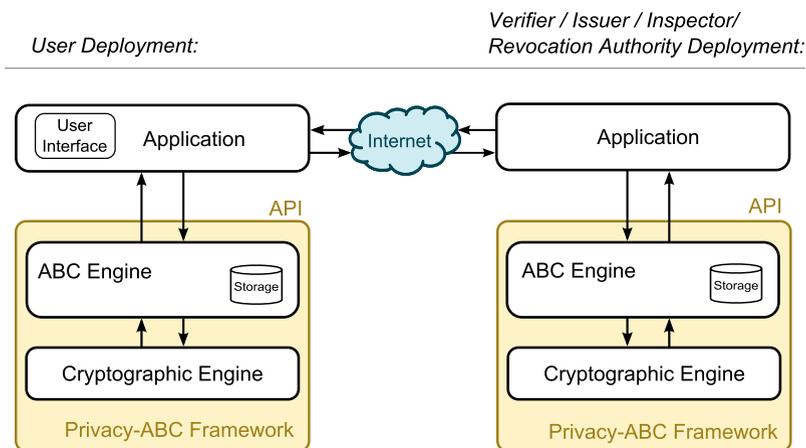


Fig. 2.3 Architecture of a Privacy-ABC System

2.3.1 Overview of the Components

We now briefly describe the different layers in our architecture and give an overview of the internal components of the ABCE layer. The latter is rather for informational purposes only, as the application developer does not have to deal with those internals of the ABCE but only invoked the external APIs. A more detailed view of the Privacy-ABC architecture and its components on the user side is shown in Figure 2.4.

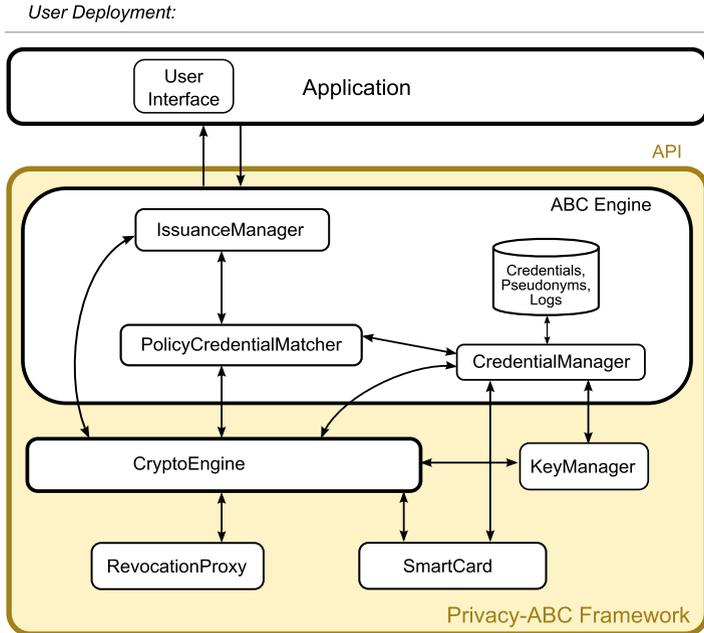


Fig. 2.4 Overview of the Privacy-ABC Architecture on the User Side

2.3.1.1 Application Layer

The application layer is actually not part of the Privacy-ABC architecture, but will operate on top of that. Roughly, the application layer comprises all application-level components, which in the case of the user-side deployment include the main application and the user interface for the identity selection (see description below). The application layer of verifiers and issuers will also contain the policy store and the access control engine.

UserInterface (User): The UserInterface displays the possible choices of pseudonyms and credentials a user can apply in an issuance or presentation session. To this

end, it shows a human-friendly description of the credentials and presentation/issuance token, namely, the information that will be revealed by presenting the token.

2.3.1.2 ABCE Layer

The ABCE layer is the core of our Privacy-ABC architecture and contains all technology-agnostic methods and components for a Privacy-ABC system. That is, it contains, e.g., the methods to parse an obtained issuance or presentation policy, perform the selection of applicable credentials for a given policy or to trigger the mechanism-specific generation or verification of the cryptographic evidence. The ABCE layer is invoked by the application-layer and calls out to the CryptoEngine to obtain the mechanism-specific cryptographic data. To perform their tasks, the internal components can also make use of other external components such as the KeyManager, Smartcard or the RevocationProxy.

IssuanceManager (*User, Issuer*): The IssuanceManager receives the incoming issuance messages and routes them either to the CryptoEngine or to the PolicyCredentialMatcher, depending on the content of the message.

PolicyCredentialMatcher (*User*): The PolicyCredentialMatcher prepares a list of choices of credentials, pseudonyms, and inspectors for the UserInterface, based on the policies it receives. When a choice was made by the user, the PolicyCredentialMatcher then provides the CryptoEngine with the description of the selected token and thereby starts the cryptographic proof generation.

PolicyTokenMatcher (*Verifier*): The PolicyTokenMatcher is responsible for checking if a token received from the user matches a given policy. This verification is done in two main steps. First, it checks whether the statements made in the token description satisfy the required statements in the policy. If the policy requested the re-use of an established pseudonym, the PolicyTokenMatcher calls on the TokenManager (described below) to look up if a presented pseudonym already exists. When the first check succeeds, i.e., the token description matches the policy, it subsequently invokes the CryptoEngine which then verifies the validity of the crypto evidence. If the verification of the crypto evidence is successful as well, the PolicyTokenMatcher stores the token in a dedicated store (if requested by the application).

Token Manager (*Verifier, Issuer*): The TokenManager stores the issuance and presentation tokens (including the used pseudonyms) that were accepted by the issuer and the verifier respectively. The issuer's token manager also stores a "history" of the issuances, which consists of the list of issuer-specified attributes (including the revocation handle) and the issuance token for all credentials that were issued.

CredentialManager (*User*): The CredentialManager is responsible for storing all secret or privacy-sensitive info of the user, i.e., credentials, pseudonyms, secrets. It also seamlessly integrates the blobstore on the smartcards (via the smartcard manager) and is responsible for detecting smartcards and getting the PIN of the

card from the user. In the course of an advanced issuance or presentation session the CredentialManager provides the PolicyCredentialMatcher with a list of all credentials and pseudonyms currently available in the storage and on all active smartcards. During issuance it further downloads and caches the default pictures associated with a credential, which are then passed to the PolicyCredentialMatcher and are possibly displayed in a UserInterface.

PrivateKeyStore (Issuer, RevocationAuthority, Inspector): The PrivateKeyStore is available for the issuer, inspector and revocation authority and is responsible for storing private keys which are generated within the ABCE.

2.3.1.3 Crypto Layer

The crypto layer contains all the technology-specific methods needed in a credential life-cycle, e.g., to generate and verify presentation/issuance tokens, inspect attributes or maintain the revocation information. The ABC4Trust reference implementation of our Privacy-ABC framework also provides a rather generic CryptoEngine that currently incorporates U-Prove and Idemix as the main credential component, and also contains cryptographic realizations for all the additional features introduced in the previous Chapter. For a more detailed description of the CryptoEngine we refer to Section 3.1.

CryptoEngine (User, Issuer, Verifier, Revocation Authority, Inspector): The CryptoEngine is responsible for all cryptographic computations in the Privacy-ABC framework. For instance, it creates pseudonyms, non-device-bound secrets, system parameters, key pairs and transforms the presentation/issuance token description into a cryptographic proof or verifies a given cryptographic proof. During issuance, the CryptoEngine of the issuer also interacts with the revocation authority (via the revocation proxy) to generate a new revocation handle and a non-revocation evidence for a new credential. Subsequently, the CryptoEngine also updates the non-revocation evidence of revocable credentials. Furthermore, the CryptoEngine provides mechanism-dependent and human-friendly proof descriptions which specify the information that is actually revealed in a presentation or issuance token and which can be used in the identity selection.

2.3.1.4 Storage & Communication Components

The Privacy-ABC architecture also contains several components that assist the work of the ABCE and Crypto layer, e.g., by providing a trusted public-key store or secure storage (and computation) on an external smartcard. As those components are rather use-case and technology-specific, they are described as individual modules and can be customized depending on the concrete scenario in which Privacy-ABCs are used.

KeyManager (*User, Issuer, Verifier, Revocation Authority*): The KeyManager is responsible for storing trusted public keys, and if needed procuring these keys in an authenticated manner.

RevocationProxy (*User, Issuer, Verifier, Revocation Authority*): The RevocationProxy is responsible for secure communication between the revocation authority and the user/issuer/verifier whenever dealing with revocable credentials. It creates, parses and dispatches revocation messages.

SmartcardManager (*User*): The SmartcardManager is responsible for interacting with smartcards. It allows the seamless operation of several cards in parallel. The smartcard manager is NOT responsible for detecting new cards or asking for the user's PIN: that is the credential manager role.

Smartcard (*User, Inspector*): The Smartcard stores the secret and sensitive data. It can be realized as software or as a physical device, and provides two different interfaces. The `DataInterface` allows one to store the credentials, inspector keys and other sensitive cryptographic objects in the card's blobstore. The `CryptoInterface` provides cryptographic functionality for issuance and presentation that is related to a secret stored on the card.

2.4 Deployment of the Architecture

In this section we describe the high-level APIs provided by our framework, and describe their usage along the main scenarios in a credential lifecycle. The API is based on the reference implementation of a Privacy-ABC framework that was created within the EU project ABC4Trust [abc, BCD⁺14, BBE⁺14]; the source code of that implementation is available at <https://github.com/p2abcengine/p2abcengine>. To focus on the main concepts of our architecture, the following description concentrates on the most significant methods and omits some convenience functions as well as simplifies the behaviour of some of the described methods.

The ABCE exposes technology-agnostic methods to the application developer that allow him to implement all the features introduced in the previous Chapter. In summary, those methods comprise the generation of cryptographic parameters and keys, import of these parameters, generation and verification of presentation tokens, issuance of credentials, inspection of tokens, and revocation of credentials or attributes.

2.4.1 Setup and Storage

To equip all parties in a Privacy-ABC system with the necessary key material, the API provides several methods for generating public and/or private cryptographic parameters.

However, before any entity can create its parameters, the global system parameters have to be generated. This is done by invoking the method `generateSystemParameters` with the desired security level as the input. The method then generates the global system parameters which define the 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. 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 could generate and publish system parameters for various security levels, which are then used by all parties.

For each party, the ABCE then offers a dedicated method to create the corresponding key material. Thereby, the ABCE stores the private parameters in the trusted storage and outputs the public part of the parameters.

Issuer Parameters: When generating issuer parameters, one must (in addition to the system parameters) specify the concrete technology and the maximal number of attributes that can appear in credential specifications that are used in conjunction with these issuer parameters. That number is required as it can influence the issuer parameters, e.g., the issuer parameters of Idemix and U-Prove will contain a dedicated generator for each attribute. Further, if the issuer supports issuer-driven revocation, the method also needs the parameters of the corresponding revocation authority as additional input.

Revocation Authority Parameters: For the generation of the revocation authority parameters, one must specify the locations where users and verifiers can retrieve all the necessary information to obtain or update their state of revocation information and non-revocation evidence. Those comprise the location to obtain the latest revocation information, the location of the initial non-revocation evidence of newly issued credentials, and the location where users can obtain updates to their non-revocation evidence.

User Secret Keys: On the user side, the ABCE allows the creation of private keys to which subsequently credentials can be bound. We note that a user may generate multiple keys by calling this method multiple times. Our reference implementation also supports the storage of private keys on external devices such as smartcards.

The ABCE further provides APIs to store public parameters of other parties. As usual, it must be guaranteed that only authenticated parameters are imported and that the public key storage is kept up-to-date. To later retrieve public parameters from the ABCE again, they are stored together with a UID as unique identifier. Similarly, the ABCE includes methods to import credential specifications which define a particular type of credential.

The main methods to setup and maintain a credential system are listed in Table 2.1. Values in brackets denote that they are optional, i.e., can also be set to *null*.

Table 2.1 ABCE Interfaces for Setup and Storage

GLOBAL & STORAGE APIS
<pre>generateSystemParameters input: int <i>securityLevel</i> output: SystemParameters</pre>
<pre>storeSystemParameters input: SystemParameters output: boolean <i>success</i></pre>
<pre>storeIssuerParameters input: IssuerParameters output: boolean <i>success</i></pre>
<pre>storeInspectorParameters input: InspectorParameters output: boolean <i>success</i></pre>
<pre>storeRevocationAuthorityParameters input: RevocationAuthorityParameters output: boolean <i>success</i></pre>
<pre>storeCredentialSpecification input: CredentialSpecification output: boolean <i>success</i></pre>
ISSUER
<pre>generateIssuerParameters input: URI <i>id</i>, SystemParameters, URI <i>technology</i>, int <i>maximalNumberOfAttributes</i>, [URI <i>revocationAuthorityId</i>] output: IssuerParameters</pre>
INSPECTOR
<pre>generateInspectorParameters input: URI <i>id</i>, SystemParameters, URI <i>technology</i> output: InspectorParameters</pre>
REVOCAION AUTHORITY
<pre>generateRevocationAuthorityParameters input: URI <i>id</i>, SystemParameters, URI <i>technology</i>, URI <i>revocationInfoLocation</i>, URI <i>nonRevocationEvidenceLocation</i>, URI <i>nonRevocationUpdateLocation</i> output: RevocationAuthorityParameters</pre>
USER
<pre>generateUserSecretKey input: SystemParameters output: URI <i>id</i></pre>

2.4.2 Presentation of a Token

The process of presentation is triggered when the application on the user’s side contacts a verifier to request access to a resource (Figure 2.5 – Step 1). Having received the request, the verifier responds with one or more presentation policies, which are aggregated in a *PresentationPolicyAlternatives* object. 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. A detailed specification of a presentation policy is given in Section 2.5.

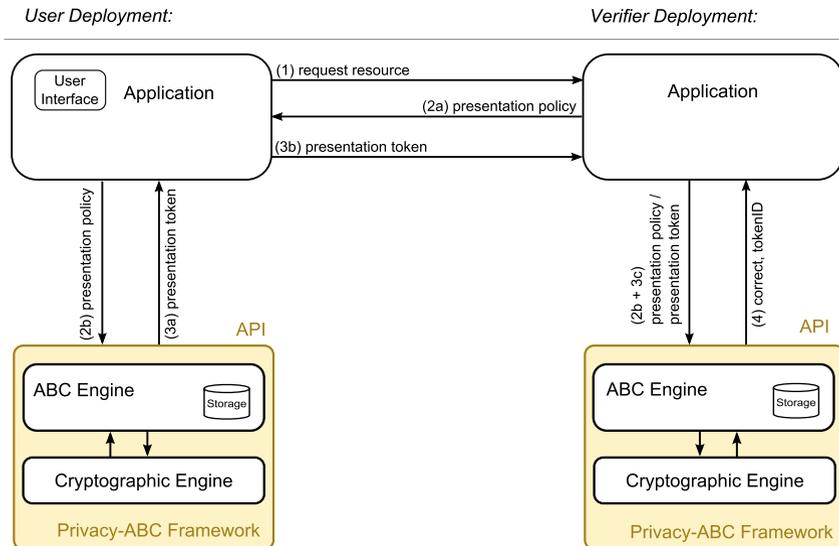


Fig. 2.5 Presentation of a Token (Application Level)

Upon receiving the policy (Figure 2.5 – Step 2.a), the application on the user’s side invokes the Privacy-ABC system first with the `createIdentitySelectorArguments` method on input of the received presentation policy alternatives (Figure 2.5 – Step 2.b). The Privacy-ABC system then determines whether the user has the necessary credentials and pseudonyms to create a token that satisfies the policy. Based on that investigation, the method returns either an object of type *UiPresentationArguments* which describes all the possible combinations of the user’s credentials and pseudonyms that satisfy the policy, or an error message indicating that the policy could not be satisfied. The user’s application layer then performs an identity selection, that is, it invokes a component (such as a graphical user interface) that supports the user in choosing her preferred combination of credentials and pseudonyms and to obtain the user’s consent in revealing her personal data.

The user's choice is recorded in an object of type *UiPresentationReturn* and passed to the `createPresentationToken` method. The Privacy-ABC system then invokes the Crypto Engine to obtain the corresponding cryptographic evidence for the selected token description. The method finally outputs a presentation token (Figure 2.5 – Step 3.a), consisting of the presentation token description and the crypto evidence, according to the user's choice. Afterwards, the presentation token is sent to the verifier (Figure 2.5 – Step 3.b).

When the verifier receives the presentation token from the user, it passes it to its ABCE layer with the method `verifyTokenAgainstPolicy` (Figure 2.5 – Step 2.b+3.c). This method verifies whether the statements made in the presentation token satisfy the corresponding presentation policy alternatives. 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 ABCE outputs a boolean indicating the correct verification and, if requested, stores the presentation token in a dedicated token store, which allows the verifier to subsequently recognize established pseudonyms.

The ABCE interfaces available for the user and verifier in the context of generating and verifying a presentation token are summarized in Table 2.2 below.

Table 2.2 ABCE Interfaces for Token Presentation and Verification

USER
<pre>createIdentitySelectorArguments input: PresentationPolicyAlternatives output: UiPresentationArguments</pre>
<pre>createPresentationToken input: UiPresentationReturn output: PresentationToken</pre>
VERIFIER
<pre>verifyTokenAgainstPolicy input: PresentationToken, PresentationPolicyAlternatives, boolean storeToken output: boolean isCorrect, [URI tokenId]</pre>
<pre>getPresentationToken input: URI tokenId output: PresentationToken</pre>

2.4.3 Issuance of a Credential

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 above. 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 (Figure 2.6 – Step 1) and indicates the credential type she wishes to obtain (Figure 2.6 – Step 2). Note that the exact details of the initial authentication are outside the scope of the Privacy-ABC framework and, for example, can be done using traditional means such as username and password. The issuer triggers the issuance of a credential through the API when receiving a correct credential request from a user. As described in Section 2.1 , 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.

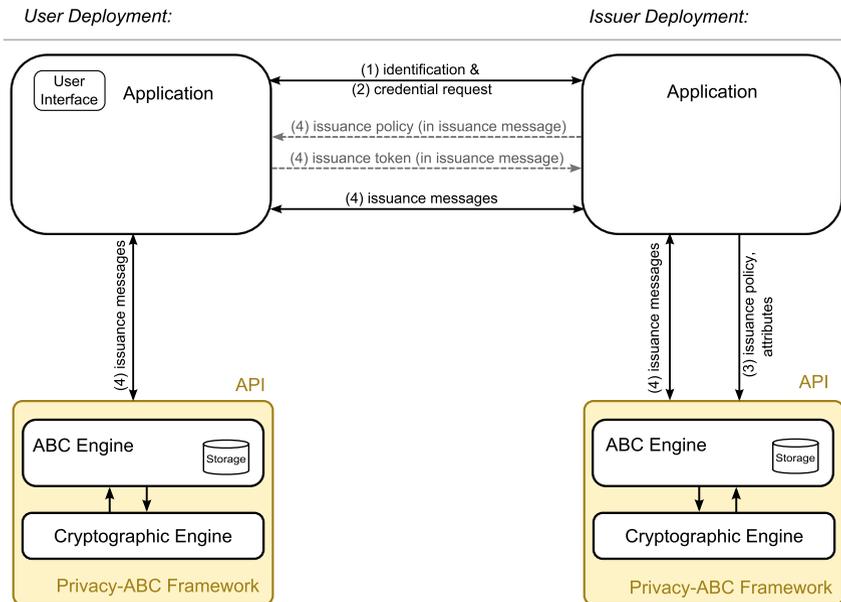


Fig. 2.6 Issuance of a Credential (Application Level)

2.4.3.1 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 invokes the `initIssuanceProtocol` method of the ABCE with the set of attributes that shall be certified in the new credential, and with an *IssuancePolicy* that merely contains the identifiers of the credential specification and the issuer parameters of the credential that is to be issued (Figure 2.6 – Step 3). This call initiates the cryptographic issuance protocol by invoking the Crypto Engine. The method returns an *IssuanceMessage* 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 the issuer pass the message to their Privacy-ABC system using the `issuanceProtocolStep` method (Figure 2.6 – Step 4). If the output of that method in turn contains an issuance message, that message is sent to the other party until the method on the user's side completed the credential generation. At the end of a successful issuance protocol, the user's Privacy-ABC system stores the new credential in the local credential store and returns the description of the credential to the user.

2.4.3.2 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 `initIssuanceProtocol` method on input of an issuance policy and the set of known user attributes that shall be certified in the new credential (Figure 2.6 – Step 3). The issuance policy must require the user to present at least one credential or one pseudonym, otherwise simple issuance is performed. The method returns an issuance message (containing the issuance policy) which must then be sent to the user.

The user in turn invokes the method `issuanceProtocolStep` with the received message. The user's Privacy-ABC system 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 in that the method's output contains an object of type *UiIssuanceArguments* for the user to perform an identity selection. The method then expects the user's response in form of a *UiIssuanceReturn* object. Finally, based on the user's choice, her Privacy-ABC system (with the help of the Crypto Engine) generates an *IssuanceToken*, which includes additional cryptographic data needed for the subsequent issuance protocol. The issuance to-

ken is wrapped in an issuance message, which the user then forwards to the issuer (Figure 2.6 – Step 4).

As for simple issuance, the issuer’s `issuanceProtocolStep` method is then called on input of the incoming issuance message from the user. The Privacy-ABC system 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). If the verification succeeds, the cryptographic issuance protocol is started, again with the help of the Crypto Engine. The method outputs an issuance message containing cryptographic data depending on the technology of the credential. The issuer then sends the returned issuance message to the user (Figure 2.6 – Step 4).

Whenever the user or the issuer receive an issuance message, they invoke their local `issuanceProtocolStep` method. The output is then 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 system stores the obtained credential and returns a description of that credential to the user.

Overall, the issuance-related APIs of the ABCE are summarized in Table 2.3.

Table 2.3 ABCE Interfaces for Credential Issuance

USER	
<code>issuanceProtocolStep</code>	input: <code>IssuanceMessage</code> output: <code>IssuanceMessage</code> , <code>CredentialDescription</code> , [<code>UiIssuanceArguments</code>]
<code>issuanceProtocolStep</code>	input: <code>UiIssuanceReturn</code> output: <code>IssuanceMessage</code>
ISSUER	
<code>initIssuanceProtocol</code>	input: <code>IssuancePolicy</code> , <code>List<Attribute></code> <i>issuerSpecifiedAttributes</i> output: <code>IssuanceMessage</code> , boolean <i>isLastMessage</i>
<code>issuanceProtocolStep</code>	input: <code>IssuanceMessage</code> output: <code>IssuanceMessage</code> , boolean <i>isLastMessage</i>
<code>extractIssuanceToken</code>	input: <code>IssuanceMessage</code> output: <code>IssuanceToken</code>

Table 2.4 ABCE Interfaces for Inspection and Revocation

INSPECTOR	
<code>inspect</code>	input: PresentationToken, URI <i>credentialAlias</i> , URI <i>attributeType</i> output: Attribute
<code>inspect</code>	input: IssuanceToken, URI <i>credentialAlias</i> , URI <i>attributeType</i> output: Attribute
REVOCATION AUTHORITY	
<code>revoke</code>	input: URI <i>revocationAuthorityId</i> , List<Attribute> <i>toRevoke</i> output: —

2.4.4 Inspection

As described in detail in Section 2.1.3.5, 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 her 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 inseparably 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, he sends the token (which he can retrieve, e.g., with the help of the `getPresentationToken` method described in Table 2.2) and the (non-cryptographic) evidence that the inspection grounds are fulfilled to the inspector. If the inspector determines by means of the evidence that these grounds are indeed fulfilled, he invokes the `inspect` method to decrypt the inspectable attributes in question (see Table 2.4).

2.4.5 Revocation

Our framework also supports revocation of credentials, thereby distinguishing whether a credentials may need to be revoked either globally (issuer-driven revocation) or for a specific context (verifier-driven revocation) (see Section 2.1 for details). To revoke a credential globally, the revocation authority calls the `revoke` method on input of the credential's revocation handle (see Table 2.4). For verifier-driven revocation, a conjunction of attributes can be revoked by calling the same method. In the latter case, all credentials that contain the combination of attribute values specified in the list will get revoked. 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.

All entities that deal with revocable credentials must ensure that their respective revocation information is up-to-date. This is handled transparently by the ABCE which – if required – will internally contact the corresponding revocation authority through the Revocation Proxy and obtain the necessary updates or information. For instance, issuers have to contact their revocation authority during issuance in order to obtain a fresh revocation handle. On the verifier side, such a process is needed to guarantee that the verifier uses the latest revocation information from the revocation authority in order to correctly detect revoked credentials.

Similarly, users have to keep the non-revocation evidence of their credentials up-to-date. The Privacy-ABC system 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.

2.5 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) [ASN08] or JavaScript Object Notation (JSON) [Cro06].

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 and features of Privacy-ABCs (see Section 2.1) by means of an example scenario, which scenario is described in the following section. For the sake of space and readability, the artifact examples for the scenario do not illustrate all the features of Privacy-ABCs. We refer the reader to [BCD⁺14] for the full specification. In the following sections, we explicitly distinguish between user attributes (as contained in a credential) and XML attributes (as defined by XML schema) whenever they could be confused.

2.5.1 Example Scenario

In this section, we describe an example scenario for illustrating the language framework artifacts that are introduced in the following sections.

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.

2.5.2 Credential Specification

A 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 2.7.

```

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>

```

Fig. 2.7 Credential specification of the identity card

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.

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).

2.5.3 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 2.8.

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.

```

1 <IssuerParameters>
2   <ParametersUID> urn:utopia:id:issuer </ParametersUID>
3   <AlgorithmID> urn:com:microsoft:uprove </AlgorithmID>
4   <SystemParametersUID> urn:utopia:id:system </SystemParametersUID>
5   <MaximalNumberOfAttributes> 4 </MaximalNumberOfAttributes>
6   <HashAlgorithm> xenc:sha256 </HashAlgorithm>
7   <CryptoParams> ... </CryptoParams>
8   <RevocationParametersUID> urn:utopia:id:ra </RevocationParametersUID>
9 </IssuerParameters>

```

```

1 <RevocationAuthorityParameters>
2   <ParametersUID> urn:utopia:id:ra </ParametersUID>
3   <RevocationMechanism> urn:privacy-abc:accumulators:cl </RevocationMechanism>
4   <RevocationInfoReference ReferenceType="url"> https:utopia.gov/id/revauth/revinfo
5   </RevocationInfoReference>
6   <NonRevocationEvidenceReference ReferenceType="url"> https:utopia.gov/id/revauth/nrevevidence
7   </NonRevocationEvidenceReference>
8   <CryptoParams> ... </CryptoParams>
9 </RevocationAuthorityParameters>

```

```

1 <SystemParameters>
2   <ParametersUID> urn:utopia:id:system </ParametersUID>
3   <CryptoParams> ... </CryptoParams>
4 </SystemParameters>

```

Fig. 2.8 Issuer, revocation authority, and system parameters

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.

2.5.4 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 2.7. 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 2.9.

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

```

1 <PresentationPolicy PolicyUID="libcard">
2   <Message>
3     <Nonce> bkQydHBQWDR4TUZzbXJKYUM= </Nonce>
4   </Message>
5   <Pseudonym Alias="nym" Scope="urn:library:issuance" Exclusive="true"/>
6   <Credential Alias="id" SameKeyBindingAs="nym">
7     <CredentialSpecAlternatives>
8       <CredentialSpecUID> urn:creds:id </CredentialSpecUID>
9     </CredentialSpecAlternatives>
10    <IssuerAlternatives>
11      <IssuerParametersUID> urn:utopia:id:issuer </IssuerParametersUID>
12    </IssuerAlternatives>
13    <DisclosedAttribute AttributeType="urn:creds:id:state"/>
14  </Credential>
15 </PresentationPolicy>

```

Fig. 2.9 Presentation policy for an identity card

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 2.9, 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 2.9, 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 2.9, 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 2.9, Lines 7-9 and 10–12). The credential element also indicates all attributes that must be disclosed by the user via **DisclosedAttribute** elements (Figure 2.9, 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 2.9, 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 2.1 , 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 2.10. It contains the presentation policy from Figure 2.9 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>

```

Fig. 2.10 Issuance policy for a library card. The presentation policy on Line 2 is depicted in Figure 2.9.

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 2.10, 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 2.10, 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 2.10, 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”.

2.5.5 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 2.11, Line 6). Similarly, the **DisclosedAttribute** elements (Figure 2.11, 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 2.11, 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.

```

1 <IssuanceToken>
2   <IssuanceTokenDescription>
3     <PresentationTokenDescription PolicyUID="libcard" >
4       <Message> ... </Message>
5       <Pseudonym Alias="nym" Scope="urn:library:issuance" Exclusive="true" />
6       <PseudonymValue> MER2VXISHI=</PseudonymValue>
7     </Pseudonym>
8     <Credential Alias="id" SameKeyBindingAs="nym" >
9       ...
10      <DisclosedAttribute AttributeType="urn:creds:id:state" >
11        <AttributeValue> Nirvana </AttributeValue>
12      </DisclosedAttribute>
13    </Credential>
14  </PresentationTokenDescription>
15  <CredentialTemplate SameKeyBindingAs="id" > ... </CredentialTemplate>
16 </IssuanceTokenDescription>
17 <CryptoEvidence> ... </CryptoEvidence>
18 </IssuanceToken>

```

Fig. 2.11 Issuance token for obtaining the library card

2.5.6 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 2.12.

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 2.12, 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 2.12, 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 [BCD⁺14] 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.

```

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:arbiter </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>

```

Fig. 2.12 Presentation policy for borrowing books

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 **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 [BCD⁺14]. 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 2.12, 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.

2.5.7 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 remainder 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 2.13) 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 parsing of the policy to figure out what exactly is being revealed. It

```

1 <UiPresentationArguments>
2   <data>
3     <credentialSpecification id="urn:utopia:lib">...</credentialSpecification>
4     <credentialSpecification id="urn:creds:id">...</credentialSpecification>
5     <issuer id="urn:utopia:lib:issuer">...</issuer>
6     <issuer id="urn:utopia:id:issuer">...</issuer>
7     <inspector id="urn:lib:arbitrator">...</inspector>
8     <revocationAuthority id="urn:utopia:id:ra">...</revocationAuthority>
9     <credentialDescription id="urn:utopia:lib:74bdfb3-6886-43ac-83f8-ca3b72ad050d">...</
10    credentialDescription>
11    <credentialDescription id="urn:creds:id:14f22b9d-06e0-4110-a8d9-b1a922462cd1">...</
12    credentialDescription>
13  </data>
14  <tokenCandidatePerPolicy policyId="0">
15    <policy>...</policy>
16    <tokenCandidate candidateId="0">
17      <tokenDescription>...</tokenDescription>
18      <credential ref="urn:utopia:lib:74bdfb3-6886-43ac-83f8-ca3b72ad050d" />
19      <credential ref="urn:creds:id:14f22b9d-06e0-4110-a8d9-b1a922462cd1" />
20      <revealedFact>
21        <description lang="EN">You prove that urn:creds:id:bdate from credential urn:creds:id:
22        is greater than 1988-04-01 (26 years ago).</description>
23      </revealedFact>
24      <revealedFact>
25        <description lang="EN">You prove that 'Full Name' from credential 'Library Card'
26        is not revoked by the verifier urn:lib:blacklist.</description>
27      </revealedFact>
28      <revealedFact>
29        <description lang="EN">You prove that urn:creds:id is not revoked by urn:utopia:id:ra.</description>
30      </revealedFact>
31      <inspectableAttribute>
32        <credential ref="urn:utopia:lib:74bdfb3-6886-43ac-83f8-ca3b72ad050d" />
33        <attributeType>urn:utopia:lib:name</attributeType>
34        <inspectionGrounds>Late return or damage.</inspectionGrounds>
35        <inspectorAlternative ref="urn:lib:arbitrator" />
36      </inspectableAttribute>
37    </tokenCandidate>
38  </tokenCandidatePerPolicy>
39  <tokenCandidatePerPolicy policyId="1">...</tokenCandidatePerPolicy>
40 </UiPresentationArguments>

```

Fig. 2.13 Message sent to the User Interface for Presentation

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 2.14). 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 2.14), 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

```
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>
```

Fig. 2.14 Message sent to the User Interface for Issuance

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 2.15) 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 2.16).

```
1 <UiPresentationReturn>
2 <chosenPolicy>0</chosenPolicy>
3 <chosenPresentationToken>0</chosenPresentationToken>
4 <chosenInspectors>urn:lib:arbitrator</chosenInspectors>
5 </UiPresentationReturn>
```

Fig. 2.15 Response from the User Interface for Presentation

The **UiIssuanceArguments** object (Figure 2.14) 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 2.16) 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

```

1 <UiIssuanceReturn>
2 <chosenIssuanceToken>0</chosenIssuanceToken>
3 <chosenPseudonymList>0</chosenPseudonymList>
4 <metadataToChange>
5 <entry>
6 <key>nym:urn:library:issuance:965999d1-25e9-49e5-8db6-ad8ae9705807</key>
7 <value>I used this to obtain my library card.</value>
8 </entry>
9 </metadataToChange>
10 </UiIssuanceReturn>

```

Fig. 2.16 Response from the User Interface for Issuance

chosen. In this example, the user has also chosen to associate new metadata to the pseudonym (Lines 4–9).

2.6 Applicability to Existing Identity Infrastructures

Many identity protocols and frameworks are in use today, and new ones are being developed by the industry, each addressing specific use cases and deployment environments. Privacy concerns exist in many scenarios targeted by these systems, and therefore it is useful to understand how they could benefit from Privacy-ABC technologies to improve their security, privacy, and scalability.

We consider the following popular systems: WS-*, SAML, OpenID, OAuth, and X.509.¹ A short description of each system is given to facilitate the discussion, but is by no means complete; the reader is referred to the appropriate documentation to learn more about a particular system. Moreover, we mostly describe how integration can be done, rather than discussing why as this is highly application-specific.

The last section describes the common challenges of these federated systems, and how Privacy-ABC technologies can help to alleviate them.

2.6.1 WS-*

The set of WS-* specifications define various protocols for web services and applications. Many of these relate to security, and in particular, to authentication and attribute-based access (such as WS-Trust [WST09], WS-Federation [WSF09], and WS-SecurityPolicy [WSS07]). These specifications can be combined to implement various systems with different characteristics.

¹ Other popular frameworks, such as Facebook Login [Fac], OpenID Connect [Ope], and Fido Alliance [Fid] are similar or built on top of the schemes presented here, and will therefore be omitted in the discussion.

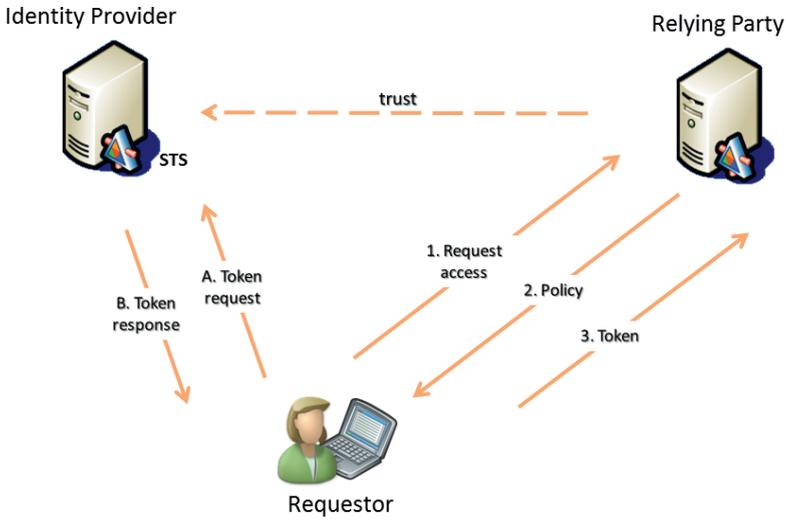


Fig. 2.17 WS-Trust protocol flow

The WS-Trust specification is the main building block that defines how *security tokens* can be obtained and presented by users. The specification does not make any assumption on the type of tokens exchanged, and provides several extensibility points and protocol flow patterns suitable for Privacy-ABC technologies.

In WS-Trust, a requestor (user) requests a security token from the Identity Providers Security Token Service (the issuer) encoding various certified claims (attributes), and presents it (either immediately or at a later time) to a Relying Party (the verifier); see Figure 2.17.

Integrating Privacy-ABC technologies in WS-Trust is straightforward due to the extensible nature of the WS-* framework. The issuance protocol is initiated by the requestor by sending, as usual, a `RequestForSecurityToken` message to the STS. The requestor and the STS then exchange as many `RequestForSecurityTokenResponse` messages as needed by the ABC issuance protocol (using the challenge-response pattern defined in Section 8 of [WS-12]). The STS concludes the protocol by sending a `RequestForSecurityTokenResponseCollection` message. Typically, this final message contains a collection of requested security tokens. Due to the nature of the Privacy-ABC technologies, the STS does not send the security tokens per se, but the requestor is able to compute its credential(s) using the exchanged cryptographic data. See Figure 2.18.

The issuance messages are tied together using a unique context, but otherwise do not specify the content and formatting of their contents. It is therefore possible to directly use the protocol artefacts defined in Section 2.5.

Presenting an ABC to a Relying Party is also straightforward. The exact mechanism to use depends on the application environment. For example, in a federated

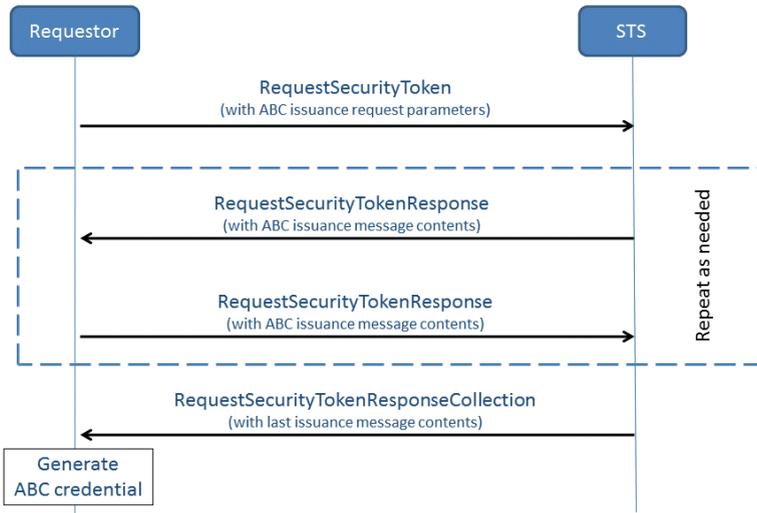


Fig. 2.18 WS-Trust issuance protocol

architecture using WS-Federation, the presentation token could be included in a `RequestForSecurityTokenResponse` message part of a `wresult` HTTP parameter. Given the support of extensible policy (using, e.g., WS-SecurityPolicy), the ABC verifier policy could be expressed by the Relying Party and obtained by the client; e.g., it could be embedded in a services federation metadata (see Section 3 of [WSF09]). Privacy-ABC technology integration into WS-Trust has been successfully demonstrated; see, e.g., [UPW11].

2.6.2 SAML

The Security Assertion Markup Language (SAML) is a popular set of specifications for exchanging certified assertions in federated environments. Different profiles exist addressing various use cases, but the core specification [SAM05] defines the main elements: the SAML assertion (a XML token type that can encode arbitrary attributes), and the SAML protocols for federated exchanges.

Typically, a User Agent (a.k.a. requester or client) requests access to a resource from a Relying Party (a.k.a. Service Provider) which in turn requests a SAML assertion from a trusted Identity Provider (a.k.a. SAML Authority). The User Agent is redirected to the Identity Provider to retrieve the SAML assertion (after authenticating to the Identity Provider in an unspecified manner) before passing it back to the Relying Party. Figure 2.19 illustrates the protocol flow.

Contrary to WS-*, the SAML protocols only permit the use of the SAML assertion token type. Therefore, one needs to profile the SAML assertion in order

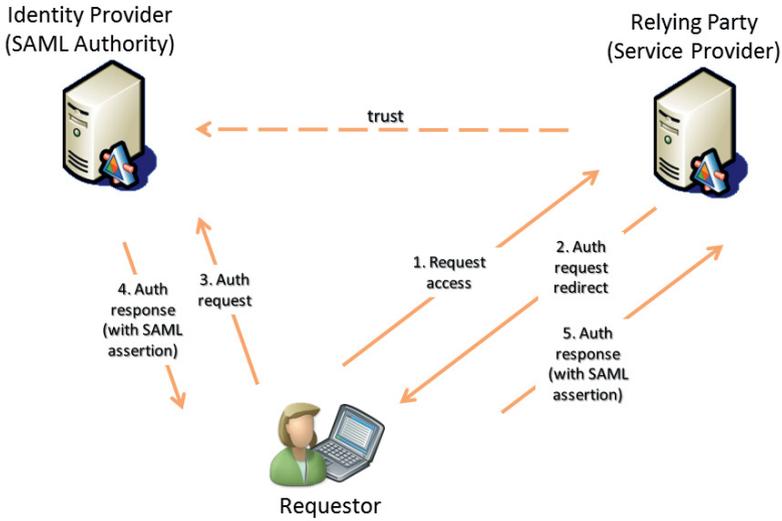


Fig. 2.19 SAML protocol flow

to use the Privacy-ABC technologies with the SAML protocols. The SAML assertion schema defines an optional `ds:Signature` element used by the Identity Provider to certify the contents of the assertion. If used, it must be a valid XML Signature [Bar02]. This means that XML Signature must also be profiled to support ABC issuer signatures.² The alternative would be to protect the SAML assertion using a custom external signature element. ABC-based SAML assertions could be used in the SAML protocols in various ways. One example would be for the client to create a modified SAML assertion using a Privacy-ABC in response to a Relying Partys authentication request rather than fetching it in real-time from the Identity Provider (replacing steps 3 and 4 in Figure 2.19). The assertion would contain the disclosed attributes, and encode the presentation tokens cryptographic data in the SAML signature. Essentially, the SAML assertion would be an alternative token type to the ABC presentation token. Additionally, the Identity Provider could issue an on-demand Privacy-ABC using the SAML protocol; this might require multiple roundtrips to accommodate the potentially interactive issuance protocol. Then the SAML assertion presented to the Relying Party would need to be created as explained above.

² This could be achieved by applying the appropriate XML transforms on the assertions contents before interpreting them as input to the ABC protocols.

2.6.3 OpenID

OpenID is a federated protocol allowing users to present an identifier³ to Relying Parties by first authenticating to an OpenID Provider. The current specification, OpenID 2.0 [Ope07], specifies the protocol. Assuming that the user has an existing OpenID identifier registered with an OpenID Provider, we illustrate the steps in Figure 2.20.

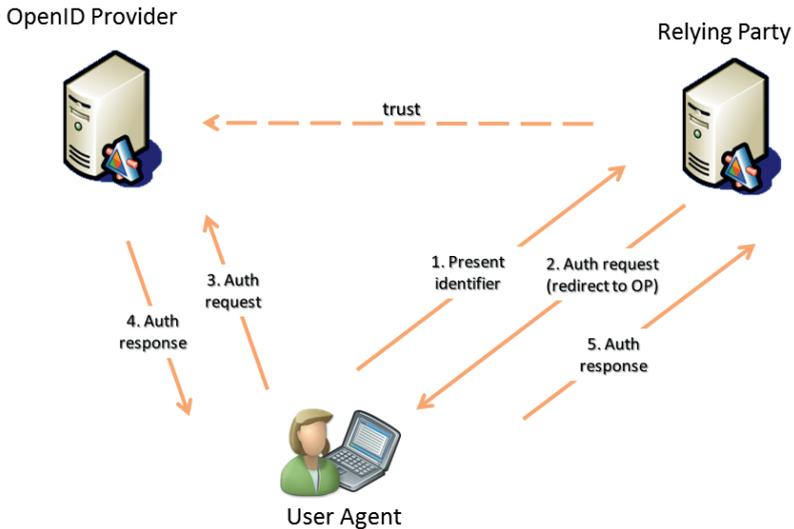


Fig. 2.20 OpenID protocol flow

1. To login to a Relying Party, the user presents her (unverified) OpenID identifier.
2. The Relying Party parses the identifier to discover the Users OpenID Provider and redirects the User Agent to it.
3. The user authenticates to the OpenID Provider; how this is achieved is out-of-scope of the OpenID specification (popular existing web deployments use usernames and passwords).
4. Upon successful authentication, the OpenID Provider redirects the User Agent to the Relying Party with a signed successful authentication message.
5. The Relying Party validates the authentication message using either a shared secret with the OpenID Provider or alternatively, by contacting the OpenID Provider directly.

OpenID follows a standard federated single sign-on model and therefore inherits the security and privacy problems of such systems. The OpenID specification de-

³ The specification describe this as a URL or XRI (eXtensible Resource Identifier), but extensions used by popular deployments use email addresses.

scribes in Section 15 some countermeasures against common concerns, but nonetheless, the systems remains vulnerable to active attackers, especially to attacks originating from protocol participants (see, e.g., [Bra] for a summary of the issues).

Privacy-ABC technologies could be used to increase both the security and privacy of the protocol, and reduce the amount of trust needed on OpenID Providers. For example, certified or scope-exclusive pseudonyms derived from an ABC issued by an OpenID Provider could be used as local Relying Party identifiers, therefore providing unlinkability between the Users spheres of activities at different Relying Parties (using the Relying Parties URL as a scope string). The cryptographic data in the corresponding ABC presentation token would need to be encoded in extension parameters defined in an ABC profile. A similar integration has been demonstrated in the PseudoID prototype [DW10], using Chaums blind signatures [Cha83].

OpenID may also be used in attribute-based access scenarios. The OpenID Attribute Exchange [HBH07] extension describes how Relying Party can request attributes of any type from the OpenID Provider by adding fetch parameters in the OpenID authentication message, and how an OpenID Provider can return the requested attributes in the response. OpenID Connect [Ope] is a new scheme built on top of OAuth (see following section) that also addresses attribute exchange.

To generate an ABC-based response, the User Agent would create the OpenID response on behalf of the OpenID Provider using the contents of a presentation token, properly encoding the disclosed attributes using the OpenID Attribute Exchange formatting and by encoding the cryptographic evidence in custom attributes.

2.6.4 OAuth

OAuth is an authorization protocol that enables applications and devices to access HTTP⁴ services on behalf of users using delegated tokens rather than the users main credentials. The current specification, OAuth 2.0 [Har12], is being developed by the IETF OAuth working group.⁵ OAuth specifies four roles. Quoting from the spec:

- resource owner:** an entity capable of granting access to a protected resource (e.g. end-user).
- resource server:** the server hosting the protected resources, capable of accepting responding to resource requests using access tokens.
- client:** an application making protected resource requests on behalf of the owner and with its authorization.
- authorization server:** the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

An example scenario is as follows: an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing

⁴ Using a transport protocol other than HTTP is undefined by the specification.

⁵ OAuth 2.0 evolved from the OAuth WRAP [HTEG10] profile which has been deprecated.

service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server) which issues the service delegation-specific credentials (access token).

A typical OAuth interaction is illustrated in Figure 2.21:

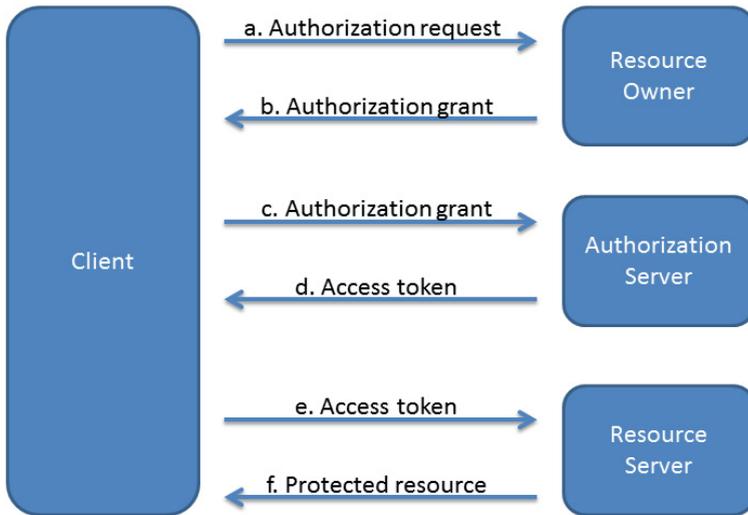


Fig. 2.21 OAuth 2.0 protocol flow

- a. The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary.
- b. The client receives an authorization grant which is a credential representing the resource owner’s authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.
- c. The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
- d. The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token.
- e. The client requests the protected resource from the resource server and authenticates by presenting the access token.
- f. The resource server validates the access token, and if valid, serves the request.

As we can see, two types of credentials are used in the protocol flow: the authorization grant and the access token. A Privacy-ABC could be used for either one,

as we will describe in the following sections⁶. The OAuth protocol flow does not allow presenting a dynamic policy to the client; if this functionality is needed, the policy would need to be obtained and processed at the application layer; otherwise, the application may use an implicit policy that drives the clients behaviour.

2.6.4.1 Authorization grant

The first step in the OAuth flow is for the client to request authorization from the resource owner and getting back an authorization grant. The OAuth specification defines four grant types (authorization code, implicit, resource owner password credentials, and client credentials) and provides an extension mechanism for defining new ones.

Although one could use the authorization code or the client credential grant types, the extension mechanism is better-suited to integrate ABC-based grants. How the Privacy-ABC is obtained by the client is out-of-scope of the OAuth flow. To present the Privacy-ABC to the authorization server, one could define a profile similar to the SAML assertion one [MCM14]. For example, the client could send the following access token request to the authorization server:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
grant_type=http://abc4trust.eu/oauth&abctoken=PEFzc2VG1v...
```

where the `abctoken` parameter would contain an encoding of a presentation token (e.g., using a base64 encoding of the XML representation). As mentioned above, the policy driving the clients presentation behaviour would be dealt with at the application level (and might be fixed for an application).

2.6.4.2 Access token

An access token is issued by the authorization server to the client and later presented to the resource server. The format and contents of the access token is not defined in the OAuth specification, and therefore one could define a way to use a Privacy-ABC to create an access token. This can be done by defining a new access token type (as explained in Section 8.1 of [Har12]), or by encoding the presentation token content into an existing extensible token type, such as the JSON Web Token [JWT].⁷

Since access tokens are typically long-lived, the issuance of the Privacy-ABC can be done out-of-band of the OAuth protocol. It can also be done directly by

⁶ The OAuth specification does not describe how the resource owner authenticates the client before issuing the authorization grant. Conceptually, this could also be done using an ABC.

⁷ The JSON Web Token format contains a set of attribute name and value pairs and corresponding metadata (including a digital signature identified by an algorithm identifier). This is supported by ABC technologies, but does not allow the representation of the most advanced features. JWT extensions, such as the Proof-Of-Possession Semantics for JSON Web Tokens [JBT], might help to enable all the ABC features.

the authorization server by embedding the issuance protocol messages in multiple access token request-response runs (in which case the returned access tokens would be the opaque issuance messages). When this process concludes, the client would be able to create a valid ABC-based access token.

To present the ABC access token, client computes a valid presentation token using an application-specific resource policy (obtained out-of-band or implicitly defined), encodes it in the right access token format, and includes it in the OAuth protected resources access request.

2.6.5 X.509 PKI

Most of the schemes presented in this section require online interactions with an Issuer to present attributes to a Relying Party. This provides flexibility about what can be disclosed to the Relying Party, but impacts the privacy vis-à-vis the Issuer (which typically learns where the attributes are presented). A Public Key Infrastructure (PKI) uses a different approach: PKI certificates encoding arbitrary attributes and issued to users are typically long-lived. The decoupling of the issuance and presentation protocols provides some privacy benefits to the user, but removes the minimal disclosure aspect. Indeed, a Verifier will learn everything that is encoded in a certificate even if a subset of the information would have been sufficient to make its access decision. The integration of Privacy-ABC technology is therefore desirable to provide these privacy benefits while offering the same security level as in PKI.

X.509 [CSF⁺08] is a popular PKI standard⁸ that defines two types of credentials: public key and attribute certificates. A public key certificate contains a user public key associated to a secret private key, and other metadata (serial number, a validity period, a subject name, etc.) The certificate is signed by a Certificate Authority. An attribute certificate, also signed by the CA, is tied to a public-key certificate and can contain arbitrary attributes. Both types of certificates can also contain arbitrary extensions.

The X.509 protocol flow is as follows. The client starts by generating a key pair, and sends a certificate request that includes the generated public key to the Certificate Authority. The Certificate Authority creates, signs and returns the X.509 certificate to the client which stores it along with the associated private key. To authenticate to a Relying Party, the client later uses the certificates private key to sign a Relying Party-specified challenge (either a random number or an application-specific message). The Relying Party verifies the signature and validates the certificate. This involves verifying the certificates Certificate Authority signature, making sure that the Certificate Authority is a trusted issuer (is or is linked to a trusted root), and making sure that the certificate has not expired and is not revoked. Checking for non-revocation can be done by either checking that the certificates serial number

⁸ Other PKI systems exist, such as PGP [CDF⁺]. We will not consider them in this document, but ABC integration would look similar.

does not appear on a Certificate Revocation List (CRL), or by querying an Online Certificate Status Protocol (OCSP) responder.⁹ See Figure 2.22.

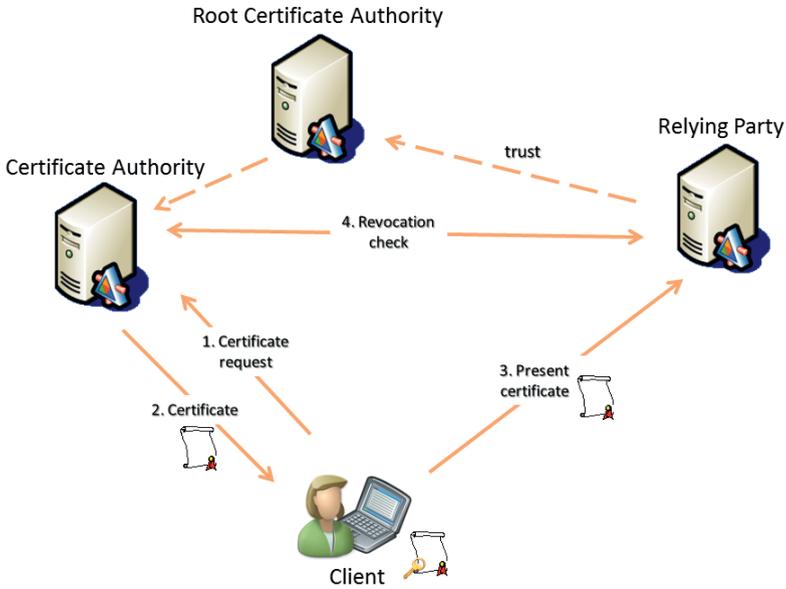


Fig. 2.22 X.509 protocol flow

Integrating Privacy-ABCs with X.509 certificates is possible and provides two immediate benefits:

- Long-lived certificates support minimal disclosure (only the relevant properties of encoded attributes are disclosed to the Relying Party rather than the full set of attributes), and
- The users public key and the Certificate Authority signatures on the certificates are unlinkable (the Certificate Authority and the Relying Parties cannot track and trace the usage of the certificate based solely on these cryptographic values).

Two integration approaches are considered next. The first one consists of encoding the ABC artefacts contents in X.509 artefacts using ABC-specific algorithm identifiers and extensions (i.e., the client would generate an X.509 certificate encoding the Privacy-ABCs contents at the end of the issuance protocol). Since the presentation protocol of an X.509 certificate is not specified, the presentation token artefact could be used almost as is, but including the modified X.509 certificate.

⁹ The mechanism and endpoint to be used are specified by the CA and encoded into the certificate.

The second and preferred¹⁰ approach would be to transform an existing X.509 certificate into a Privacy-ABC that can be presented to various Relying Parties. The following example illustrates the concept: The protocol flow would be as follows:

1. The client visits the ABC issuer and presents her X.509 certificate.
2. After validating the certificate and its ownership by the User, the ABC Issuer issues a Privacy-ABC encoding the certificates information into attributes:
 - a. The certificates expiration date is encoded in an attribute.
 - b. The certificates serial number is encoded as the revocation handle.
 - c. The revocation information (e.g., the CRL endpoint)¹¹ is encoded in an attribute.
 - d. The Certificate Authority identifier is encoded in an attribute.
 - e. The other certificate fields might also be encoded in the Privacy-ABC if they need to be presented to Relying Parties.
3. The client later presents the ABC to the Verifier, disclosing the following information:
 - a. Disclose the Certificate Authority identifier¹² and revocation information attributes.
 - b. Prove that the underlying certificate is not expired by proving that the undisclosed expiration date is not before the current time.
 - c. Prove that the serial number does not appear on the current CRL (this can be achieved using repetitive negation proofs on the CRL elements).¹³
4. The Verifier would perform these validation steps (on top of the normal ABC validation):
 - a. Verify that the Certificate Authority is from a trusted set of issuers.
 - b. Retrieve the current CRL (using the disclosed revocation information) and verify the non-revocation proof.
 - c. Verify the non-expiration proof.

After these steps, the Verifier is convinced that the user possesses a valid (i.e., non-expired, non-revoked) X.509 certificate from a trusted Certificate Authority.

¹⁰ We claim that this approach is preferred because of the broad existing code base implementing X.509. It would be easier to develop an conversion module on top of existing X.509 components.

¹¹ This example uses a CRL as the revocation mechanism. Using OCSP would also be possible by having the client prove to the OCSP responder directly that the ABC is not revoked, and presenting a freshly issued receipt to the Relying Party.

¹² Alternatively, the client could prove that the CA is from a trusted set specified by the Verifier.

¹³ Alternatively, an ABC Revocation Authority could create an accumulator for the revoked values.

2.6.6 Integration Summary

The systems presented above follow a similar federated pattern of a Relying Party requesting, through the user, login or attribute information from a trusted Identity Provider. In PKI and OAuth the certified information (certificate and access token, respectively) are typically obtained in advance and reused over time, while in the other systems, the information is retrieved on-demand from the Identity Provider.

These architectures have some security, privacy, and scalability challenges that might be problematic in some scenarios:

- The Identity Provider can often access the Relying Party using a users identity without the users knowledge. This is trivial in systems where the Identity Provider creates the pseudonym (like in SAML, OpenID, OAuth, WS-Federation). In systems where a user secret is employed (like in PKI, or in some WS-Trust profiles), this is more complicated but still could be possible.¹⁴ Moreover, Identity Providers can also selectively deny access to users by refusing to issue security tokens (discriminating on the requesting user or requested service).
- For authentication depending on knowledge of a user secret (e.g., username/password), phishing attacks on the credential provided to the Identity Provider result in malicious access to all Relying Parties that accept that identity.
- Strong authentication to the Identity Provider is often supported (including multi-factor asymmetric-based authentication), but the resulting security tokens (e.g., SAML assertion, OAuth access token, OpenID authentication response) are typically weaker software-only bearer token which can be intercepted and replayed by adversaries.
- The Identity Provider typically learns which Relying Party the user is trying to access. For on-demand security token issuance, this information is often provided to the Identity Provider in order to protect the security token (e.g., to encrypt it for the Relying Party) or to redirect the user to the right location. When security tokens are long-lived (like in PKI), this information is still available if the Identity Providers and Relying Parties compare notes (since signatures on security tokens generated using conventional cryptography are traceable).
- Central Identity Providers in on-demand federated systems limit the scalability of the systems because if they are offline, users will not be able to access any Relying Parties. This makes them interesting targets for denial of service attacks.

Privacy-ABC technologies help alleviate these issues by increasing the security, privacy, and scalability of these systems. Indeed:

- Since Privacy-ABCs are by default untraceable, even when obtained on-demand, Identity Providers are not able to track and trace the usage of the users information.

¹⁴ As an example, in PKI, a Certificate Authority would not be able to re-issue a valid certificate containing the users public key, but could re-issue one with a matching serial number and subject and key identifiers often used for user authentication.

- Since Privacy-ABCs can be obtained in advance and stored by the user while still being able to disclose the minimal amount of information needed for a particular transaction, the real-time burden of the issuer is diminished, improving scalability.
- Since Privacy-ABCs are based on asymmetric cryptography, presenting login pseudonyms and certified attributes involve using a private key unknown to the Issuer, meaning that the Identity Provider (or another adversary) is unable to hijack the users identity at a particular Relying Party.

Privacy-ABC technologies offer a wide range of features; not all of them trivially compatible with the systems presented in this section. The important point is that Privacy-ABC technologies offer a superset of the functionality and of the security/privacy/scalability characteristics of these systems. Protocol designers and architects can therefore pick and choose which features and characteristics they would like to use to improve existing systems or their future revisions.

It is also important to note that Privacy-ABC technologies can be used in conjunction with these frameworks, since many real-life applications wont have the luxury to modify the existing standards and development libraries. Most of the privacy concerns occur in cross-domain data sharing, i.e., when information travels from one domain to another. Therefore, an ABC proxy can be used as a privacy filter between domains using well-known federated token transformer pattern (such as the WS-Trust STS). This is useful to avoid modifying legacy applications and infrastructure, and still benefit from the security and privacy properties of Privacy-ABC technologies.

2.7 Trust Relationships in the Ecosystem of Privacy-ABCs

Several incidents in the past have demonstrated the existence of possible harm that can arise from misuse of people’s personal information such as blackmailing, impersonation, and so on. Giving credible and provable reassurances to people is required to build trust and make people feel secure to use the electronic services offered by companies or governments on-line. Indeed the use of Privacy-ABCs can help mitigate many serious threats to user’s privacy. However, some risks still remain, which are not addressed by Privacy-ABCs, requiring some degree of trust between the involved entities. In this section, we focus on identifying the trust relationships between the involved entities in the ecosystem of Privacy-ABCs and provide a concrete answer to “*who needs to trust whom on what?*”.

2.7.1 The Meaning of Trust

what do we mean by “trust”? A wide variety of definitions of trust exist in the bibliography [Har04][O’H04]. A comprehensive study of the concept has been presented

in the work by McKnight and Chervany [MC96], where the authors provide a classification system for different aspects of trust. In their work, they define trust intention as “*the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible.*” [MC96]

Their definition embodies (a) the prospect of negative consequences in case the trusted party does not behave as expected, (b) the dependence on the trusted party, (c) the feeling of security, and the (d) situation-specific nature of trust. So, trust intention shows the willingness to trust a given party in a given context, and implies that the trusting entity has made a decision about the various risks of allowing this trust.

2.7.2 Related Work

Some work already exists in trust relationships in identity management systems. For example, Jøsang et al. [JP04] analyse some of the trust requirements in several existing identity management models. They consider the federated identity management model, as well as the isolated or the centralized identity management model and they focus on the trust requirements of the users into the service and identity providers, but also between the identity providers and service providers.

Delessy et al. [DFLP07] define the Circle of Trust pattern, which represents a federation of service providers that share trust relationships. The focus of their work however lies more on the architectural and behavioural aspects, rather than on the trust requirements which must be met to establish a relationship between two entities.

Later, Kylau et al. [KTMM09] concentrated explicitly on the federated identity management model and identify possible trust patterns and the associated trust requirements based on a risk analysis. The authors extend their scenarios by considering also scenarios with multiple federations. Nevertheless, their work does not match the ecosystem of Privacy-ABCs.

It seems that there is no work that discusses systematically the trust relationships in identity management systems that incorporate Privacy-ABCs. However, some steps have been done towards systematic threat analysis in such schemes, by the establishments of a quantitative threat modelling methodology that can be used to identify privacy-related risks on Privacy-ABC systems [LSK12].

2.7.3 Trust Relationships

To provide a comprehensible overview of the trust relationships, we describe the trust requirements from each entity’s perspective. Therefore, whoever likes to realise one of the roles in the ecosystem of Privacy-ABCs could easily refer to that entity

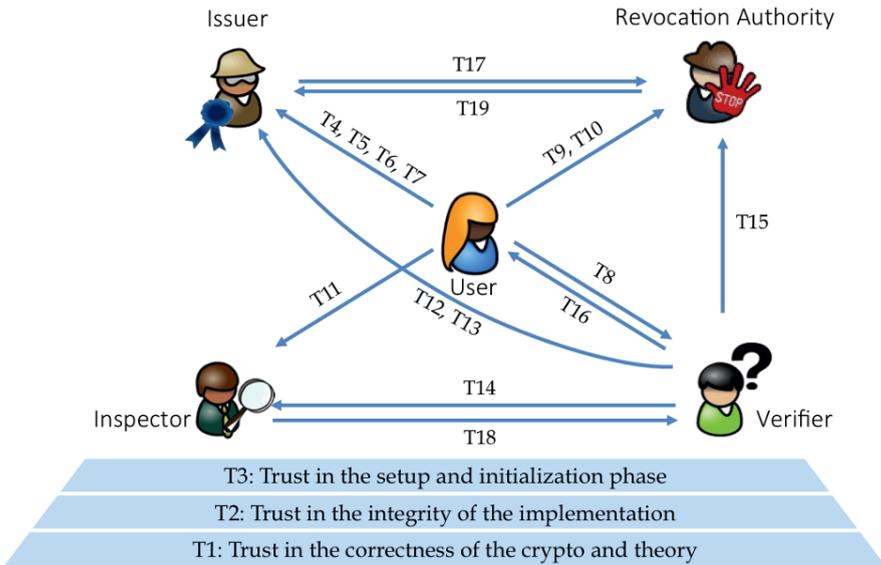


Fig. 2.23 Visualization of the trust relationships

and learn about the necessary trust relationships that need to be established. Figure 2.23 depicts an overview of the identified trust relationships between the involved parties, which we will describe in the next sections. On the bottom of Figure 2.23, the general trust requirements by all the parties are demonstrated.

2.7.3.1 Assumptions

Before delving into the trust relationships, it is important to elaborate on the assumptions that are required for Privacy-ABCs to work. Privacy-ABCs are not effective in case of tracking and profiling methods that work based on network level identifiers such as IP addresses or the ones in the lower levels. Therefore, in order to benefit from the full set of features offered by Privacy-ABCs, the underlying infrastructure must be privacy-friendly as well. If it is ensured that no additional information is being collected by the service providers, users can employ Privacy-ABCs without any concern. Otherwise, the recommendation for the users would be to employ network anonymizer tools to cope with this issue.

Another important assumption concerns the verifiers' enthusiasm for collecting data. Theoretically, greedy verifiers have the chance to demand for any kind of information they are interested in and avoid offering the service if the user is not willing to disclose these information. Therefore, the assumption is that the verifiers reduce the amount of requested information to the minimum level possible either due to regulations or any other motivation such as not having to invest in technology to protect the data.

2.7.3.2 Trust by all the parties

Independent from their roles, all the involved parties need to consider a set of fundamental trust assumptions that relates to design, implementation and setup of the underlying technologies. It is worth noting that these kind of trust relationships exist for any kind of technologies. The most fundamental trust assumption by all the involved parties concerns the theory behind the actual technologies utilized underneath. Everybody needs to accept that in case of a proper implementation and deployment, the cryptographic protocols will offer the functionalities and the features that they claim. However, this trust relationship can be relaxed by making the security proofs publicly available so that different expert communities can verify them and vouch for their correctness.

T1 *All the involved parties need to put trust in the correctness of the underlying cryptographic protocols.*

Even a protocol that is formally proven to be privacy preserving does not operate appropriately when the implementation is flawed. Consequently, the realization of the corresponding cryptographic protocol and the related components must be trustworthy. For example, the Users need to trust the implementation of the so-called UserAgent and the smart card application meaning that they must rely on the assertion that the provided hardware and software components do not misbehave in any way and under any circumstances, which might jeopardise the User's privacy. It is worth noting that there are mechanisms such as *formal verification* and *code inspection* which can boost the users' trust in the implementations.

T2 *All the involved parties need to put trust in the trustworthiness of the implemented platform and the integrity of the defined operations on each party.*

A correct implementation of privacy preserving technologies cannot be trustworthy when the initialization phase has been compromised. For example, some cryptographic parameters need to be generated in a certain way in order to guaranty the privacy preserving features of a given technology. A diversion in the initialization process might introduce vulnerabilities to the future operation of the users. Nevertheless, it is possible to provide some information to the public so that the experts can check whether the initialization is done properly.

T3 *All the involved parties need to put trust in the trustworthiness of the system setup and the initialization process.*

2.7.3.3 Users' Perspective

In typical scenarios, verifiers grant access to some services based on the credentials that the users hold. A malicious issuer can trouble a user and cause denial of service by not providing credible credentials in time or deliberately embedding invalid

information in the credentials. For example, in case of a discount voucher scenario, the issuer of the vouchers can block some specific group of users with fake technical failures of the issuance service until the offer is not valid anymore.

T4 *The users need to put trust in the issuers delivering accurate and correct credentials in a timely manner.*

When designing a credential, the issuer must take care that the structure of the attributes and the credential will not impair the principle of minimal disclosure. For example, embracing name and birth date in another attribute such as registration id is not an appropriate decision since presenting the latter to any verifier results in undesirable disclosure of data. In this regard, making the credential specifications public enables the independent auditors to review them and therefore reduce the concerns of the users who might not have the knowledge to evaluate the credentials on their own.

T5 *The users need to trust that the issuers design the credentials in an appropriate manner, so that the credential content does not introduce any privacy risk itself.*

Similar to any other electronic certification system, dishonest issuers have the possibility to block a user from accessing a service without any legitimate reason by revoking her credentials. Therefore the users have to trust that the issuer has no interest in disrupting users activities and will not take any action in this regard as long as the terms of agreement are respected.

T6 *The users need to trust that the issuers do not take any action to block the use of credentials as long as the user complies with the agreements.*

It is conceivable that a user loses control over her credentials and therefore contacts the issuer requesting for revocation of those credentials. If the issuer delays processing the user's request the lost or stolen credentials can be misused to harm the owner.

T7 *The users need to trust that the issuers will promptly react and inform the revocation authorities when the users claim losing control over their credentials.*

One of the possible authentication levels using Privacy-ABCs is based on a so-called *scope-exclusive pseudonym* where the verifier is able to impact the generation of pseudonyms by the users and limit the number of partial identities that a user can obtain in a specific context. For example, in case of an on-line course evaluation system, the students should not be able to appear under different identities and submit multiple feedbacks even though they are accessing the system pseudonymously. In this case, the verifier imposes a specific *scope* to the pseudonym generation process so that every time a user tries to access the system, it has no choice other than

showing up with the same pseudonym as the previous time in this context. In this situation, a dishonest verifier can try to unveil the identity of a user in a pseudonymous context or correlate activities by imposing the “same” scope identifier in generation of pseudonyms in another context where the users are known to the system. However, similar to some other trust relationships, independent auditors could attest these policies when they are publicly available.

T8 *The users need to trust that the verifiers do not misbehave in defining policies in order to cross-link different domains of activities.*

If a revocation process exists in the deployment model, the user needs to trust the correct and reliable performance of the revocation authority. Delivering illegitimate information or hindrance to provide genuine data can disrupt granting user access to her desired services.

T9 *The users need to trust that the revocation authorities perform honestly and do not take any step towards blocking a user without legitimate grounds.*

Depending on the revocation mechanism setting, the user might need to show up with her identifier to the revocation authority in order to obtain the non-revocation evidence of her credentials for an upcoming transaction. If the revocation authority and the verifier collude, they might try to correlate the access timestamps and therefore discover the identity of the user who requested a service. A possible way to reduce this risk would be to regularly update the non-revocation evidence independent of their use of credentials.

T10 *The users need to trust that the revocation authorities do not take any step towards collusion with the verifiers in order to profile the users.*

Embedding encrypted identifying information within an authentication token for inspection purposes makes the users dependent of the trustworthiness of the inspector. As soon as the token is submitted to the verifier, the inspector is able to lift the anonymity of the user and disclose her identity. Therefore the role of inspector must be taken by an entity that a user has established trust relationship with. Nevertheless, there exist techniques that could help to avoid putting trust on a single entity but a group of inspectors. In this case, a minimum number of inspectors need to collaborate in order to retrieve the identity information from the presentation token.

T11 *The users need to trust that the inspectors do not disclose their identities without making sure that the inspection grounds hold.*

2.7.3.4 Verifiers' Perspective

Provisioning of the users in the ecosystem is one of the major points where the verifiers have to trust the issuers to precisely check upon the attributes that they are attesting. It holds for any certification scheme that the verifiers rely on the certified information by the issuers for the authentication phase, therefore the issuers assumed to be trustful.

T12 *The verifiers need to trust that the issuers are diligent and meticulous when evaluating and attesting the users' attributes.*

When a user loses her credibility, it is the issuer's responsibility to take the appropriate action in order to block the further use of the respective credentials. Therefore, the verifiers rely on the issuers to immediately request revocation of the user's credentials when a user is not entitled anymore.

T13 *The verifiers need to trust that the issuers will promptly react to inform the revocation authorities when a credential loses its validity.*

In an authentication scenario where inspection is enabled, the only party who is able to identify a misbehaving user is the inspector. The verifier is not able to deal with the case if the inspector does not to cooperate. Therefore, similar to trust relationship T11 by the users, the verifiers dependent of the fairness and honesty of the inspector. Moreover, in a similar fashion, the trust can be distributed to more than one inspector to reduce the risk of misbehaviour. In this case, a subset of all the inspectors would enough to proceed with the inspection.

T14 *The verifiers need to trust that the inspectors fulfil their commitments and will investigate the reported cases fairly and deliver the identifiable information in case of verified circumstances.*

The validity of credentials without expiration information is checked through the information that the verifier acquires from the revocation authority. A compromised revocation authority can deliver outdated or illegitimate information to enable a user to get access to resources even with revoked credentials. Therefore the revocation authority needs to be a trusted entity from the verifiers' perspective.

T15 *The verifiers need to trust that the revocation authorities perform honestly and deliver the latest genuine information to the verifiers.*

Often user credentials are designed for individual use, and sharing is not allowed. Even though security measures such as hardware tokens can be employed to support this policy and limit the usage of the credentials to their owners, the users can still share the tokens and let others benefit from services that they are not normally eligible for. The verifiers have no choice than trusting the users and the infrastructure on this matter.

T16 *The verifiers need to trust that the users do not share their credentials with the others, if this would be against the policy.*

2.7.3.5 Issuers' Perspective

As mentioned earlier T13, the issuer is responsible to take the appropriate steps to block further use of a credential when it loses its validity. The issuer has to initiate the revocation process with the revocation authority and trust that the revocation authority promptly reacts to it in order to disseminate the revocation status of the credential. For instance, when a user cancels her subscription for an online magazine, the publisher would like to stop her access to the service right after the termination of the contract. A compromised revocation authority can delay or ignore this process to let the user benefit from existing services.

T17 *The Issuers need to trust that the revocation authorities perform honestly and react to the revocation requests promptly and without any delay.*

2.7.3.6 Inspectors' Perspective

In order to have a fair inspection process, the inspection grounds must be precisely and clearly communicated to the users in advance. It can be said that presenting inspection grounds is as challenging as privacy policies where long, ambiguous and tedious texts would cause typical users to overlook or misunderstand the conditions. Therefore, in case of an inspection request, the inspector has to rely on the verifier that the users had been informed about these conditions properly.

T18 *The Inspector need to trust that the verifier has properly informed the users about the actual circumstances that entitle the verifier for de-anonymisation of the users.*

2.7.3.7 Revocation Authorities' Perspective

Revocation authorities are in charge of delivering up-to-date information about the credentials' revocation status to the users and the verifiers. However, they are not in a position to decide whether a credential must be revoked or not, without receiving revocation requests from the issuers. Therefore, their correct operations depends on the diligent performance of the issuers.

T19 *In order to provide reliable service, the revocation authorities need to trust that the issuers deliver legitimate and timely notice of the credentials to be revoked.*

2.8 Policy-based View of the Architecture

Policy can be represented at different levels, ranging from business goals to device-specific configuration parameters [WSS⁺01]. In this section, with the term “policy” we refer to a more abstract concept than the *issuance policy* and *presentation policy* artefacts of ABC4Trust. We consider policy to be “a definite goal, course or method of action to guide and determine present and future decisions”, as defined in [WSS⁺01]. A view on the ABC4Trust architecture from this policy perspective delivers useful observations, even though policy handling is something that happens at a layer higher than the ABC4Trust architecture.

The ABC4Trust architecture does not define the roles and the corresponding operational processes for Policy Decisions Points (PDP) [WSS⁺01] and Policy Enforcement Points (PEP) [WSS⁺01], as this falls outside of its scope. However, we would like to emphasize that the ABC4Trust architecture offers valuable technical possibilities, awareness of which can be useful when designing and implementing PDPs. In the next step, it provides the technical means to support the Policy Actions [WSS⁺01] and enables the PEPs. To elaborate more, we consider an example for different stages in the life-cycle of Privacy-ABCs, namely, issuance, presentation, revocation and inspection. Specifically for the inspection and revocation phase, let us take the example of the Söderhamn pilot, which supports both.

One of the interesting features of Privacy-ABCs that concerns the credential issuance phase is the *carry-over attribute*. It allows blind transfer of an attribute value from another credential to the one being issued. A typical use of such mechanism is when the credential is issued to anonymous users but the issuer needs to make sure that the new credential cannot be transferred to anybody else. Therefore, the issuer binds the credential to the user’s identity (e.g., Passport NR), retrieved as an attribute from another trusted credential that the users holds, but without actually being able to see the attribute value. Knowing about such a feature, which does not exist in the common identity management platforms, could prevent the decision makers from investing in much more expensive infrastructure and processes in order to achieve the same goal. When such a decision is made, it can be expressed in the XML artefact *issuance policy* and enforced when a credential issuance is taking place.

With regard to the presentation phase, the knowledge that attributes can be technically treated separately helps privacy advocates make the argument that credentials should only contain the minimum information, e.g. whether the user is of legal age and that there is no need to collect more information, while still all the required guarantees are offered to the relaying party.

Taking the Söderhamn pilot as an example: the school administration acted as a PDP by deciding (for compliance with Swedish regulations for schools) that the School Community Interaction Platform must make it possible for misbehaving students to be identified - in specific cases such as bullying or harassment. The decision was expressed in the *presentation policy* of the various sections of the system so that only Privacy-ABC *presentation tokens* with inspectable data would be considered to grant access to the activity area. As a result, there was a process to reveal the iden-

tity of misbehaving students in extraordinary circumstances. The PEP was where the system requested the users to include inspection data in their Privacy-ABC presentation tokens to access a resource. A possible further PEP would be at the entity executing the inspection (possibly the school management together with another entity, so that the 4-eye principle would be followed). Note that Privacy-ABCs allow to change the policy requiring inspectable tokens at any point in time without the need to reissue credentials.

In the case of credential revocation, the school administration decided that whoever is not part of the school anymore should not be able to participate in the activities of the community platform. This decision was reflected by the application designers in the *credential specification* as well as the deployment architecture in order to enable the revocation process. We can consider two points where the policy enforcement was taking place: The first point was at the submission of the revocation request by the school administration to the revocation authority. In the next stage, the policy was enforced everytime the platform refused to give access to a user with a revoked credential.

References

- [abc] ABC4Trust EU Project. <https://www.abc4trust.eu>.
- [ASN08] Abstract syntax notation one (ASN.1), 2008. International Telecommunication Union - ITU-T recommendation X.680.
- [Bar02] Bartel, Mark and Boyer, John and Fox, Barb and LaMacchia, Brian and Simon, Ed. XML-Signature Syntax and Processing. <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>, February 2002.
- [BBE⁺14] Thomas Baignères, Patrik Bichsel, Robert R Enderlein, Hans Knudsen, Kasper Damgård, Jonas Jensen, Gregory Neven, Janus Nielsen, Pascal Paillier, and Michael Stausholm. Final Reference Implementation. Deliverable D4.2, The ABC4Trust EU Project, 2014. Available at <https://abc4trust.eu/download/D4.2%20Final%20Reference%20Implementation.pdf>, Last accessed on 2014-11-08.
- [BCD⁺14] Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Stephan Krenn, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Janus Dam Nielsen, Christian Paquin, Franz-Stefan Preiss, Kai Rannenberg, Ahmad Sabouri, and Michael Stausholm. Architecture for Attribute-based Credential Technologies - Final Version. Deliverable D2.2, The ABC4Trust EU Project, 2014. Available at https://abc4trust.eu/download/Deliverable_D2.2.pdf, Last accessed on 2014-11-08.
- [Bra] Stefan Brands. The ID Corner blog. The problem(s) with OpenID. <http://www.untrusted.ca/cache/openid.html>.

- [CDF⁺] J. Callas, L. Donnerhackle, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. <http://www.rfc-editor.org/rfc/rfc4880.txt>.
- [Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.
- [CKL⁺14] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael østergaard Pedersen. Scientific Comparison of ABC Protocols: Part I Formal Treatment of Privacy-Enhancing Credential Systems. Deliverable D3.1, The ABC4Trust EU Project, 2014. Available at <https://abc4trust.eu/download/Deliverable\%20D3.1\%20Part\%201.pdf>, Last accessed on 2014-11-08.
- [Cro06] Douglas Crockford. The application/json media type for JavaScript Object Notation (JSON). Technical Report RFC 4627, Internet Engineering Taskforce (IETF), 2006.
- [CSF⁺08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Technical report, IETF, May 2008.
- [DFLP07] Nelly Delessy, Eduardo B Fernandez, and Maria M Larrondo-Petrie. A pattern language for identity management. In *Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*, pages 31–31. IEEE, 2007.
- [DW10] Arkajit Dey and Stephen Weis. PseudoID: Enhancing Privacy in Federated Login. In *Hot Topics in Privacy Enhancing Technologies*, pages 95–107, 2010.
- [Fac] Facebook Login. <https://developers.facebook.com/products/login/>.
- [Fid] Fido Alliance. <http://fidoalliance.org>.
- [Har04] Russell Hardin. *Trust and trustworthiness*, volume 4. Russell Sage Foundation, 2004.
- [Har12] Dick Hardt. OAuth 2.0 Authorization Protocol. <http://tools.ietf.org/html/rfc6749>, October 2012.
- [HBH07] Dick Hardt, Johnny Bufu, and Josh Hoyt. OpenID Attribute Exchange 1.0. http://openid.net/specs/openid-attribute-exchange-1_0.html, December 2007.
- [HTEG10] D. Hardt, A. Tom, B. Eaton, and Y. Goland. OAuth Web Resource Authorization Profiles. <http://tools.ietf.org/html/draft-hardt-oauth-01>, January 2010. draft version 19 at time of writing.
- [JBT] M. Jones, J. Bradley, and H. Tschofenig. Proof-Of-Possession Semantics for JSON Web Tokens (JWTs). <http://tools.ietf.org/html/draft-jones-oauth-proof-of-possession-00>.

- [JP04] Audun Jøsang and Stéphane Lo Presti. Analysing the relationship between risk and trust. In *Trust Management*, pages 135–145. Springer, 2004.
- [JWT] Json web token (jwt). <http://datatracker.ietf.org/doc/draft-ietf-oauth-json-web-token>. draft version 19 at time of writing.
- [KBC05] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93–108, 2005.
- [KTMM09] Uwe Kylau, Ivonne Thomas, Michael Menzel, and Christoph Meinel. Trust requirements in identity federation topologies. In *Advanced Information Networking and Applications, 2009. AINA'09. International Conference on*, pages 137–145. IEEE, 2009.
- [LSK12] Jesus Luna, Neeraj Suri, and Ioannis Krontiris. Privacy-by-design based on quantitative threat modeling. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pages 1–8. IEEE, 2012.
- [MC96] D. Harrison Mcknight and Norman L. Chervany. The Meanings of Trust. Technical report, University of Minnesota, 1996.
- [MCM14] C. Mortimore, B. Campbell, and Jones M. SAML 2.0 Bearer Assertion Profiles for OAuth 2.0. <http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-19>, March 2014. draft version 19 at time of writing.
- [O'H04] Kieron O'Hara. *Trust: From Socrates to Spin*. Icon Books Ltd, 2004.
- [Ope] OpenID Connect. <http://openid.net/connect/>.
- [Ope07] OpenID Authentication 2.0. http://openid.net/specs/openid-authentication-2_0.html, December 2007.
- [SAM05] Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>, March 2005.
- [UPW11] U-Prove WS-Trust Profile V1.0. <http://www.microsoft.com/u-prove>, March 2011.
- [WS-12] WS-Trust 1.4. <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>, April 2012.
- [WSF09] Web Services Federation Language (WS-Federation) Version 1.2. <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html>, May 2009.
- [WSS⁺01] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson, and Steve Waldbusser. Terminology for Policy-Based Management. Internet RFC 3198, November 2001.
- [WSS07] WS-SecurityPolicy 1.2. <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cs.html>, April 2007.

- [WST09] WS-Trust 1.4. <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.html>, February 2009.



<http://www.springer.com/978-3-319-14438-2>

Attribute-based Credentials for Trust
Identity in the Information Society

Rannenbergh, K.; Camenisch, J.; Sabouri, A. (Eds.)

2015, XV, 391 p. 122 illus., Hardcover

ISBN: 978-3-319-14438-2