D2.2 - Architecture for Attribute-based Credential Technologies - Final Version

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, Michael Stausholm

Editor:	Ahmad Sabouri (Goethe University Frankfurt)
Reviewers:	Norbert Götze (Nokia Solutions and Networks), Jonas Lindstrøm Jensen (Alexandra Institute)
Identifier:	D2.2
Type	Deliverable
Version	1.0
Date:	14/08/2014
Status:	Final
Class:	Public

Abstract

The goal of ABC4Trust is to address the federation and interchangeability of technologies that support trustworthy yet Privacy-preserving Attribute-based Credentials (Privacy-ABCs). Towards this goal, one of the main objectives of the project is to define a common, unified architecture for Privacy-ABC systems to allow comparing their respective features and combining them into common platforms. The first version of this architecture is described in deliverable D2.1 of the project. Its main contribution is the specification of the data artefacts exchanged between the implicated entities (i.e. issuer, user, verifier, revocation authority, etc.), in such a way that the underlying differences of concrete Privacy-ABC implementations are abstracted away through the definition of formats that can convey information independently from the mechanism-specific cryptographic data. It also defines all technology-agnostic components and corresponding APIs a system needs to implement in order to perform the corresponding operations. This Deliverable (D2.2) comes to present the final version of the architecture. This document targets to keep early adopters up-to-date, so it presents only those changes that are relevant to the development of applications and removes the details of the internal components.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Members of the ABC4TRUST consortium

1.	Alexandra Institute AS	ALX	Denmark
2.	CryptoExperts SAS	CRX	France
3.	Eurodocs AB	EDOC	Sweden
4.	IBM Research Zurich	IBM	Switzerland
5.	Johann Wolfgang Goethe Universität Frankfurt	GUF	Germany
6.	Microsoft NV	MS	Belgium
7.	Miracle A/S	MCL	Denmark
8.	Nokia Solutions and Networks Management International	NSN	Germany
9.	Computer Technology Institute & Press - "DIOPHANTUS"	CTI	Greece
10.	Söderhamn Kommun	SK	Sweden
11.	Technische Universität Darmstadt	TUD	Germany
12.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany

List of Contributors

Chapter	**Author(s)**
Executive Summary	Ahmad Sabouri (GUF), Ioannis Krontiris (GUF)
First Chapter	Ahmad Sabouri (GUF), Ioannis Krontiris (GUF), Kai Rannenberg (GUF)
Second Chapter	Gregory Neven (IBM)
Third Chapter	Jan Camenisch (IBM), Anja Lehmann (IBM)
Fourth Chapter	Patrik Bichsel (IBM), Jan Camenisch (IBM), Maria Dubovitskaya (IBM), Robert R. Enderlein (IBM), Anja Lehmann (IBM), Gregory Neven (IBM), Christian Paquin (MS), Franz-Stefan Preiss (IBM)
Fifth Chapter	Jan Camenisch (IBM), Anja Lehmann (IBM), Gregory Neven (IBM), Janus Dam Nielsen (ALX), Christian Paquin (MS), Franz-Stefan Preiss (IBM), Michael Stausholm (ALX)
Sixth Chapter	Patrik Bichsel (IBM), Robert R. Enderlein (IBM), Stephan Krenn (IBM)
Seventh Chapter	Jan Camenisch (IBM), Maria Dubovitskaya (IBM), Robert R. Enderlein (IBM), Anja Lehmann (IBM), Gregory Neven (IBM), Christian Paquin (MS), Franz-Stefan Preiss (IBM)
Eighth Chapter	Ahmad Sabouri (GUF), Ioannis Krontiris (GUF), Kai Rannenberg (GUF)
Ninth Chapter	Christian Paquin (MS)

Executive Summary

This deliverable presents the the final version of the architectural framework for Privacy-ABC technologies that has been produced within the ABC4Trust project. The architecture allows different realizations of these technologies to coexist, be interchanged, and federated. This enables users to obtain credentials following different Privacy-ABC technologies and use them indifferently on the same hardware and software platforms, as well as service providers to adopt whatever Privacy-ABC technology best suits their needs.

More specifically, the architecture has been designed to decompose future implementations of Privacy-ABC technologies into sets of modules and specify the abstract functionality of these components in such a way that they are independent from algorithms or cryptographic components used 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.

The architecture of ABC4Trust is defined by following a layered approach, where the so-called ABCE (ABC Engine) top layer provides the application developers with a technology agnostic API, thereby abstracting the internal design and structure. Furthermore, the architecture defines 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 document emphasizes on the XML based specification of the corresponding messages exchanged during the issuance, presentation, revocation, and inspection of Privacy-ABCs.

The architecture has been improved compared to version 1, considering the valuable feedback we received from its realization in the reference implementation work package (WP4) as well as its deployment in the ABC4Trust pilots. This deliverable removes the details of how the ABCE layer looks internally and gives a simpler and more modular explanation of its functionality. Correspondingly, it presents an updated "external" API that the ABCE layer offers to the application layer, as well as an updated version of the data formats. It also presents some updates in the definition of concepts and features of Privacy-ABCs.

This deliverable also presents the new crypto architecture, which replaces version 2 of IBM's Identity Mixer (Idemix) Library. Its main advantage compared to the old Idemix library is the increased modularity of its design. This modularity allowed us to implement additional features, such as supporting U-Prove credentials, and a predicate for checking linear combinations among attributes.

In order to facilitate understanding and adoption of the ABC4Trust architecture, the document provides an analysis of the trust relationships required in the ecosystem of Privacy-ABCs as well as an example scenario of an online library to better illustrate the different artefacts of the language framework.

Consideration is also given on how the proposed architecture integrates with existing identity management systems. The deliverable provides an analysis regarding the applicability of the ABC4Trust architecture to the popular existing identity protocols and frameworks such as WS-*, SAML, OpenID, OAuth and X.509. It shows how Privacy-ABCs can be applied in these protocols and how the former can help to alleviate some of their security, privacy and scalability issues of the latter.

Contents

List of Figures

1 Introduction

Many electronic applications and services require some authentication of participants to establish trust relations, either for only one endpoint of communication or for both. One widely used mechanism for this is password-based authentication. Today, individuals are asked to maintain dozens of different usernames and passwords, one for each website with which they interact. This authentication mechanism is not always optimal, since it creates a burden to individuals and encourages the reuse of passwords through multiple services, which in return makes online fraud and identity theft easier. Spoofed website, stolen passwords and compromised accounts have negative impact not only to individuals but also to businesses and governments, who are unable to offer high-value online services.

Strong authentication techniques can deliver higher level of security, however, the commonly used ones introduce some privacy issues. For instance, X.509 [X50] suffers from the "Over Identification" problem, meaning that due to its static nature, the user has to reveal all the attributes in the credential even if only a subset of them is required. Furthermore, technologies like OpenID [HBH07] that allow selective disclosure of the attributes, cause the so-called "Calling Home" problem that enables the Identity Service Provider to profile the transactions of the user.

In this chapter, we start with a brief discussion about the privacy issues concerning the current IdM systems. Then we introduce Privacy-ABCs and how they can be used to effectively resolve these privacy issues. Following that, we go through the objectives and the goals, which ABC4Trust project is aiming for, and continue with describing the design decisions and strategies which have been considered in the ABC4Trust architectural work. The last section provides an overview of the document structure and the organization of the following chapters.

1.1 Privacy issues of current IdM systems

In their everyday offline transactions, people have to present credentials in order to perform a number of operations. There are several aspects in these transactions that are privacy-respecting, but have not been preserved in similar transactions online. For example, when individuals have to present their ID card to open a bank account or board an airplane, the government authority issuing the ID cards does not learn about every place individuals have to present their card.

However, there are also some aspects of offline transactions that are not privacy-respecting. For example, showing the ID card to buy alcohol at the store reveals extraneous information, such as the exact date of birth or the address (e.g. in the case of German ID Card), while what is actually requested is to prove that one is over a certain age. This is not really a problem in the offline world, because the infrastructure (i.e., the clerk behind the counter) is not equipped to log and remember all disclosed information; but things change in the online world: disclosed information is forever remembered.

Users' online privacy is increasingly threatened as a number of countries are introducing or are about to introduce electronic identity cards (eID) and drivers licenses [FP10], expanding the use of credentials in the online world. Moreover, electronic ticketing and toll systems are also widely used all over the world. As such electronic devices become widespread for identification, authentication, and payment (which links them to people through credit card systems) in a broad range of scenarios.

One desirable goal of building online identity management systems should be to keep the privacy-respecting aspects of the offline paradigm and resolve the negative aspects. To see the problems that emerge for privacy, one has to observe the flow and storage of information between the involved entities. A typical identity management is based on the relations between three core parties: the user (U) who requests a service from the Relying Party (RP) that offers the service and relies on the Identity Service Provider (IdSP) to provide authentic information about the identity of the user.

Here we briefly review some of the main privacy aspects of the practices followed today, based on the parties and the relations introduced [Bra00].

1.1.1 IdSP knows all user transactions

Even though the IdSP does not necessarily need to know where the user is authenticating and which service she is requesting for, this happens in a large portion of the existing federated identity management systems. More specifically, in those systems where the Identity Service Provider is involved each time a user authenticates to a Relying Party, the IdSP might be able to keep track of the user actions, depending on the exchanged information between the IdSP and the RP. This enables the IdSP to trace and link all communications and transactions of each user. Moreover, if the user does not perform an active role in the information exchange between the IdSP and the RP (e.g. OpenID [HBH07]), there is a high security risk of identity theft and impersonation of the user by the IdSP or an intruder who has gained access to the IdSP resources.

1.1.2 Linkability across domains

In today's identity management systems, each Relying Party can store all the tokens that are presented to it, and can link them together. The simplest example is X.509 certificates [X50] where the certificate's public key and issuer's signature act as kind of digital fingerprint, inescapably leaving a digital trail wherever the citizen presents the certificate. In this manner, dossiers can automatically be compiled for each individual about his or her habits, behavior, movements, preferences, characteristics, and so on. Different Relying Parties can exchange and link their data on the same basis.

1.1.3 Proportionality often violated

There are many scenarios where the use of certificates unnecessarily reveals the identity of their holder, for instance scenarios where a service platform only needs to verify the age of a user but not his/her actual identity.

A typical example is the citizen PKI, where each citizen is provided with a X.509 certificate [X50] as the digital identifier for securely accessing the online services offered by the governments. These certificates contain a set of attributes such as full name, date of birth, gender, and ID number, and inevitably all will be revealed to the Relying Party each time the certificate is presented.

Revealing more information than necessary not only harms the privacy of the users, but also increases the risk of abuse of information such as identify theft when information revealed falls in the wrong hands.

1.2 Privacy-preserving Attribute-based Credentials

Over the past years, a number of technologies have been developed to build Privacy-preserving Attribute-based Credential (Privacy-ABC) systems in a way that they can be trusted, like normal cryptographic certificates, while at the same time they protect the privacy of their holder, resolving the problems discussed in the previous section, in addition to other properties.

Such Privacy-ABCs are issued just like ordinary cryptographic credentials (e.g., X.509 credentials) using a digital (secret) signature key. However, as we will see in Chapter 2, Privacy-ABCs allow their holder to derive new tokens, in such a way that the privacy of the user is protected. Still, these transformed tokens can be verified just like ordinary cryptographic credentials (using the public verification key of the issuer) and offer the same strong security.

There are a handful of proposals on how to realize a Privacy-ABC system in the literature [Cha85, Bra94, CL01, CL04]. Notable is especially the appearance of two technologies, IBM's Identity Mixer and Microsoft's U-Prove, as well as extended work done in past EU projects. In particular, the EU-funded projects PRIME[1] and PrimeLife[2] have actually shown that the state-of-the art research prototypes of Privacy-ABC systems can indeed confront the privacy challenges of identity management systems.

[1] www.prime-project.eu
[2] www.primelife.eu

The PRIME project has designed an architecture for privacy-enhancing identity management that combines anonymous credentials with attribute-based access control, and anonymous communication. That project has further demonstrated the practical feasibility with a prototypical implementation of that architecture and demonstrators for application areas such as e-learning and location-based services. PRIME has, however, also uncovered that in order for these concepts to be applicable in practice further research is needed in the areas of user interfaces, policy languages, and infrastructures.

The PrimeLife project has set out in 2008 to take up these challenges and made successful steps towards solutions in these areas. For instance, it has shown that Privacy-ABC systems can be employed on smartcards and thus address the requirements of privacy-protecting eID cards [BCGS09]. Also, in the last decade, a large number of research papers have been published solving probably all roadblocks to employ Privacy-ABC technologies in practice. This includes means to revoke certificates [Ngu05, BDDD07, CL02c, CKS09], protection of credentials from malware [Cam06], protection against credential abuse [CHK+06, CHL06], proving properties about certified attributes [CG08, CC+08], and means to revoke anonymity in case of misuse [CS03b].

Despite all of this, the effort of understanding Privacy-ABC technologies so-far was rather theoretical and limited to individual research prototypes. Indeed, PRIME and PrimeLife showed that Privacy-ABC technologies provide the desirable level of privacy protection, but so far this was demonstrated in a very limited number of actual production environments with real users.

Furthermore, there has been no commonly agreed set of functions, features, formats, protocols, and metrics to gauge and compare these Privacy-ABC technologies, and it was hard to judge the pros and cons of the different technologies to understand which ones are best suited to which scenarios.

Thus, there has been a gap between the technical cryptography and protocol sides of these technologies and the reality of deploying them in production environments. A related problem with these emerging technologies was the lack of standards to deploy them. For instance, a position paper published by ENISA on "Privacy Features of European eID Card Specifications" [NA09] observed that Privacy-ABC "technologies have been available for a long time, but there has not been much adoption in mainstream applications and eID card applications" even though countries such as Austria and Germany have taken some important steps in this sense.

1.2.1 Privacy-ABCs Highlights

A detailed explanation of the features and properties of Privacy-ABCs is presented in Chapter 2. Nevertheless, here we intend to highlight what Privacy-ABCs in principal can offer.

1. Minimal Disclosure: Privacy-ABCs allow the users to derive authentic and verifiable tokens from their credentials that contain only the required information, therefore avoid disclosing all the attributes in the credentials. For instance, citizens of a country can obtain an identity credential from their government and use this credential to participate in an online opinion poll of their residence district with just disclosing the postcode attribute of their address. Furthermore, it is even possible to make proof of owning a specific type of credential without revealing any of the attested attributes. As an example, student of Frankfurt University could get free access to an online cinema providing a proof of holding a valid "Frankfurt University Student Credential".
2. Unlinkability: Another key feature of Privacy-ABCs is unlinkability that comes in two different types, namely, unlinkability to their issuance, and unlinkability across different presentations. The former indicates that the issuance and the usage (presentation) of a credential cannot be correlated unless due to the disclosed attributes. Therefore, if the user does not reveal any identifiable information, the credential issuer and the service provider (verifier) cannot learn more about the user even if they collude. Similarly, in the latter case, it would not be possible to associate different transactions of the same user when the revealed information do not provide any linkability.
3. Partial Identities and identifiers: The concept of Pseudonyms in Privacy-ABCs facilitates establishing different profiles and putting borders between different activities of an individual within the same context or different ones, and therefore achieve a controlled level of linkability. Pseudonyms are

similar to public keys and are derived from the users' secret keys. Nevertheless, unlike public keys of which there is only one for every secret key, users can generate an unlimited number of unlinkable pseudonyms for a single secret key.

1.3 The ABC4Trust Project

The aim of the ABC4Trust project is to deepen the understanding in Privacy-ABC technologies, enable their efficient and effective deployment in practice, and their federation in different domains. To this end, the project:

1. produces an architectural framework for Privacy-ABC technologies that allows different realizations of these technologies to coexist, be interchanged, and federated:
   (a) Identify and describe the different functional components of Privacy-ABC technologies, e.g. for issuance of credentials and presentation of tokens;
   (b) Produce a specification of data formats, interfaces, and protocol formats for this framework;
2. Defines criteria to compare the properties of realizations of these components in different technologies; and
3. Provides reference implementations for each of these components.

With a comparative understanding of today's available Privacy-ABC technologies, it will be easier for different user communities to decide which technology best serves them in which application scenario. It will also be easier to migrate to newer Privacy-ABC technologies that will undoubtedly appear over time. In addition, the same users may want to access applications requiring different Privacy-ABC technologies, and the same applications may want to cater to user communities preferring different Privacy-ABC technologies. Finally, the architecture and protocol specifications proposed by the ABC4Trust project pave the road towards establishing standards that allow for the interchangeability and federation of Privacy-ABC technologies.

1.4 The ABC4Trust Architecture

From the three project goals above, this document focuses on the first one, namely the architecture for Privacy-ABC technologies. This is presented extensively in the chapters to follow. It is however useful for the reader to first understand the goals and design considerations that were taken into account during the design of this architecture. This subsection elaborates on these decisions and prepares the reader for the chapters that follow.

1.4.1 Goals of the Architecture

A contribution of this project to the state of the art is the definition of a common unified architecture for federating and interchanging different Privacy-ABC systems in a way that

1. users will be able to obtain credentials for many Privacy-ABC technologies and use them indifferently on the same hardware and software platforms, and
2. service providers and IdSPs will be able to adopt whatever Privacy-ABC technology best suits their needs.

Furthermore, the architecture has been designed to decompose implementations of Privacy-ABC technologies into sets of modules and specify the abstract functionality of these components in such a way that they are independent from algorithms or cryptographic components used 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.

Indeed the project aims not only to federate Privacy-ABC systems but to let them coexist on the same platform. This implies that different systems must be able to share common architecture elements such as

user interfaces or credential stores. Thus, common APIs must be enforced across different Privacy-ABC implementations to ensure their possible coexistence and interchangeability on the same network nodes. Similarly, different systems should use common communication wrappers to encode and exchange tokens and other items when communicating with peers on different network nodes, so that a token recipient can immediately determine what Privacy-ABC technology the token pertains to.

As a result, the architectural framework specifies unified data formats and protocols across Privacy-ABC implementations to enable not just coexistence and interchangeability on the same network nodes but also coexistence and possible federation across different network nodes.

1.4.2 Architectural Strategies

This section describes the design decisions and strategies that affect the overall organization of the architecture and its higher-level structures.

1.4.2.1 A Layered Approach

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. So, the focus of the ABC4Trust architecture is to define and standardise the ABCE layer and its interfaces to the upper layers (e.g. Application). With this respect, it does not analyse the internals of the other layers, but it only concentrates on defining the interfaces necessary for those layers to use the functionality of the ABCE and incorporate the corresponding tokens in the overall system. Equally important in the architecture is the specification of the data artefacts exchanged between the implicated entities, 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.

In particular, this document concentrates on the following aspects:

- *Functionality and interfaces*– We define the functionality of the different layers focusing on the ABCE layer and its components (see Chapter 3). We then describe how to integrate and use the ABCE layer along the main use-cases, i.e. presentation of a token, issuance of a credential, inspection and revocation. For each of these phases, we also describe the corresponding interfaces offered by the ABCE layer to the application layer (see Chapter 5), so that developers can build easily ABC-enabled applications.
- *Data specification* – The issuance and presentation of Privacy-ABC credentials are interactive processes, potentially involving multiple exchanges of messages. Chapter 4 defines the contents, encoding and processing of these messages. In particular, it specifies the data artefacts exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials. Note that the document remains generic on which specific protocols are used to issue or present Privacy-ABCs. There are several existing messaging protocols, in which these credentials can be embedded, or new ones could be defined in the future.
- *New crypto architecture* – we also present a modular model for the crypto layer. The main responsibilities of the Cryptographic Engine are to generate cryptographic key material, issue new credentials by means of a two-party protocol, generate the cryptographic evidence for a Presentation Token to prove that a user satisfies a Presentation Policy, and verify such a proof. This crypto architecture defines the building blocks of Privacy-ABC systems and their interfaces allowing implementation of additional features and extending the functionalities.

1.4.2.2 Building Privacy-ABC-enabled applications

ABC4Trust has provided an open reference implementation of the architecture described in this document as part of its contributions. The reference implementation of ABC4Trust has embedded into example applications showing how to integrate the reference components into a sample client-server platform.

Application developers can integrate the reference implementation of the ABC4Trust architecture directly in their applications, without having to know how its layers are internally structured. That means the application can incorporate user authentication functionality using Privacy-ABC, according to the access policy of the requested service, by executing directly the necessary protocols for policy and token exchange. For that, the application developers must simply follow the interfaces and data formats described in this document.

To facilitate adoption of Privacy-ABCs, Chapter 7 considers an example scenario of an online library to better illustrate the different artefacts in the language framework of the ABC4Trust architecture. Moreover, in Chapter 8, we present an analysis of the trust relationships required in the ecosystem of Privacy-ABCs so that the technology adopters and decision makers get a better understanding of the setting.

Nevertheless, other approaches are also possible to integrate Privacy-ABCs. For example, following a passive federated redirection pattern, the application may redirect the user to a Relying Party Secure Token Service (RP-STS) component for authentication. This is shown and discussed in more details in Chapter 9, where we discuss how the ABC4Trust architecture can be integrated with existing federated systems.

1.5 Structure of the document

The rest of this document is organized as follows:

Chapter 2 gives an introduction to the features supported by Privacy-ABCs and the actors involved, in different kind of interactions, namely the presentation of tokens, inspection, credential issuance and revocation.

Chapter 3 presents the modules of the ABC4Trust architecture and concentrates in particular on the ABCE layer. It revisits each of the scenarios introduced in Chapter 2 and shows specifically how they are performed by the ABCE modules.

Chapter 4 can be considered as the core part of this document. It provides the specification for data artefacts exchanged during the Privacy-ABCs lifecycle (issuance, presentation, revocation, and inspection). It introduces an XML notation to specific all the necessary data formats, e.g. credentials contents, access policies, presentation tokens, etc., as well as their wrapper message formats.

Chapter 5 defines the APIs for each of the ABCE modules. More specifically, it specifies the interfaces exposed to the outside world (and in particular to the application layer).

Chapter 6 presents the proposed architecture for the crypto layer. More specifically, it presents the building blocks and their interconnections.

Chapter 7 provides an example scenario to further illustrate on the XML artefacts of the language framework.

Chapter 8 analyses the ecosystem of Privacy-ABCs and explains the trust relationships that are required between the involved entities.

Chapter 9 presents an overview of the most popular identity protocols and frameworks (e.g. WS-*, SAML, OpenID, OAuth, and X.509) and describes the common challenges of these federated systems concerning security, privacy and scalability. The analysis provided in this chapter, demonstrate how Privacy-ABC technologies can help to alleviate these challenges. The reader may note that in this chapter the Identity Service Provider is named “Identity Provider”. The reason for this is, that many of the existing protocols use this term, though it is misleading, as the respective entity does not provide identities.

1.6 What is new compared to H2.2

This document is an update of Heartbeat H2.2. The most important differences compared to H2.2 are listed below; other changes are mostly minor.

- We added a section explaining the security and privacy features of Privacy-ABCs (Section 2.8), without going into the details of formal cryptographic definitions.
- We updated the architecture chapter to be consistent with the reference implementation (Section 3).
- We added a new section describing the new cryptographic architecture (Section 6).
- We also added a new section with the XML artefacts for an example usage scenario of an online library (Section 7).
- Finally, we added a new section describing the various trust relationships in a complete Privacy-ABC system (Section 8).

2 Features and Concepts of Privacy-ABCs

This section provides a detailed explanation on the features supported by privacy-enhancing attribute-based credentials (Privacy-ABCs), on the different involved entities, and on the type of interactions that they engage in.

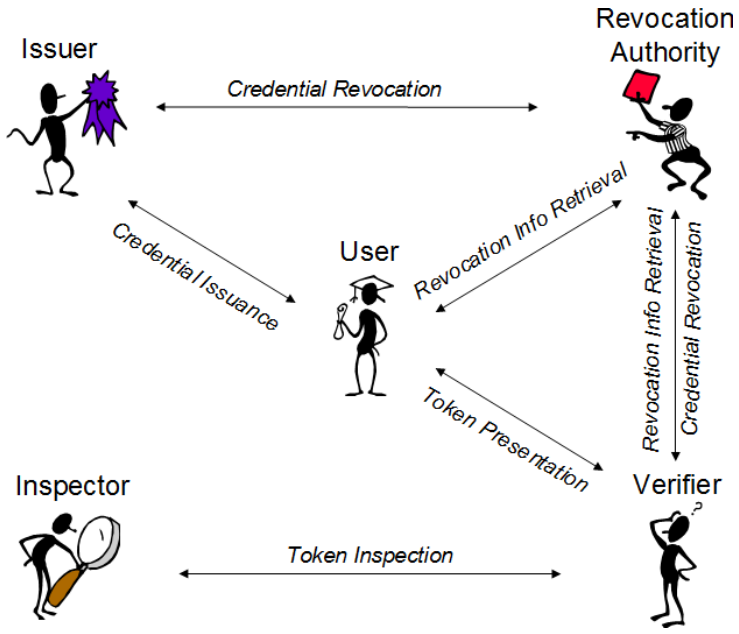

Figure 1: Entities and interactions diagram

Figure 1 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 requester or the 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 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 required information and the supporting cryptographic evidence. In the identity management literature, the Verifier is sometimes also referred to as the relying party, the server, or the 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

in the sense that being revoked by one Verifier does not affect a user's 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 already have in their possession or trust the Issuer and Revocation Authority data needed to validate a presentation token. Nothing prevents, however, a User to collect this data and present it to the verifier in a certified manner in a setup phase by piggybacking on an existing infrastructure (e.g., by signing the artifacts using an X.509 certificate). This would add flexibility to the system and allow dynamic trust establishments between the parties.

2.1 Credentials

A *credential* is a certified container of attributes issued by an Issuer to a User. An attribute is described by the *attribute type*, determining the semantics of the attribute (e.g., first name), and the *attribute value*, determining 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. Of course, the trustworthiness of the Issuer's statement depends on its identity vetting processes: an Issuer that requires Users to physically show up at an office with a valid picture ID is probably more reliable than an Issuer that certifies any self-claimed value without further evidence.

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 specifies how the attribute values are mapped to their integer representation. Depending on the data type and size of the attribute value, this encoding may involve a cryptographic hash to be applied.

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 public-key infrastructure (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.

2.2 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 $<$ 1996 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 the presentation token must reveal from these credentials. 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 all public information, including 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 in principle cryptographically unlinkable and untraceable, 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, we will later discuss additional mechanisms that, with the User's consent, enable a dedicated third party to recover this link again (see Section 2.5 for more details).

Obviously, presentation tokens are only as unlinkable as the information 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.

2.3 Key Binding

To prevent "credential pooling", i.e., multiple Users sharing their credentials, credentials can optionally be bound to a *secret key*, i.e. a cryptographically strong random value that is assumed to be 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 rightful owner of the credential was involved in the creation of the presentation token. As an extra protection layer, the credentials can also be bound to a trusted physical device, such as a smart card, by keeping the secret key in a protected area of the device. That is, the key cannot be extracted from the device, but the device does participate in the presentation token generation to include an implicit proof of knowledge of this key in the token. Thus, for credentials that are key-bound to a physical device it is impossible to create a presentation token without the device.

The issuance of a key-bound credential can optionally be performed 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 2.6). 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 keys. Thereby, the secret key becomes a valuable "master secret" that, when revealed to a third party, allows the latter to take over the User's entire digital identity.

2.4 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 2.3 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 a corresponding presentation token and is useful in login scenarios, e.g., to replace usernames/passwords.
- *Certified pseudonyms* are verifiable pseudonyms derived from a secret key that also underlies an issued credential. By imposing same-key binding in the presentation policy and token (see Section 2.3), 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 the user's identity.
- *Scope-exclusive pseudonyms* are verifiable pseudonyms that are guaranteed to be unique per scope string and per 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.

2.5 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. Just like Issuer parameters, the User must be able to obtain a trusted copy of the Inspector's public key, e.g., through a PKI.

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 clear text 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.

2.6 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 providing some bootstrap 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 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. carry over "self-claimed" attribute values, i.e., values chosen by the User,
3. be bound to the same secret key as an existing credential or verifiable pseudonym (see Sections 2.3 and 2.4), and

all without the Issuer being able to learn the carried-over attribute values or secret key(s) 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 (see Section 2.2). The User prepares a special presentation token that fulfils the stated presentation policy, but that contains additional cryptographic information to enable carrying over attribute and key binding information. 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.

2.7 Revocation

No identification system is complete without a proper means of revoking credentials. 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 revoked for accessing sport stadiums, but is still valid for voting or submitting tax applications.

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 again. However, 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. This document describes 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 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 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 by the specified Revocation Authority, and that the Issuer denies any responsibility for revoked credentials. Issuer-driven revocation is typically used when credentials have been compromised or lost, or when the User is denied all 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 which by default should not be 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.
- In *Verifier-driven revocation*, 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 banning credentials with blacklisted attribute values from being accepted in a presentation, or restricting access to credentials with whitelisted attribute values.

2.8 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.

2.8.1 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 (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. The detection of the replay attack, i.e., of repeating nonces, must happen in the application built on top of the ABCE. 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, and by switching off cookies in the browser. Achieving complete unlinkability can be hard: intrinsic hardware characteristics of the user's device such as clock skews may be exploitable as unique device fingerprints [KBC05], as may the list of plugins and fonts installed in the user's browser [Eck10].

2.8.2 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.

2.8.3 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.

2.8.4 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.

2.8.5 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 if the attribute values come from a small space or if the adversary 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, to encrypt under a different public key, 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.

2.8.6 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.

3 Architecture

This Section briefly describes the architecture of Privacy-ABC systems, their components and the relations among those and shows how to deploy the provided functionalities in the main usage scenarios.

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 Section 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.

This Section assumes that the reader is already familiar with the general features and concepts of Privacy-ABCs described in Section 2. It gives a high-level description of the Privacy-ABC-core architecture and its components. Thus, it can also be seen as an introduction to Section 4 which describes the data formats that are exchanged among Privacy-ABC entities and to Section 5 that presents the application programming interfaces (API) of the *ABCE* layer components. Note that this Section already refers to the *external* methods provided by the *ABCE* layer which are described in more detail in Section 5.

We start by describing the main functionalities of the different layers and components in Section 3.1. We then describe how to integrate and use the *ABCE* layer along the main use-cases. That is, in Section 3.2.1 we provide an overview of the setup-functionalities that are provided by the *ABCE* layer. Section 3.2.2 is devoted to the presentation of tokens, thereby describing the steps that a User and Verifier have to perform in order to create and to verify a presentation token. The process of the issuance of a credential is described in Section 3.2.3, and can incorporate some of the presentation steps described in the previous section. Section 3.2.4 then deals with the inspection process that can be used to reveal some previously hidden attributes, and Section 3.2.5 describes the *ABCE* functionalities in the context of revocation.

3.1 Architectural Design

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 simplified overview of our Privacy-ABC architecture is depicted in Figure 2, and a more detailed view of the architecture on the user side is shown in Figure 3.

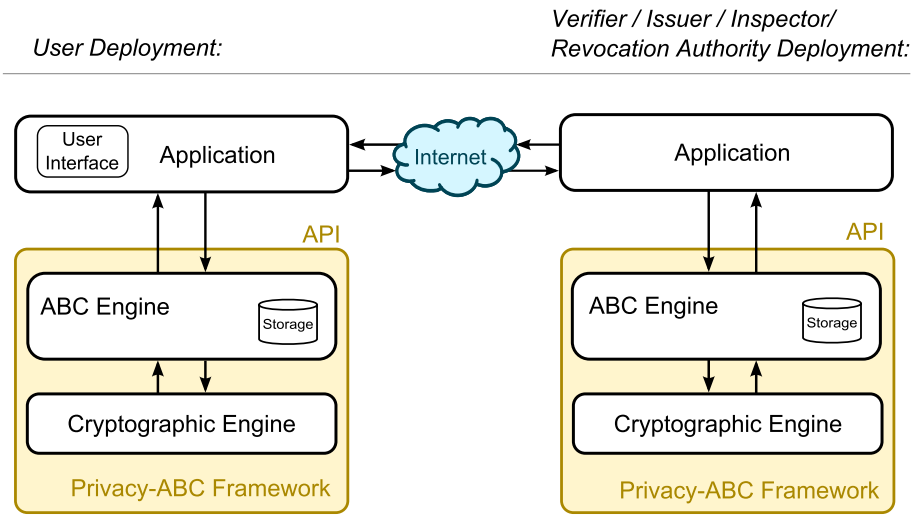

Figure 2: Architecture of a Privacy-ABC System.

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 policy enforcement point.

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.

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. The user-side components listed below are also depicted in Figure 3.

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.

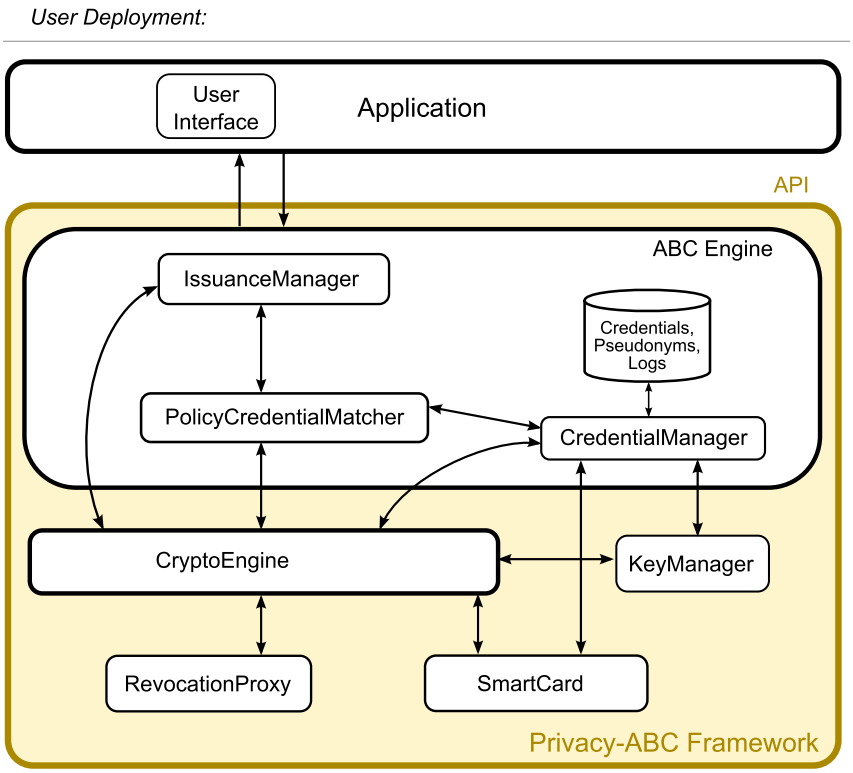

Figure 3: Overview of the Privacy-ABC Architecture on the User Side

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 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 Chapter 6.

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.

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 communicating 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. 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 memory (the so-called *blobstore*). The CryptoInterface provides cryptographic functionality for issuance, presentation, and inspection that is related to a secret stored on the card.

3.2 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. 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. A more detailed description of the API is given in Section 5, after we introduce the data formats.

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.

3.2.1 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. Note that the system parameters specify the security level that is to be 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 1. Values in brackets denote that they are optional, i.e., can also be set to *null*.

Global & Storage APIs	
`generateSystemParameters`	
input:	int *securityLevel*
output:	SystemParameters
`storeSystemParameters`	
input:	SystemParameters
output:	boolean *success*
`storeIssuerParameters`	
input:	IssuerParameters
output:	boolean *success*
`storeInspectorParameters`	
input:	InspectorParameters
output:	boolean *success*
`storeRevocationAuthorityParameters`	
input:	RevocationAuthorityParameters
output:	boolean *success*
`storeCredentialSpecification`	
input:	CredentialSpecification
output:	boolean *success*
Issuer	
`generateIssuerParameters`	
input:	URI *id*, SystemParameters, URI *technology*, int *maximalNumberOfAttributes*, [URI *revocationAuthorityId*]
output:	IssuerParameters
Inspector	
`generateInspectorParameters`	
input:	URI *id*, SystemParameters, URI *technology*
output:	InspectorParameters
Revocation Authority	
`generateRevocationAuthorityParameters`	
input:	URI *id*, SystemParameters, URI *technology*, URI *revocationInfoLocation*, URI *nonRevocationEvidenceLocation*, URI *nonRevocationUpdateLocation*
output:	RevocationAuthorityParameters
User	
`generateUserSecretKey`	
input:	SystemParameters
output:	URI *id*

Table 1: ABCE Interfaces for Setup and Storage

3.2.2 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 4 – 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 4.

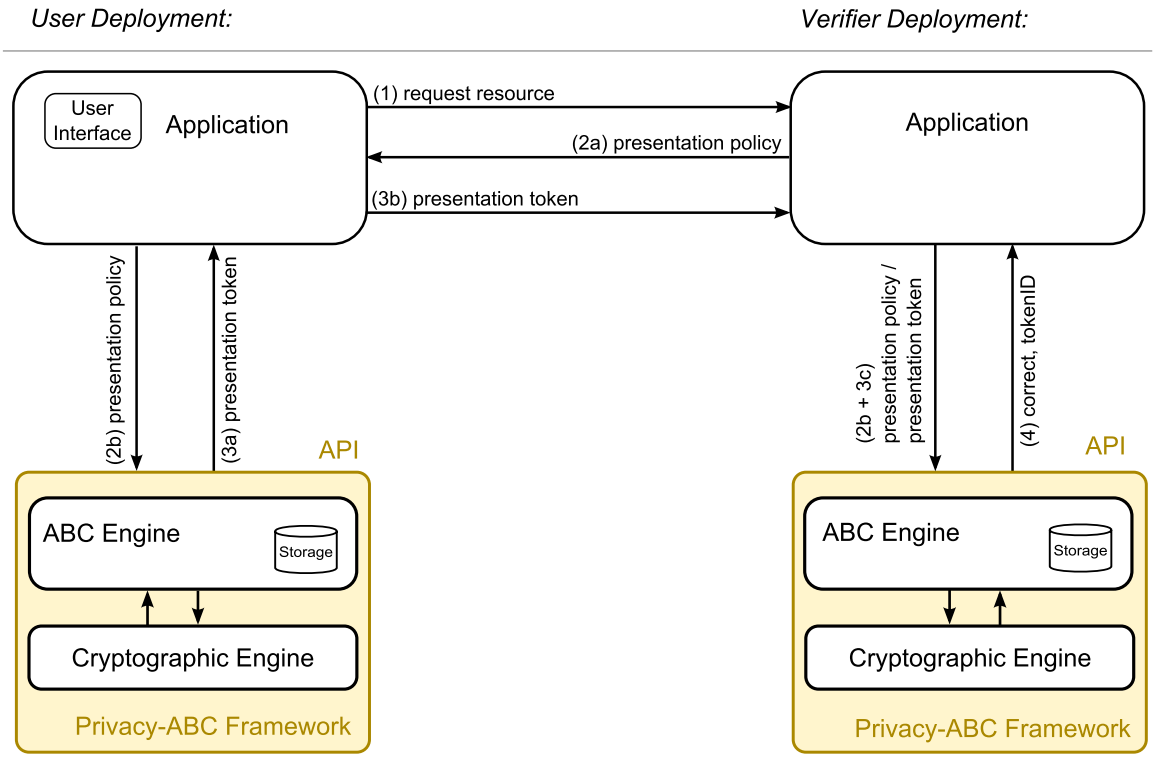

Figure 4: Presentation of a Token (Application Level)

Upon receiving the policy (Figure 4 – 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 4 – 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 4 – 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 4 – 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 4 – 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 or inspect tokens.

The ABCE interfaces available for the user and verifier in the context of generating and verifying a presentation token are summarized in Table 2.

User	
`createIdentitySelectorArguments`	
input:	PresentationPolicyAlternatives
output:	UiPresentationArguments
`createPresentationToken`	
input:	UiPresentationReturn
output:	PresentationToken
Verifier	
`verifyTokenAgainstPolicy`	
input:	PresentationToken, PresentationPolicyAlternatives, boolean *storeToken*
output:	boolean *isCorrect*, [URI *tokenId*]
`getPresentationToken`	
input:	URI *tokenId*
output:	PresentationToken

Table 2: ABCE Interfaces for Token Presentation and Verification

3.2.3 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 5 – Step 1) and indicates the credential type she wishes to obtain (Figure 5 – 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 2.6, 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.

3.2.3.1 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 5 – 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 5 – 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

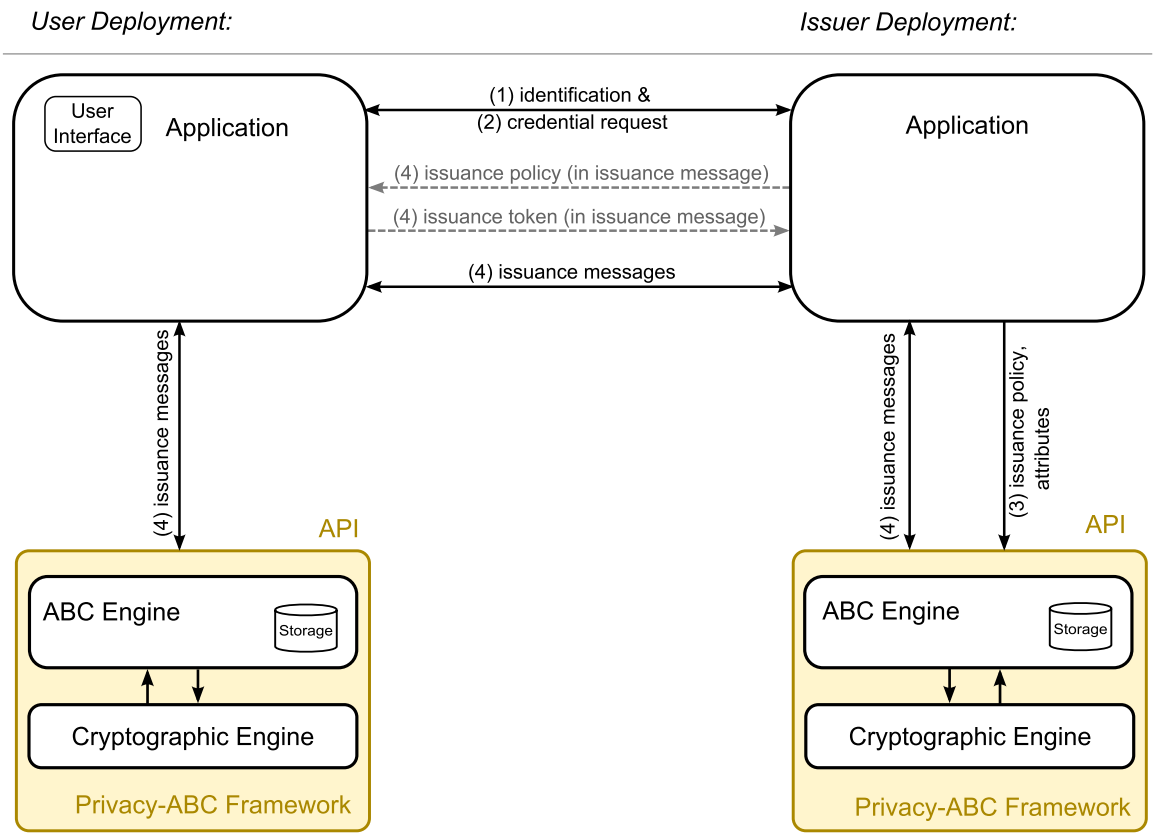

Figure 5: Issuance of a Credential (Application Level)

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.

3.2.3.2 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 5 – Step 3). If the issuance policy is non-trivial, i.e., requires the user to present at least one credential or one pseudonym, then advanced issuance is performed, otherwise a simple issuance protocol is started. 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 token is wrapped in an issuance message, which the user then forwards to the issuer (Figure 5 – 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 5 – Step 4).

Whenever the user or the issuer receives an issuance message, he invokes his 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 3.

User	
`issuanceProtocolStep`	
input:	IssuanceMessage
output:	IssuanceMessage, CredentialDescription, [UiIssuanceArguments]
`issuanceProtocolStep`	
input:	UiIssuanceReturn
output:	IssuanceMessage
Issuer	
`initIssuanceProtocol`	
input:	IssuancePolicy, List<Attribute> *issuerSpecifiedAttributes*
output:	IssuanceMessage, boolean *isLastMessage*
`issuanceProtocolStep`	
input:	IssuanceMessage
output:	IssuanceMessage, boolean *isLastMessage*
`extractIssuanceToken`	
input:	IssuanceMessage
output:	IssuanceToken

Table 3: ABCE Interfaces for Credential Issuance

3.2.4 Inspection

As described in detail in Section 2.5, 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) 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 4).

3.2.5 Revocation

Our framework also supports revocation of credentials, thereby distinguishing whether a credential may need to be revoked either globally (issuer-driven revocation) or for a specific context (verifier-driven revocation) (see Section 2.7 for details). To revoke a credential globally, the revocation authority calls the `revoke` method on input of the credential's revocation handle, which must be handed to the revocation authority by the issuer (see Table 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 be unusable to satisfy the presentation policy. 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.

Inspector		
	`inspect`	
	input:	PresentationToken, URI *credentialAlias*, URI *attributeType*
	output:	Attribute
	`inspect`	
	input:	IssuanceToken, URI *credentialAlias*, URI *attributeType*
	output:	Attribute
Revocation Authority		
	`revoke`	
	input:	URI *revocationAuthorityId*, List<Attribute> *toRevoke*
	output:	—

Table 4: ABCE Interfaces for Inspection and Revocation

4 ABC4Trust Protocol Specification

Given the multitude of distributed entities involved in a full-fledged Privacy-ABC system, the communication formats through which these entities interact must be fixed. Rather than profiling an existing standard format for identity management protocols such as SAML, WS-Trust, or OpenID, we felt that the many unique features of Privacy-ABCs were more suitably addressed by defining a dedicated format. In particular, existing standards do not support typical Privacy-ABC features such as pseudonyms, inspection, privacy-enhanced revocation, or advanced issuance protocols. In Chapter 9, we discuss how our Privacy-ABC infrastructure could be integrated with a number of existing frameworks.

This chapter provides the specification for data artifacts exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials for use in the ABC4Trust project. Our specification separates the mechanism-independent information conveyed by the artifacts from the opaque mechanism-specific cryptographic data. This specification only defines the format for the mechanism-independent information. It provides anchor points for where instantiating technologies, in particular, U-Prove and Identity Mixer, can insert mechanism-specific data, but does not fix standard formats for this data.

For the specification we use XML notation in the spirit of XML Schema, but refrain from providing a full-fledged XML Schema specification within this document for the sake of readability; we do, however, make available a separate XML schema file for the artifacts defined here[3]. Although the artifacts are defined in XML, one could create a profile using a different encoding (ASN.1, JSON, etc.) See the corresponding schema file for more details.

We start in Section 4.1 with introducing the terminology and notation used throughout this chapter. Section 4.2 then provides the artifacts for the setup of the different Privacy-ABC entities, which includes e.g., the description of the credential type and the public parameters of an Issuer and Inspector. In Section 4.3 the specifications for all artifacts related to revocation are given. For the presentation of a token, the corresponding specifications of a presentation policy and a presentation token are introduced in Section 4.4. Section 4.5 is then dedicated to the Issuance of a credential and provides artifacts for the issuance policy and issuance token. Section 4.6 introduces the data formats that are sent to and expected from (graphical) user interfaces. Finally, Section 4.7 describes some additional XML schemas used by the web services API.

4.1 Terminology and Notation

4.1.1 Notational Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, and “MAY” in this document are to be interpreted as described in [Bra97].

This specification uses the following syntax to define outlines for XML data:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  “?” (0 or 1)
  “*” (0 or more)
  “+” (1 or more)
- The character “|” is used to indicate a choice between elements.
- The characters “(“ and “)” are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- XML namespace prefixes (see Table 5) are used to indicate the namespace of the element being defined.

[3] Available under `https://abc4trust.eu/download/xml/ABC4Trust_schema_H2.1.xsd`

- XML elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions.

4.1.2 Namespaces

The base XML namespace URI used by the definitions in this document is as follows:

Table 5: XML namespaces

Prefix	XML namespace	Specification
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema2]
abc	`http://abc4trust.eu/wp2/abcschemav1.0`	This document

4.2 Setup

4.2.1 Credential Specification

The credential specification describes the contents of the credentials. It can be created by the issuer or by any external authority so that multiple issuers can issue credentials of the same specification. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a XML-signed document or provided as part of some metadata retrievable from a trusted source.

```
<abc:CredentialSpecification Version="1.0" KeyBinding="xs:boolean"
Revocable="xs:boolean">
  <abc:SpecificationUID>xs:anyURI</abc:SpecificationUID>
  <abc:numericalId>xs:integer</abc:numericalId>?
  <abc:FriendlyCredentialName xml:lang="xs:language"/>*
  <abc:DefaultImageReference>xs:anyURI</abc:DefaultImageReference>?
  <abc:AttributeDescriptions MaxLength="xs:unsignedInt">
    <abc:AttributeDescription Type="xs:anyURI"
     DataType="xs:anyURI" Encoding="xs:anyURI">
  <abc:FriendlyAttributeName lang="xs:language">xs:string</abc:FriendlyAttributeName>*
<abc:AllowedValue>...</abc:AllowedValue>*
    </abc:AttributeDescription>*
  </abc:AttributeDescriptions>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:CredentialSpecification`

This element contains the credential specification defining the contents of issued credentials adhering to this specification.

`/abc:CredentialSpecification/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:CredentialSpecification/@KeyBinding`

This attribute indicates whether credentials adhering to this specification must be bound to a secret key. See Section 2.3 for more information on key binding.

`/abc:CredentialSpecification/@Revocable`

This attribute indicates whether credentials adhering to this specification are revocable or not. If the `Revocable` attribute is set to true, then this credential specification MUST contain a dedicated attribute for the revocation handle with attribute type `http://abc4trust.eu/wp2/abcschemav1.0/revocationhandle`. The data type and encoding mechanism for the revocation handle are defined by the cryptographic mechanism used for revocation.

The revocation handle is automatically assigned a unique value by the issuance algorithm, possibly involving a communication step with the Revocation Authority. Even though there are no syntactical restrictions imposing this, presentation policies SHOULD NOT request to reveal the value of the revocation handle, as doing so enables Verifiers to link presentations tokens generated with the same credential. If necessary, inspection can be used to only reveal the value of the revocation handle under specific circumstances.

`/abc:CredentialSpecification/abc:SpecificationUID`

This element contains a URI that uniquely identifies the credential specification.

`/abc:CredentialSpecification/abc:NumericalId`

This element contains an optional numerical identifier of the credential specification, which must be unique across the system. This numerical identifier will be included as a hidden attribute in all credentials that use this specification; this makes it unambiguous which credential specification was used to issue the credential, especially if the same issuer parameters are used with multiple credential specifications. The range of that numerical identifier is the same as that of an unsigned integer attribute. If this field is left blank, a cryptographic hash of the credential specification XML is used instead as numerical identifier.

`/abc:CredentialSpecification/abc:FriendlyCredentialName`

This optional element provides a friendly textual name for the credential. The content of this element MUST be localized in a specific language.

`/abc:CredentialSpecification/abc:FriendlyCredentialName/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyCredentialName` element have been localized.

`/abc:CredentialSpecification/abc:DefaultImageReference`

This optional element contains a reference where the default image can be obtained for credentials issued according to this credential specification.

When implementing a Privacy-ABC system, downloading images from the identity providers should be handled carefully. The reference to the external image resource must not be used every time the credential is presented. To avoid linkability when using the credential, the corresponding image must be downloaded and stored locally at the user's side during the issuance.

`/abc:CredentialSpecification/abc:AttributeDescriptions`

This element contains the descriptions of the attributes issued using this specification, encoded in order in the n child elements. It is empty if n=0, i.e., if `abc:AttributeDescriptions` has no child elements.

`.../abc:AttributeDescriptions/abc:AttributeDescription`

This element contains the description of one credential attribute.

`.../abc:AttributeDescriptions/abc:AttributeDescription/@MaxLength`

This attribute specifies the maximal length in bits of the integers to which attribute values are mapped using the encoding function. The keylength of any Issuer Parameters used to issue credentials adhering to this credential specification must be large enough so that attributes of the bitlength specified here can be supported. It is up to each specific credential mechanism to describe which keylength supports which attribute bitlength.

`.../abc:AttributeDescriptions/abc:AttributeDescription/@Type`

This attribute contains the unique identifier of an attribute type encoded in credentials adhering to this specification. The attribute type is a URI, to which a semantic is associated by the definition of the attribute type. The definition of attribute types is outside the scope of this document; we refer to Section 7.5 in [Sta09a] for examples. The attribute type (e.g., `http://example.com/firstname`) is *not* to be confused with the data type (e.g., `xs:string`) that is specified by the `DataType` attribute.

`.../abc:AttributeDescriptions/abc:AttributeDescription/@DataType`

This attribute contains the data type of the credential attribute. The supported attribute data types are the following subset of XML Schema data types. We refer to the XML Schema specification (http://www.w3.org/TR/xmlschema-2) for more information on these data types.

- `http://www.w3.org/2001/XMLSchema#string`
- `http://www.w3.org/2001/XMLSchema#anyURI`
- `http://www.w3.org/2001/XMLSchema#date`
- `http://www.w3.org/2001/XMLSchema#time`
- `http://www.w3.org/2001/XMLSchema#dateTime`
- `http://www.w3.org/2001/XMLSchema#integer`
- `http://www.w3.org/2001/XMLSchema#boolean`

When specifying values for attributes of these types, the following additional restrictions must be adhered to:

- Values of type `xs:date` MUST NOT contain a timezone
- Values of type `xs:time` MUST NOT contain a timezone
- Values of type `xs:dateTime` MUST contain a timezone

`.../abc:AttributeDescriptions/abc:AttributeDescription/@Encoding`

To be embedded in a Privacy-ABC, credential attribute values must typically be mapped to integers of a fixed length indicated by the `AttributeDescription/@MaxLength` attribute. The `Encoding` XML attribute specifies how the value of this credential attribute is mapped to such an integer.

Each data type has one or more possible encoding algorithms. The encoding used may influence which values can be encoded, whether inspection can be used for this attribute, and which predicates can be proved over the attribute values (see Section 4.4.1). In order to apply a predicate over multiple credential attributes, the credential attributes MUST have the same encoding.

The following is a list of supported encodings and their respective properties. Recommendations for typical usage are included as comments.

- Encoding: `urn:abc4trust:1.0:encoding:string:sha-256`
  Data type: `http://www.w3.org/2001/XMLSchema#string`
  Restrictions: none
  Inspectable: no (hash value only)
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:string-equal`
  `urn:abc4trust:1.0:function:string-not-equal`
  Comments: Best suited for strings of arbitrary lengths that are unlikely to be used for inspection.
- Encoding: `urn:abc4trust:1.0:encoding:string:utf-8`
  Data type: `http://www.w3.org/2001/XMLSchema#string`
  Restrictions: the UTF-8 encoded string must be shorter than `@MaxLength` — 8 bits or `@MaxLength`/8 — 1 bytes
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:string-equal`
  `urn:abc4trust:1.0:function:string-not-equal`
  Comments: Best suited for short strings where the possibility to use inspection should be kept open. For long strings that are likely to require inspection, please consider splitting up the attribute into multiple attributes with this encoding.
- Encoding: `urn:abc4trust:1.0:encoding:string:prime`
  Data type: `http://www.w3.org/2001/XMLSchema#string`
  Restrictions: Can only be used for attributes where the value range is restricted by a list of

.../abc:AttributeDescription/abc:AllowedValueelements.
Inspectable: yes
Supported predicates:
urn:oasis:names:tc:xacml:1.0:function:string-equal
urn:abc4trust:1.0:function:string-not-equal
urn:abc4trust:1.0:function:string-equal-one-of
Comments: Best choice for attributes with a limited value range where presentation policies are likely to request showing that the attribute value is one of a given list of strings without revealing the exact value.

- Encoding: urn:abc4trust:1.0:encoding:anyUri:sha-256
  Data type: http://www.w3.org/2001/XMLSchema#anyURI
  Restrictions: none
  Inspectable: no (hash value only)
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
  urn:abc4trust:1.0:function:anyURI-not-equal
  Comments: Best suited for URIs of arbitrary lengths that are unlikely to be used for inspection.
- Encoding: urn:abc4trust:1.0:encoding:anyUri:utf-8
  Data type: http://www.w3.org/2001/XMLSchema#anyURI
  Restrictions: shorter than @MaxLength bytes
  Inspectable: yes
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
  urn:abc4trust:1.0:function:anyURI-not-equal
  Comments: Best suited for short URIs where the possibility to use inspection should be kept open. For long URIs that are likely to require inspection, please consider splitting up the attribute into multiple attributes with this encoding.
- Encoding: urn:abc4trust:1.0:encoding:anyURI:prime
  Data type: http://www.w3.org/2001/XMLSchema#string
  Restrictions: Can only be used for attributes where the value range is restricted by a list of .../abc:AttributeDescription/abc:AllowedValue elements.
  Inspectable: yes
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
  urn:abc4trust:1.0:function:anyURI-not-equal
  urn:abc4trust:1.0:function:anyURI-equal-one-of
  Comments: Best choice for attributes with a limited value range where presentation policies are likely to request showing that the attribute value is one of a given list of URIs without revealing the exact value.
- Encoding: urn:abc4trust:1.0:encoding:dateTime:unix:signed
  Data type: http://www.w3.org/2001/XMLSchema#dateTime
  Restrictions: none
  Inspectable: yes
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
  urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than
  urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal
  urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than
  urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal
  urn:abc4trust:1.0:function:dateTime-not-equal
  Comments: Good default choice for times that can be far in the past and/or future. Greater-than and less-than predicates may be slightly less efficient using this encoding.

- Encoding: `urn:abc4trust:1.0:encoding:dateTime:unix:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#dateTime`
  Restrictions: since 1970
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-equal`
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than`
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal`
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than`
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal`
  `urn:abc4trust:1.0:function:dateTime-not-equal`
  Comments: Best choice for times after 1970 that are likely to be used in combination with greather-than or less-than predicates.
- Encoding: `urn:abc4trust:1.0:encoding:dateTime:prime`
  Data type: `http://www.w3.org/2001/XMLSchema#dateTime`
  Restrictions: Can only be used for attributes where the value range is restricted by a list of `.../abc:AttributeDescription/abc:AllowedValue` elements.
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:dateTime-equal`
  `urn:abc4trust:1.0:function:dateTime-not-equal`
  `urn:abc4trust:1.0:function:dateTime-equal-one of`
  Comments: Best choice for attributes with a limited value range where presentation policies are likely to request showing that the attribute value is one of a given list of times without revealing the exact value.
- Encoding:`urn:abc4trust:1.0:encoding:date:unix:signed`
  Data type: `http://www.w3.org/2001/XMLSchema#date`
  Restrictions: none
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:date-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal`
  `urn:abc4trust:1.0:function:date-not-equal`
  Comments: Good default choice for dates that can be far in the past and/or future. Greater-than and less-than predicates may be less efficient using this encoding.
- Encoding: `urn:abc4trust:1.0:encoding:date:unix:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#date`
  Restrictions: since 1970
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:date-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal`
  `urn:abc4trust:1.0:function:date-not-equal`
  Comments: Best choice for times after 1970 that are likely to be used in combination with greather-than or less-than predicates.

- Encoding: `urn:abc4trust:1.0:encoding:date:since1870:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#date`
  Restrictions: since 1870
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:date-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal`
  `urn:abc4trust:1.0:function:date-not-equal`
  Comments: Best choice for birth dates, which are likely to fall after 1870 but are likely to require efficient greather-than or less-than predicates.
- Encoding:`urn:abc4trust:1.0:encoding:date:since2010:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#date`
  Restrictions: since 2010
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:date-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than`
  `urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal`
  `urn:abc4trust:1.0:function:date-not-equal`
  Comments: Best choice for expiration dates, which are likely to fall after 2010 but are likely to require efficient greather-than or less-than predicates.
- Encoding: `urn:abc4trust:1.0:encoding:date:prime`
  Data type:`http://www.w3.org/2001/XMLSchema#date`
  Restrictions: Can only be used for attributes where the value range is restricted by a list of `.../abc:AttributeDescription/abc:AllowedValue` elements.
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:date-equal`
  `urn:abc4trust:1.0:function:date-not-equal`
  `urn:abc4trust:1.0:function:date-equal-one of`
  Comments: Best choice for attributes with a limited value range where presentation policies are likely to request showing that the attribute value is one of a given list of dates without revealing the exact value.
- Encoding: `urn:abc4trust:1.0:encoding:boolean:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#boolean`
  Restrictions: none
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:boolean-equal`
  `urn:abc4trust:1.0:function:boolean-not-equal`
- Encoding: `urn:abc4trust:1.0:encoding:integer:unsigned`
  Data type: `http://www.w3.org/2001/XMLSchema#integer`
  Restrictions: positive (including zero)
  Inspectable: yes
  Supported predicates:
  `urn:oasis:names:tc:xacml:1.0:function:integer-equal`
  `urn:oasis:names:tc:xacml:1.0:function:integer-greater-than`

urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:integer-less-than
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
urn:abc4trust:1.0:function:integer-not-equal
Comments: Best for integers that cannot take negative values.

- Encoding: urn:abc4trust:1.0:encoding:integer:signed
  Data type: http://www.w3.org/2001/XMLSchema#integer
  Restrictions: none
  Inspectable: yes
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:integer-equal
  urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
  urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
  urn:oasis:names:tc:xacml:1.0:function:integer-less-than
  urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
  urn:abc4trust:1.0:function:integer-not-equal
  Comments: Best choice for integers that can have positive or negative values.
- Encoding: urn:abc4trust:1.0:encoding:integer:prime
  Data type: http://www.w3.org/2001/XMLSchema#integer
  Restrictions: Can only be used for attributes where the value range is restricted by a list of .../abc:AttributeDescription/abc:AllowedValue elements.
  Inspectable: yes
  Supported predicates:
  urn:oasis:names:tc:xacml:1.0:function:integer-equal
  urn:abc4trust:1.0:function:integer-not-equal
  urn:abc4trust:1.0:function:integer-equal-one of
  Comments: Best choice for attributes with a limited value range where presentation policies are likely to request showing that the attribute value is one of a given list of integers without revealing the exact value.

.../abc:AttributeDescriptions/abc:AttributeDescription/abc:FriendlyAttributeName

This optional element provides a friendly textual name for the attribute in the credential. The content of this element MUST be localized in a specific language.

.../abc:AttributeDescriptions/abc:AttributeDescription/abc:FriendlyAttributeName/@xml:lang

A required language identifier, using the language codes specified in [Alv01], in which the content of abc:FriendlyAttributeName element have been localized.

.../abc:AttributeDescriptions/abc:AttributeDescription/abc:AllowedValue

When present, a list of AllowedValue elements restricts the range of the value of this credential attribute to the specified list of values. Each AllowedValue element contains one possible value of the credential attribute. If abc:AttributeDescription contains one or more abc:AllowedValue elements, the actual value of the attribute of an issued credential MUST be from the specified set of allowed values. The contents of the abc:AllowedValue elements MUST be of the data type specified by the abc:AttributeDescription/@DataType attribute of the parent abc:AttributeDescription element.

4.2.2 System Parameters

In order for multiple issuers to agree on the cryptographic parameters to use throughout the system, all entities in the system must agree on one set of system parameters. These parameters have to be generated once, before any of the issuer parameters and other keys are generated. How this artifact is protected (authenticated) is application specific.

```
<abc:SystemParameters Version="1.0" SystemParametersUID="xs:anyURI">
  <abc:CryptoParams>...</abc:CryptoParams>
</abc:SystemParameters>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:SystemParameters`

This element contains the system parameters.

`/abc:SystemParameters/@Version`

The version attribute has been made optional. It will be removed but currently it is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:SystemParameters/@SystemParametersUID`

This element contains a URI that uniquely identifies the system parameters. This field will be required in future versions.

`/abc:SystemParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters to which all issuer, verifiers, users, revocation authorities, and inspectors need to perform any of the cryptographic tasks. The content of this element is defined in an external profile.

4.2.3 Issuer Parameters

In order to issue credentials, the issuer must specify system parameters, and generate a key pair consisting of a secret issuing key and a public verification key. The issuer publishes its public parameters using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a certificate signed by a certification authority, or could be provided as part of some metadata retrievable from a trusted source.

Note that one set of issuer parameters can be used to issue credentials according to several different credential specifications.

```
<abc:IssuerParameters Version="1.0">
  <abc:ParametersUID>xs:anyURI</abc:ParametersUID>
  <abc:FriendlyIssuerDescription lang="xs:language">
    xs:string
  </abc:FriendlyIssuerDescription>*
  <abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
  <abc:SystemParametersUID>xs:anyURI</abc:SystemParametersUID>
  <abc:MaximalNumberOfAttributes>xs:int</abc:MaxNumberOfAttributes>
  <abc:HashAlgorithm>xs:anyUID</abc:HashAlgorithm>
  <abc:CryptoParams>...</abc:CryptoParams>
  <abc:RevocationParametersUID>...</abc:RevocationParametersUID>?
</abc:IssuerParameters>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuerParameters`

This element contains an issuer's public parameters.

`/abc:IssuerParameters/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:IssuerParameters/abc:ParametersUID`

This element contains a URI that uniquely identifies the public issuer parameters.

`/abc:IssuerParameters/abc:FriendlyIssuerDescription`

This optional element provides a friendly textual description of the issuer. The content of this element MUST be localized in a specific language.

`/abc:IssuerParameters/abc:FriendlyIssuerDescription/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyIssuerDescription` element have been localized.

`/abc:IssuerParameters/abc:AlgorithmID`

This element identifies the algorithm of the public issuer parameters. The algorithm URIs `urn:abc4trust:1.0:algorithm:idemix` for Identity Mixer and `urn:abc4trust:1.0:algorithm:uprove` for U-Prove MUST be supported; other algorithms MAY be supported.

`/abc:IssuerParameters/abc:SystemParametersUID`

This element identifies the system parameters that are to be used with these issuer parameters.

`/abc:IssuerParameters/abc:MaximalNumberOfAttributes`

One set of issuer parameters can be used to issue credentials adhering to multiple credential specifications. This element specifies the maximum number of attributes for such credentials. The number of attributes in a credential is fixed by credential specification. For revocable credentials, the revocation handle counts towards the maximum number of attributes.

`/abc:IssuerParameters/abc:HashAlgorithm`

This element specifies the hash algorithm that is to be used in the generation of the presentation tokens derived from credentials issued under these parameters. This hash algorithm is not to be confused with the encoding algorithm that maps attribute values to integers and may also specify a hash function to apply to long attribute values. The hash algorithm SHA-256 with identifier `urn:abc4trust:1.0:hashalgorithm:sha-256` MUST be supported; other algorithms MAY be supported.

`/abc:IssuerParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to issue, use, and verify credentials. The content of this element is defined in an external profile based on the value of the abc:AlgorithmID element.

`/abc:IssuerParameters/abc:RevocationParametersUID`

This optional element contains the parameters identifier of a revocation authority that is responsible for revoking credentials issued under these issuer parameters. The parameters referred to here are determined by the issuer (i.e., issuer-driven revocation), meaning that any presentation token involving credentials issued under these issuer parameters MUST be checked against the latest revocation information associated to the revocation parameters referenced by this element.

4.2.4 Inspector Public Key

In order to decrypt encrypted attributes, an inspector must generate a key pair consisting of a secret decryption key and a public encryption key. The inspector publishes its public key using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a certificate signed by a certification authority, or could be provided as part of some metadata retrievable from a trusted source.

```
<abc:InspectorPublicKey Version="1.0">
  <abc:PublicKeyUID>xs:anyURI</abc:PublicKeyUID>
  <abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
  <abc:FriendlyInspectorDescription lang="xs:language">
   xs:string
  </abc:FriendlyInspectorDescription>*
  <abc:CryptoParams>...</abc:CryptoParams>
</abc:InspectorPublicKey>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:InspectorPublicKey`

This element contains an inspector's public key.

`/abc:InspectorPublicKey/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:InspectorPublicKey/abc:PublicKeyUID`

This element contains a URI that uniquely identifies the public key.

`/abc:InspectorPublicKey/abc:AlgorithmID`

This element identifies the algorithm of the public key. The Camenisch-Shoup inspection algorithm [CS03b] with identifier `urn:abc4trust:1.0:inspectionalgorithm:camenisch-shoup03` MUST be supported; other algorithms MAY be supported.

`/abc:InspectorPublicKey/abc:FriendlyInspectorDescription`

This optional element provides a friendly textual description for the inspector's public key. The content of this element MUST be localized in a specific language.

`/abc:InspectorPublicKey/abc:FriendlyInspectorDescription/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyInspectorDescription` element have been localized.

`/abc:InspectorPublicKey/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to issue, use, and verify credentials. The content of this element is defined in an external profile based on the value of the `abc:AlgorithmID` element.

4.2.5 Verifier Parameters

In order for the verifier to communicate to the user which cryptographic algorithms he supports, and provide additional parameters for these algorithms, the verifier must generate a set of verifier parameters and send them to the user. How this artifact is protected (authenticated) is application specific.

```
<abc:VerifierParameters Version="1.0" VerifierParametersId="xs:anyURI" SystemParametersId="xs:anyURI">
  <abc:CryptoParams>...</abc:CryptoParams>
</abc:VerifierParameters>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:VerifierParameters`

This element contains the verifier parameters.

`/abc:VerifierParameters/@Version`

The version attribute has been made optional. It will be removed but currently it is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:VerifierParameters/@VerifierParametersId`

This element contains a URI that uniquely identifies these verifier parameters.

`/abc:VerifierParameters/@SystemParametersId`

This element contains a URI that references the system parameters that are to be used with these verifier parameters.

`/abc:VerifierParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters which the user needs to determine which cryptographic algorithms are supported by the verifier, and which cryptographic parameters must be used in those algorithms. The content of this element is defined in an external profile.

4.3 Revocation

A Revocation Authority maintains information about valid and, in particular, revoked credentials. To do so, it first generates public parameters and possibly corresponding secret parameters. It publishes its public parameters together with a description of the particular revocation method that is used and a reference to the location where the most current revocation information will be published.

Some revocation mechanisms, such as the implemented accumulator scheme, require users to obtain an additional piece of information called non-revocation evidence in order to be able to prove that their credential is still valid.

The different revocation mechanisms vary quite strongly in how the non-revocation evidence is created and maintained. Depending on the specific mechanism, the non-revocation evidence

- may be the same for all users, or may be different for each user and/or each issued credential;
- may be sensitive information that the user needs to keep strictly secret, or may be leaked to other participants without further harm;
- may be first created during the issuance of the credential, during the first usage (presentation) of the credential, or at any time between issuance and first usage;
- may have to be kept up-to-date with the non-revocation information, or may remain the same for the lifetime of the credential.

The Revocation Authority can also include references to the locations where the users can obtain the information to create and to update their non-revocation evidence. Both the initialization of the non-revocation evidence and the update may be multi-leg cryptographic protocols.

4.3.1 Revocation Authority Parameters

Each Revocation Authority generates and publishes its parameters at setup. The parameters are static, i.e., they do not change over time as more credentials are revoked.

```
<abc:RevocationAuthorityParameters Version="1.0">
  <abc:ParametersUID>xs:anyURI</abc:ParametersUID>
  <abc:RevocationMechanism>xs:anyURI</abc:RevocationMechanism>
  <abc:RevocationInfoReference ReferenceType="xs:anyURI">
    ...
</abc:RevocationInfoReference>?
  <abc:NonRevocationEvidenceReference ReferenceType="xs:anyURI">
    ...
</abc:NonRevocationEvidenceReference>?
  <abc:NonRevocationEvidenceUpdateReference ReferenceType="xs:anyURI">
    ...
</abc:NonRevocationEvidenceUpdateReference>?
<abc:CryptoParams>...</CryptoParams>?
</abc:RevocationAuthorityParameters>
```

`/abc:RevocationAuthorityParameters`

This element contains the public parameters of the Revocation Authority

`/abc:RevocationAuthorityParameters/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:RevocationAuthorityParameters/abc:ParametersUID`

This element contains a unique identifier for these Revocation Authority parameters.

`/abc:RevocationAuthorityParameters/RevocationMechanism`

This attribute indicates the mechanism or algorithm used to revoke credentials. The revocation mechanisms `urn:abc4trust:1.0:algorithm:uprove` and `urn:abc4trust:1.0:algorithm:idemix` MUST be supported; other revocation mechanisms MAY be supported.

`/abc:RevocationAuthorityParameters/abc:RevocationInfoReference`

This optional element contains a reference to the endpoint where the most current public revocation information corresponding to these parameters can be obtained.

`/abc:RevocationAuthorityParameters/abc:NonRevocationEvidenceReference`

This optional element contains a reference to the endpoint with the information about how to obtain the (possibly private) user-specific non-revocation evidence object.

`/abc:RevocationAuthorityParameters/abc:NonRevocationEvidenceUpdateReference`

This optional element contains a reference to the endpoint the most current information for updating the non-revocation evidence can be obtained.

`/abc:RevocationAuthorityParameters/abc:RevocationInfoReference/@ReferenceType`

This attribute indicates the type of reference to the revocation information endpoint.

`/abc:RevocationAuthorityParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters that are needed to verify the Revocation Information. The content of this element is defined in an external profile based on the value of the `abc:RevocationMechanism` element.

4.3.2 Revocation Information

A Revocation Authority regularly publishes the most recent revocation information, allowing Users to prove and Verifiers to ensure that the credentials used to generate a presentation token have not been revoked. Contrary to the Revocation Authority parameters, the revocation information changes over time, e.g., at regular time intervals, or whenever a new credential is revoked.

The Revocation Authority publishes the revocation information using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a XML-signed document or provided as part of some metadata retrievable from a trusted source.

```
<abc:RevocationInformation Version="1.0">
  <abc:RevocationInformationUID>xs:anyURI</abc:InformationUID>
<abc:RevocationAuthorityParametersUID>
  xs:anyURI
  </abc:RevocationAuthorityParametersUID>
<abc:Created>xs:dateTime</abc:Created>?
<abc:Expires>xs:dateTime</abc:Expires>?
  <abc:CryptoParams>...</abc:CryptoParams>
</abc:RevocationInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:RevocationInformation`

This element contains the current revocation information, as published by the Revocation Authority. At each update of the revocation information, a new `abc:RevocationInformation` element is generated.

`/abc:RevocationInformation/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:RevocationInformation/abc:RevocationInformationUID`

This element contains the unique identifier of the revocation information. This identifier is different for each version of the revocation information, i.e., a new URI is used at every update.

`/abc:RevocationInformation/abc:RevocationAuthorityUID`

This element contains the identifier of the parameters of the revocation authority that published the revocation information.

`/abc:RevocationInformation/abc:Created`

This optional element contains the date and time when the revocation information was updated or first published.

`/abc:RevocationInformation/abc:Expires`

This optional element contains the date and time until when the revocation information is valid.

`/abc:IssuerParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to verify whether a credential is still valid. (The content of this element is defined in an external profile based on the value of the `@RevocationMechanism` attribute specified in the referenced `abc:RevocationAuthorityParameters` element)

4.3.3 Non-Revocation Evidence

The exact details of how and when the non-revocation evidence is created and updated vary greatly among the different revocation mechanisms. We therefore simply define an artifact that acts as a wrapper for a message in a (possibly multi-legged) evidence creation or update protocol. These messages are sent to and received as a response from the evidence creation and update endpoints specified in the Revocation Authority parameters.

```
<abc:RevocationMessage Context="...">
<abc:RevocationAuthorityParametersUID>
  xs:anyURI
  </abc:RevocationAuthorityParametersUID>
  <abc:RevocationMessageType>
  xs:string
  </abc:RevocationMessageType>
  <abc:CryptoParams>...</abc:CryptoParams>
</abc:RevocationMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:RevocationMessage/@Context`

This attribute contains a unique identifier for this protocol session, so that the different flows in the protocol session can be linked together. The request MUST contain a Context attribute. The revocation authority MUST reject requests with context values already in use.

`/abc:RevocationMessage/abc:RevocationAuthorityParametersUID`

This element contains the identifier of the parameters of the revocation authority that creates the non-revocation evidence information.

`/abc:RevocationMessage/abc:RevocationMessageType`

This element serves to communicate the purpose of this message to the communication endpoint. For example, the verifier may specify to retrieve the most recent revocation information using this field.

`/abc:RevocationMessage/abc:CryptoParams`

This element describes the mechanism-specific (cryptographic) parameters needed to obtain the non-revocation evidence information for building or updating the evidence.

4.4 Presentation

The user agent can create presentation tokens using one or more credentials in its possession. The verifier can optionally insist that all credentials used to generate the token are bound to the same user (i.e., to the same user secret) or device.

In a typical ABC presentation interaction, the user first requests access to a protected resource, upon which the verifier sends a presentation policy that describes which credentials the user should present to obtain access. The user agent then checks whether it has the necessary credentials to satisfy the verifier's presentation policy, and if so, generates a presentation token containing the appropriate cryptographic evidence.

Upon receiving the presentation token, the verifier checks that the cryptographic evidence is valid for the presented credentials and checks that the token satisfies the presentation policy. If both tests succeed, it grants access to the resource.

4.4.1 Presentation Policy

The verifier's policy describes the class of presentation tokens that it will accept. It is expressed by means of a `abc:PresentationPolicyAlternatives` element, with the following schema:

```
<abc:PresentationPolicyAlternatives Version="1.0">
<abc:PresentationPolicy PolicyUID="xs:anyURI"?>
    <abc:Message>
      <abc:Nonce>...</abc:Nonce>?
      <abc:FriendlyPolicyName lang="xs:language">
       xs:string
      </abc:FriendlyPolicyName>*
      <abc:FriendlyPolicyDescription lang="xs:language">
       xs:string
      </abc:FriendlyPolicyDescription>*
  <abc:VerifierIdentity>xs:any</abc:VerifierIdentity>?
      <abc:ApplicationData>...</abc:ApplicationData>?
    </abc:Message>?
    <abc:Pseudonym Exclusive="xs:boolean"? Scope="xs:string"
     Established="xs:boolean"? Alias="xs:anyURI"?
     SameKeyBindingAs="xs:anyURI"?>
      <abc:PseudonymValue> </abc:PseudonymValue>?
    </abc:Pseudonym>*
    <abc:Credential Alias="xs:anyURI"? SameKeyBindingAs="xs:anyURI"?>
      <abc:CredentialSpecAlternatives>
        <abc:CredentialSpecUID>...</abc:CredentialSpecUID>+
      </abc:CredentialSpecAlternatives>
      <abc:IssuerAlternatives>
        <abc:IssuerParametersUID
         RevocationInformationUID="xs:anyURI"?>
          ...
        </abc:IssuerParametersUID>+
      </abc:IssuerAlternatives>
      <abc:DisclosedAttribute AttributeType="xs:anyURI"
         DataHandlingPolicy="xs:anyURI"?>
        ( <abc:InspectorAlternatives>
       <abc:InspectorPublicKeyUID>...</abc:InspectorPublicKeyUID>+
      </abc:InspectorAlternatives>
          <abc:InspectionGrounds>...</abc:InspectionGrounds>
        )?
        </abc:DisclosedAttribute>*
    </abc:Credential>*
    <abc:VerifierDrivenRevocation>
       <abc:RevocationParametersUID>...</abc:RevocationParametersUID>
       <abc:Attribute CredentialAlias="xs:anyURI"
                      AttributeType="xs:anyURI">+
    </abc:VerifierDrivenRevocation>*
    <abc:AttributePredicate Function="xs:anyURI">
    ( <abc:Attribute CredentialAlias="xs:anyURI"
       AttributeType="xs:anyURI" DataHandlingPolicy="xs:anyURI"?/>
      |
       <abc:ConstantValue>...</abc:ConstantValue>
    )+
    </abc:AttributePredicate>*
  </abc:PresentationPolicy>+
</abc:PresentationPolicyAlternatives>
```

The following describes the attributes and elements listed in the schema outlined above: `/abc:PresentationPolicyAlternatives`

This element contains a presentation policy, which may contain multiple policy alternatives as child elements. The presented token must satisfy at least one of the specified policies.

`/abc:PresentationPolicyAlternatives/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy`

This element contains one policy alternative.

`.../abc:PresentationPolicy/@PolicyUID`

This attribute assigns a unique identifier to this presentation policy that can be referenced from presentation tokens that satisfy the policy.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Message`

This optional element specifies a message to be authenticated (signed) by the private key of each credential in the token.

`.../abc:PresentationPolicy/abc:Message/abc:Nonce`

This optional element contains a random nonce.

`.../abc:PresentationPolicy/abc:Message/abc:FriendlyPolicyName`

This optional element provides a friendly textual name for the policy. The content of this element MUST be localized in a specific language

`.../abc:PresentationPolicy/abc:Message/abc:FriendlyPolicyName/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyPolicyName` element have been localized.

`.../abc:PresentationPolicy/abc:Message/abc:FriendlyPolicyDescription`

This optional element provides a friendly textual description for the policy. The content of this element MUST be localized in a specific language.

`.../abc:PresentationPolicy/abc:Message/abc:FriendlyPolicyDescription/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyPolicyDescription` element have been localized.

`.../abc:PresentationPolicy/abc:Message/abc:VerifierIdentity`

This optional element contains the identity of the verifier (e.g., his URL, public key, or SSL certificate hash) for whom the presentation token must be constructed. The presentation token will authenticate the verifier identity, offering some protection against man-in-the-middle attacks if the user's application software can parse and verify the verifier's identity.

`.../abc:PresentationPolicy/abc:Message/abc:ApplicationData`

This optional element can contain any application-specific data. The contained data MAY be human readable, depending on the application, and displayed to the user.

`.../abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Pseudonym`

When present, this optional element indicates that a pseudonym must be presented with the presentation token. If this policy does not involve any credentials to be presented, then a verifiable pseudonym must be presented. Otherwise, a certified pseudonym associated to the presented credentials must be presented. See Section 2.4 for more information on pseudonyms.

`.../abc:PresentationPolicy/abc:Pseudonym/@Scope`

This attribute indicates a string to which the pseudonym is associated. The user agent is assumed to maintain state information to keep track of which pseudonym it previously used for which scope. There can be multiple verifiable or certified pseudonyms associated to the same scope string, but a scope-exclusive pseudonym is guaranteed to be unique with respect to the scope string and the user secret. In the former case, the scope string is merely a hint to the user agent which of its stored pseudonyms can be reused in the presentation token, or to which scope string it should associate a newly created pseudonym. In the latter case, the scope string uniquely determines the pseudonym that needs to be used. The scope string MAY encode an identifier of the verifier and/or of the requested resource. See Section 2.4 for more information on the use of pseudonyms.

`.../abc:PresentationPolicy/abc:Pseudonym/@Exclusive`

When present and set to true, this attribute indicates that a scope-exclusive pseudonym is to be presented with the token. The value of the `@Scope` attribute determines the scope with respect to which the pseudonym must be generated. See Section 2.4 for more information on scope-exclusive pseudonyms.

`.../abc:PresentationPolicy/abc:Pseudonym/@Established`

When set to true, this attribute indicates that the pseudonym to be presented by the User must re-authenticate under a pseudonym that was previously established with the Verifier. When set to false or when not present, this attribute indicates that the User may establish a new pseudonym in the presentation token.

`.../abc:PresentationPolicy/abc:Pseudonym/@Alias`

This optional attribute defines an alias for this pseudonym so that it can be referred to from other pseudonyms or credentials to enforce same key binding, or, if this presentation token is part of an issuance token, to support carrying over key binding to the newly issued credential. See the `/abc:IssuancePolicy/abc:CredentialTemplate/abc:UnknownAttributes /abc:KeyBinding/abc:PseudonymInfo/@Alias` element.

`.../abc:PresentationPolicy/abc:Pseudonym/@SameKeyBindingAs`

If present, this XML attribute 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 and the referred pseudonym or credential have to be bound to the *same key*. Insisting credentials to be bound to the same key limits users from sharing credentials.

The pseudonym or credential that is referred to does not have to refer back to this pseudonym. If the referred to pseudonym or credential also has a `SameKeyBindingAs` attribute that refers to a third pseudonym or credential, then all three pseudonyms/credentials must be bound to the same key. In other words, `SameKeyBindingAs` induces a transitive relationship.

`.../abc:PresentationPolicy/abc:Pseudonym/abc:PseudonymValue`

When present, this optional element indicates that a pseudonym with the given value must be presented, the value being encoded as content of type `xs:base64Binary`. Note that this feature only makes sense if the verifier has reason to believe that the user to whom the policy is sent knows the user secret (and, if applicable, pseudonym metadata) underlying the given pseudonym, for example, because he established the pseudonym in a previous presentation token.

`.../abc:PresentationPolicy/abc:Credential`

This optional element specifies a credential that has to be used in the generation of the token. Omitting this element may be useful, for example, when the user can obtain access by merely presenting an existing verifiable pseudonym.

`.../abc:PresentationPolicy/abc:Credential/@Alias`

This optional attribute creates an alias for this credential to refer to attributes from this credential in attribute predicates. See the `.../abc:PresentationPolicy/abc:AttributePredicates` element.

`.../abc:PresentationPolicy/abc:Credential/@SameKeyBindingAs`

If present, this XML attribute contains an alias referring either to a Pseudonym element within this policy, or to another `Credential` element for a credential with key binding. This indicates that the current credential and the referred pseudonym or credential have to be bound to the *same key*. Insisting credentials to be bound to the same key limits users from sharing credentials.

The pseudonym or credential that is referred to does not have to refer back to this credential. If the referred to pseudonym or credential also has a `SameKeyBindingAs` attribute that refers to a third pseudonym or credential, then all three pseudonyms/credentials must be bound to the same key. In other words, `SameKeyBindingAs` induces a transitive relationship.

`.../abc:PresentationPolicy/abc:Credential/abc:CredentialSpecAlternatives`

This element contains a list of credential specifications. The issued credential used to instantiate this credential in the presentation token must adhere to one of the listed credential specifications.

`.../abc:Credential/abc:CredentialSpecAlternatives/abc:CredentialSpecUID`

This element contains one credential specification identifier that can be used to instantiate this credential in the presentation token.

`.../abc:Credential/abc:IssuerAlternatives`

This element contains a list of identifiers for issuer parameters UID. The issued credential used to instantiate this credential in the presentation token must be issued under one of the listed issuer parameters.

`.../abc:Credential/abc:IssuerAlternatives/abc:IssuerParametersUID`

This element contains one issuer parameters identifier that is accepted for this credential in the presentation token.

This specification defines two dedicated values for the issuer parameters:

- The value `http://abc4trust.eu/wp2/issuerparameters/unsigned` indicates that the attribute values in this credential are self-claimed, without any form of authentication by either an external issuer or the user herself.
- The value `http://abc4trust.eu/wp2/issuerparameters/pseudonymously-self-signed` indicates that the attribute values in this credential are self-claimed and signed under the pseudonym of the user provided in the same presentation token. This value can only occur when the presentation policy contains a `/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Pseudonym` element.

`.../abc:IssuerAlternatives/abc:IssuerParametersUID/@RevocationInformationUID`

If the issuer parameters referred to in this element specify an Issuer-driven Revocation Authority, i.e., if the referred `abc:IssuerParameters` element contains an `abc:RevocationParametersUID` child element, then this optional XML attribute can indicate for which version of the revocation information the presented token must be valid. By specifying the current revocation information identifier in the presentation policy, the User does not have to get in touch with the Revocation Authority to check whether her non-revocation evidence information is still up to date, thereby avoiding a possible source of linkability.

`.../abc:PresentationPolicy/abc:Credential//abc:DisclosedAttribute`

This element specifies an attribute of this credential that has to be revealed in the presentation token, either to the verifier itself, or to an external inspector.

Even though there are no syntactical restrictions imposing this, presentation policies SHOULD NOT request to reveal the value of the revocation handle (with attribute type `http://abc4trust.eu/wp2/abcschemav1.0/revocationhandle`), as doing so enables Verifiers to link presentation tokens generated with the same credential. If necessary, inspection can be used to only reveal the value of the revocation handle under specific circumstances.

`.../abc:Credentials/abc:Credential/abc:DisclosedAttribute/@AttributeType`

This attribute specifies the type of the credential attribute of which the value must be revealed in the presentation token. If multiple credential specifications are allowed for this credential (i.e., if multiple

abc:CredentialSpecUID elements are listed in the abc:CredentialSpecAlternatives child element of the ancestor abc:Credential element), then the specified attribute type MUST occur in all listed credential specifications.

For each credential and each attribute type, there MUST be at most one abc:DisclosedAttribute element without abc:InspectorAlternatives child element. Likewise, for each credential and each attribute type, there MUST be at most one abc:DisclosedAttribute element with the same abc:InspectionGrounds child element.

.../abc:Credential/abc:DisclosedAttribute/@DataHandlingPolicy

This XML attribute can be used to refer to an external data handling policy describing how the Verifier will treat the revealed attribute value once it is received. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

.../abc:Credential/abc:DisclosedAttribute/abc:InspectorAlternatives

This optional element lists a number of inspector public key identifiers. When present, this element indicates that the value of this attribute does not have to be revealed to the verifier, but must be encrypted under one of the listed inspector public keys. See Section 2.5 for more details on revealing attributes to an inspector.

.../abc:DisclosedAttribute/abc:InspectorAlternatives/abc:InspectorPublicKeyUID

This element contains one identifier of an inspector public key under which the attribute value can be encrypted.

.../abc:Credential/abc:DisclosedAttribute/abc:InspectionGrounds

This optional element contains a string describing the valid grounds or circumstances under which the inspector can be asked to decrypt the attribute value. This element must be present whenever a sibling abc:InspectorAlternatives element is present. See Section 2.5 for more details on revealing attributes to an inspector.

.../abc:PresentationPolicy/abc:VerifierDrivenRevocation

This optional element specifies all parameters for checking if a (set of) attribute value(s) from the specified credentials was not revoked using verifier-driven revocation.

Verifier-driven revocation can be based on combinations of attributes from a set of different credentials, in which case there will be multiple abc:Attribute elements per one abc:VerifierDrivenRevocation element. Then the User has to prove that a disjunctive combination of these attribute values was not revoked with respect to the specified abc:RevocationParametersUID.

.../abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:RevocationParametersUID

This element contains the UID of the revocation authority parameters. The User needs to provide a proof that a following (set of) attribute value(s) was not revoked according to the specified set of parameters.

.../abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:Attribute

This element specifies a credential attribute that is used for verifier-driven revocation.

.../abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:Attribute/@CredentialAlias

This attribute specifies the alias of the credential from which the attribute is used. The specified value MUST also occur as an Alias attribute in an abc:Credential element within this abc:PresentationPolicy.

.../abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:Attribute/@AttributeType

This attribute refers to the attribute within the credential that is to be used for verifier driven revocation.

.../abc:PresentationPolicy/abc:AttributePredicate

This element specifies a predicate that must hold over the attribute values. To satisfy the policy, the presentation token must for each of the listed predicates either prove (in a data-minimizing way) that the

credential attributes satisfy the specified predicate, or must reveal the value of the involved attribute(s) so that the verifier can check whether the predicate is satisfied. The child elements are the ordered list of arguments of the predicate.

`.../abc:PresentationPolicy/abc:AttributePredicate/@Function`

This attribute specifies the boolean function for this predicate. See Section 4.4.3 for a list of supported functions and their implications on the list of arguments in the child elements. Note that not all predicate functions can be used for all attributes: the allowed predicate functions depend on the data type and on the chosen encoding of the credential attributes. See Section 4.2.1 for a list of which predicates can be used in combination with which data types and encodings.

`.../abc:AttributePredicate/abc:Attribute`

This element specifies a reference to a credential attribute that is to be used as an argument of the predicate.

`.../abc:AttributePredicate/abc:Attribute/@CredentialAlias`

This attribute specifies the alias of the credential from which the attribute must be used. The specified alias MUST also occur as an Alias attribute in an `abc:Credential` element within the ancestor `abc:PresentationPolicy` element.

`.../abc:AttributePredicate/abc:Attribute/@AttributeType`

This attribute refers to the attribute within the credential that is to be used as an argument in the predicate.

`.../abc:AttributePredicate/abc:Attribute/@DataHandlingPolicy`

This XML attribute can be used to refer to an external data handling policy describing how the Verifier will treat the information that the attribute value satisfies the specified predicate. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`.../abc:AttributePredicate/abc:ConstantValue`

This element contains a constant value that is to be used as an argument in the predicate. The data type of the argument depends on the function of the predicate. We refer to Section 4.2.1 for a list of supported functions and the data types of their arguments.

4.4.2 Presentation Token

The presentation of one or multiple credentials results in a presentation token that is sent to the verifier. The syntax for the element is:

```
<abc:PresentationToken Version="1.0">
 <abc:PresentationTokenDescription PolicyUID="xs:anyURI"
  TokenUID="xs:anyURI"?>
  <abc:Message>
    <abc:Nonce>...</abc:Nonce>?
     <abc:FriendlyPolicyName lang="xs:language">
      xs:string
     </abc:FriendlyPolicyName>*
     <abc:FriendlyPolicyDescription lang="xs:language">
      xs:string
     </abc:FriendlyPolicyDescription>*
     <abc:VerifierIdentity>xs:any</abc:VerifierIdentity>
    <abc:ApplicationData>...</abc:ApplicationData>?
  </abc:Message>?
  <abc:Pseudonym Scope="xs:string"? Exclusive="xs:boolean"?
   Alias="xs:anyURI"? SameKeyBindingAs="xs:anyURI"?>
      <abc:PseudonymValue>...</abc:PseudonymValue>
  </abc:Pseudonym>*
    <abc:Credential Alias="xs:anyURI"? SameKeyBindingAs="xs:anyURI"?>
      <abc:CredentialSpecUID>...</abc:CredentialSpecUID>
      <abc:IssuerParametersUID>...</abc:IssuerParametersUID>
      <abc:RevocationInformationUID>
        ...
      </abc:RevocationInformationUID>?
        <abc:DisclosedAttribute AttributeType="xs:anyURI"
         DataHandlingPolicy="xs:anyURI"?>
        ( <abc:InspectorPublicKeyUID>...</abc:InspectorPublicKeyUID>
          <abc:InspectionGrounds>...</abc:InspectionGrounds>
        )?
          <abc:AttributeValue>...</abc:AttributeValue>
      </abc:DisclosedAttribute>*
   </abc:Credential>*
   <abc:VerifierDrivenRevocation>
       <abc:RevocationInformationUID>...</abc:RevocationInformationUID>
       <abc:Attribute AttributeType="xs:anyURI"        CredentialAlias="xs:anyURI" >+
   </abc:VerifierDrivenRevocation>*
   <abc:AttributePredicate Function="xs:anyURI">
    ( <abc:Attribute CredentialAlias="xs:anyURI"
       AttributeType="xs:anyURI"
       DataHandlingPolicy="xs:anyURI"?/>
      |
      <abc:ConstantValue>...</abc:ConstantValue>
    )+
   </abc:AttributePredicate>*
  </abc:PresentationTokenDescription>
  <abc:CryptoEvidence>...</abc:CryptoEvidence>
</abc:PresentationToken>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:PresentationToken`

This element contains a presentation token.

`/abc:PresentationToken/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:PresentationTokenDescription`

This element contains a technology-agnostic description of the revealed information.

`.../abc:PresentationPolicy/@PolicyUID`

This attribute refers to the UID of the presentation policy that this token satisfies.

`.../abc:PresentationPolicy/@TokenUID`

This optional attribute assigns a unique identifier to this presentation token.

`.../abc:PresentationTokenDescription/abc:Message`

This optional element specifies a message that is authenticated (signed) by the private key of each credential in the token.

`.../abc:PresentationTokenDescription/abc:Message/abc:Nonce`

This optional element contains a random nonce that is to be signed by a presentation token satisfying this policy. The nonce is generated by the Issuer and prevents replay attacks.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyName`

This optional element provides a friendly textual name for the policy. The content of this element MUST be localized in a specific language.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyName/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyPolicyName` element have been localized.

`.../abc:PresentationTokenDescription/abc:Message/abc:VerifierIdentity`

This optional element contains the identity of the verifier (e.g., his URL, public key, or SSL certificate hash) to whom this presentation token is intended. The presentation token authenticates the verifier identity, meaning that it cannot be changed after the token was created. This can offer protection against man-in-the-middle attacks if the user's application software has a way to parse and verify the verifier's identity.

The format and verification of the verifier identity must be performed by the application logic. The ABCE does not perform any such checks.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyDescription`

This optional element provides a friendly textual description for the policy. The content of this element MUST be localized in a specific language.

`.../abc:Message/abc:FriendlyPolicyDescription/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyPolicyDescription` element have been localized.

`.../abc:PresentationTokenDescription/abc:Message/abc:ApplicationData`

This optional element can contains data of type string.

`.../abc:PresentationTokenDescription/abc:Pseudonym`

When present, this element indicates that a pseudonym is presented with the presentation token. If this policy does not involve any credentials, then this is a verifiable pseudonym, otherwise it is a certified pseudonym associated to the presented credentials. See Section 2.4 for more information on pseudonyms.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Scope`

This optional attribute indicates that the presented pseudonym is for a specific scope (e.g., a resource identifier) See Section 2.4 for more information on the use of pseudonyms. The user agent is assumed to maintain state information to keep track of which pseudonym it previously used for which scope.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Exclusive`

When present, this attribute indicates that a scope-exclusive pseudonym is presented with the token. The value of the `@Scope` attribute determines the scope with respect to which the pseudonym was generated. See Section 2.4 for more information on scope-exclusive pseudonyms.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Alias`

This optional attribute defines an alias for this pseudonym so that it can be referred to from other pseudonyms or credentials to enforce same key binding, or, if this presentation token is part of an issuance token, to support carrying over key binding to the newly issued credential. See the `/abc:IssuancePolicy/abc:CredentialTemplate/abc:UnknownAttributes/abc:KeyBinding/abc:PseudonymInfo/@Alias` element.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@SameKeyBindingAs`

If present, this XML attribute contains an alias referring either to another `Pseudonym` element within this presentation token, or to a `Credential` element for a credential with key binding. This indicates that the current pseudonym and the referred pseudonym or credential are bound to the *same key*.

The pseudonym or credential that is referred to does not have to refer back to this pseudonym. If the referred to pseudonym or credential also has a `SameKeyBindingAs` attribute that refers to a third pseudonym or credential, then all three pseudonyms/credentials are bound to the same key. In other words, `SameKeyBindingAs` induces a transitive relationship.

`.../abc:PresentationTokenDescription/abc:Pseudonym/abc:PseudonymValue`

This element contains the value of the pseudonym encoded as content of type `xs:base64Binary`.

If the token contains no `abc:Credentials` element but does contain an `abc:Pseudonym`, then this presentation token merely proves knowledge of the secret key underlying the pseudonym.

`.../abc:PresentationTokenDescription/abc:Credential`

This optional element specifies a credential that is presented in this token. If the token contains no `abc:Credential` element but does contain an `abc:Pseudonym`, then this presentation token merely proves knowledge of the user secret underlying the pseudonym.

`.../abc:PresentationTokenDescription/abc:Credential/@Alias`

This optional attribute defines an alias for this credential to refer to attributes from this credential in attribute predicates. See the `/abc:PresentationToken/abc:AttributePredicates` element.

`.../abc:PresentationTokenDescription/abc:Credential/@SameKeyBindingAs`

If present, this XML attribute contains an alias referring either to a `Pseudonym` element within this presentation token, or to another `Credential` element for a credential with key binding. This indicates that the current credential and the referred pseudonym or credential are bound to the *same key*.

The pseudonym or credential that is referred to does not have to refer back to this credential. If the referred to pseudonym or credential also has a `SameKeyBindingAs` attribute that refers to a third pseudonym or credential, then all three pseudonyms/credentials are bound to the same key. In other words, `SameKeyBindingAs` induces a transitive relationship.

`.../abc:Credential/abc:CredentialSpecUID`

This element contains the credential specification identifier of the presented credential.

`.../abc:PresentationTokenDescriptionabc:Credential/abc:IssuerParametersUID`

This element contains the issuer public key identifier of the presented credential.

`.../abc:PresentationTokenDescriptionabc:Credential/abc:RevocationInformationUID`

This optional element contains an identifier of the revocation information with respect to which the presented credential is proved to be non-revoked. The revocation information referenced here corresponds to the issuer-driven revocation parameters referenced from the issuer parameters; see the `/abc:PresentationToken/abc:PresentationTokenDescription/abc:Credential/abc:VerifierDrivenRevocation` element for verifier-driven revocation.

When verifying the token, the verifier has to independently obtain the current revocation information using the mechanism specified by the revocation authority parameters referenced in the `IssuerParameters`. It is up to the verifier to check that the revocation information UID referenced in this element is indeed the most recent one.

`.../abc:PresentationTokenDescription/abc:Credential/abc:Attributes`

This element lists the attributes from this credential that are revealed by this presentation token, either in the clear to the verifier itself, or encrypted to an external inspector.

`.../abc:PresentationTokenDescription/abc:Credential/abc:DisclosedAttribute`

This element specifies one attribute of this credential that is revealed in the presentation token.

`.../abc:Credential/abc:DisclosedAttribute/@AttributeType`

This attribute specifies the type of the credential attribute of which the value is revealed.

There MUST be at most one `abc:DisclosedAttribute` element without `abc:InspectorPublicKeyUID` child element per credential and per attribute type. Also, for `abc:DisclosedAttribute` elements with an `abc:InspectorPublicKeyUID` child element, there MUST be at most one `abc:DisclosedAttribute` element per credential and per attribute type with the same `abc:InspectionGrounds` child element.

`.../abc:Credential/abc:DisclosedAttribute/@DataHandlingPolicy`

This optional XML attribute can be used to refer to an external data handling policy that the Verifier has to adhere to concerning the revealed attribute value. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`.../abc:Credential/abc:DisclosedAttribute/abc:InspectorPublicKeyUID`

This optional element contains the identifier of the inspector public key under which the attribute value is encrypted.

`.../abc:Credential/abc:DisclosedAttribute/abc:InspectionGrounds`

This optional element contains a string describing the valid grounds or circumstances under which the inspector can be asked to decrypt the attribute value. This element must be present whenever a sibling `abc:InspectorPublicKeyUID` element is present. See Section 2.5 for more details on revealing attributes to an inspector.

`.../abc:Credential/abc:DisclosedAttribute/abc:AttributeValue`

This element specifies the value of the revealed attribute. When encrypted to an inspector, this element MAY contain data of type `xs:base64Binary` representing the ciphertext for the encrypted attribute. However, there is no guarantee that this data by itself is decryptable by the inspector. When requesting decryption of an attribute, the complete presentation token must always be sent to the inspector.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation`

This optional element specifies all parameters for checking if a (set of) attribute value(s) from the specified credentials was not revoked using verifier-driven revocation, as requested in the presentation policy by the verifier.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:RevocationInformationUID`

This element contains an identifier of revocation information with respect to which the presented (combination of) attribute value(s) is proved to be non-revoked. The revocation information referenced here corresponds to the verifier-driven revocation parameters mentioned in the verifier's presentation policy; see the `/abc:PresentationToken/abc:Credential/abc:RevocationInformationUID` element for issuer-driven revocation.

When verifying the token, the verifier has to independently obtain the current revocation information using the mechanism specified by the revocation authority parameters referenced in the presentation policy. It is up to the verifier to check that the revocation information UID referenced in this element is indeed the most recent one.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute`

This element specifies a credential attribute that is used for verifier-driven revocation. In case of multiple attributes specified, the User proves that a disjunctive combination of the attribute values was non-revoked with respect to `abc:RevocationInformationUID`.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute/@CredentialAlias`

This attribute specifies the alias of the credential from which the attribute is used. The specified value MUST also occur as an `Alias` attribute in an `abc:Credential` element within this `abc:PresentationToken`.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute/@AttributeType`

This attribute refers to the exact attribute within the credential which is used for verifier driven-revocation.

`.../abc:PresentationTokenDescription/abc:AttributePredicate`

This optional element specifies a predicate that is guaranteed to hold by this token. The child elements are the ordered list of arguments of the predicate.

`.../abc:AttributePredicate/@Function`

This attribute specifies the boolean function for this predicate. See Section 4.4.3 for a list of supported functions and their implications on the list of arguments in the child elements. Note that not all predicate functions can be used for all attributes: the allowed predicate functions depend on the data type and on the chosen encoding of the credential attributes. See Section 4.2.1 for a list of which predicates can be used in combination with which data types and encodings.

`.../abc:AttributePredicate/abc:Attribute`

This element specifies a reference to a credential attribute that is used as an argument of the predicate.

`.../abc:AttributePredicate/abc:Attribute/@CredentialAlias`

This attribute specifies the alias of the credential from which the attribute is used. The specified value MUST also occur as an Alias attribute in an `abc:Credential` element within this `abc:PresentationToken`.

`.../abc:AttributePredicate/abc:Attribute/@AttributeType`

This attribute refers to the exact attribute within the credential that is used as an argument in the predicate.

`.../abc:AttributePredicate/abc:Attribute/@DataHandlingPolicy`

This optional XML attribute can be used to refer to an external data handling policy that the Verifier has to adhere to with respect to the information that the attribute value satisfies the specified predicate. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`.../abc:AttributePredicate/abc:ConstantValue`

This element contains a constant value that is used as an argument in the predicate. The data type of the argument depends on the function of the predicate. See Section 4.4.3 for a list of supported functions and their implications on the list of arguments in the child elements.

`/abc:PresentationToken/abc:CryptoEvidence`

This element contains the cryptographic evidence for the presentation token.

4.4.3 Functions for Use in Predicates

When evaluating predicates over attributes in presentation/issuance policies and tokens, the following list of function URIs from [Sta05] for (in)equality testing of different data types MUST be supported. We refer to Appendix A of [Sta05] for the semantics of these functions and the data types of their arguments. In order to prove predicates over credential attributes, the involved attributes MUST use the same encoding (see Section 4.2.1).

```
urn:oasis:names:tc:xacml:1.0:function:string-equal
urn:oasis:names:tc:xacml:1.0:function:boolean-equal
urn:oasis:names:tc:xacml:1.0:function:integer-equal
urn:oasis:names:tc:xacml:1.0:function:date-equal
urn:oasis:names:tc:xacml:1.0:function:time-equal
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:integer-less-than
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:date-greater-than
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:date-less-than
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal
```

Moreover, this specification defines the following list of new functions for inequality testing.

```
urn:abc4trust:1.0:function:string-not-equal
urn:abc4trust:1.0:function:boolean-not-equal
urn:abc4trust:1.0:function:integer-not-equal
urn:abc4trust:1.0:function:date-not-equal
urn:abc4trust:1.0:function:time-not-equal
urn:abc4trust:1.0:function:dateTime-not-equal
urn:abc4trust:1.0:function:anyURI-not-equal
```

For `type` being one of `string, boolean, integer, date, time, dateTime`, or `anyURI`, the semantics of function `urn:abc4trust:1.0:function:type-not-equal` is defined as follows. The function SHALL take two arguments of data-type `http://www.w3.org/2001/XMLSchema#type` and SHALL return an `http://www.w3.org/2001/XMLSchema#boolean`. The function SHALL return `true` if and only if the application of the corresponding function `urn:oasis:names:tc:xacml:1.0:function:type-equal` evaluated on the same arguments returns `false`. Otherwise, it SHALL return `false`.

Finally, this specification defines the following list of functions for testing equality against a list of candidate values.

```
urn:abc4trust:1.0:function:string-equal-oneof
urn:abc4trust:1.0:function:boolean-equal-oneof
urn:abc4trust:1.0:function:integer-equal-oneof
urn:abc4trust:1.0:function:date-equal-oneof
urn:abc4trust:1.0:function:time-equal-oneof
urn:abc4trust:1.0:function:dateTime-equal-oneof
urn:abc4trust:1.0:function:anyURI-equal-oneof
```

For `type` being one of `string, boolean, integer, date, time, dateTime`, or `anyURI`, the semantics of function `urn:abc4trust:1.0:function:type-equal-oneof` is defined as follows. The function SHALL take two or more arguments of data-type `http://www.w3.org/2001/XMLSchema#type` and SHALL return an `http://www.w3.org/2001/XMLSchema#boolean`. The function SHALL return `true` if and only if the application of the corresponding function `urn:oasis:names:tc:xacml:1.0:function:type-equal` evaluated on the first argument and one of the arguments other than the first returns `true`. Otherwise, it SHALL return `false`.

Note that not all predicate functions can be used for all attributes: the allowed predicate functions depend on the data type and on the chosen encoding of the credential attributes. See Section 4.2.1 for a list of which predicates can be used in combination with which data types and encodings.

4.5 Issuance

Issuance of Privacy-ABCs is an interactive process between the User and the Issuer, possibly involving multiple exchanges of messages. This document specifies the contents, encoding, and processing of the

messages; an application needs to define how to exchange them, e.g., by embedding them in existing messaging protocols. [4]

An overview of a typical issuance interaction is given in Figure 6. The User initiates the interaction by sending an issuance request to the Issuer, optionally specifying the requested credential specification UID.

In the simplest case, the credential is issued "from scratch", i.e., without relation to any existing credentials. Even in this case, the issuance protocol may consist of multiple exchanges of issuance messages.

In a more advanced setting, the new credential that is being issued may carry over attribute values, the user secret from credentials that the User already owns, or may require attributes values to be generated jointly at random. We refer to Section 2.6 for more details on the possibilities of advanced issuance protocols.

In the advanced setting, the issuer responds to the initial request with its issuance policy, which specifies which information the user must present in an issuance token in order to obtain the requested credential, which features of existing credentials will be carried over to the new credential, and which attributes will be generated jointly at random. The user responds with an issuance token. Then, a number of interaction rounds may take place to perform the cryptographic issuance protocol. At the end of these rounds, the Issuer sends the final message allowing the User to construct the issued credential.

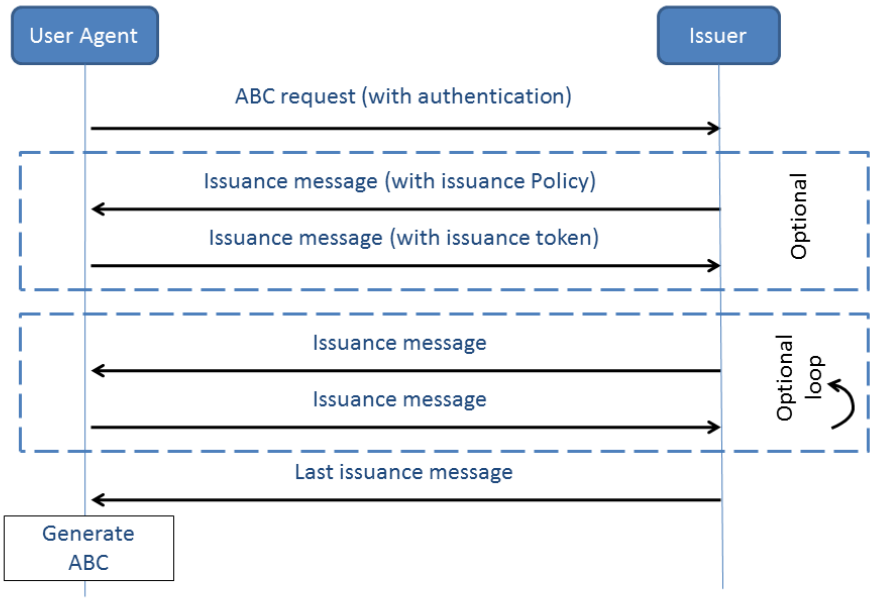

Figure 6: Issuance of Privacy-ABCs

Some notes:

- The endpoint to contact, and its authentication requirements, are application specific. The issuance protocol SHOULD be done over a secure channel to protect the confidentiality of the attribute values.
- Since the exchange is multi-legged, the parties must keep the cryptographic state of each issuance instance between the message exchanges.

[4] For example, WS-Trust [Sta09b] specifies an issuance challenge-response pattern that can be used to carry the ABC issuance messages, embedding them in `RequestSecurityToken` and `RequestSecurityTokenResponse` messages.

User authentication is out of scope of this document. Authentication information MAY be provided along the issuance messages.

4.5.1 Issuance Policy

Optionally, the Issuer may respond to the User's initial request by sending the issuance policy. In an issuance policy, the Issuer describes which credentials he will issue based on which issuance token presented by the User. The newly issued credential can "carry over" certain features from the existing credentials used in generating the issuance token, without revealing these features to the Issuer. Namely, the newly issued credential can be bound to the same User, to the same device, or to the same revocation handle as one of the existing credentials. Also, attribute values in the new credential can be carried over from attributes in the existing credentials, without the Issuer being able to see these attribute values.

In case of simple issuance, i.e., where the User does not have to prove ownership of existing credentials or established pseudonyms, the issuance policy merely specifies the credential specification and the issuer parameters for the credential to be issued. The issuance policy is then used only locally by the Issuer to trigger the issuance protocol.

```
<abc:IssuancePolicy Version="1.0">
  <abc:PresentationPolicy ... > ... </abc:PresentationPolicy>?
  <abc:CredentialTemplate SameKeyBindingAs="xs:anyURI"?>
    <abc:CredentialSpecUID>...</abc:CredentialSpecUID>
    <abc:IssuerParametersUID>...</abc:IssuerParametersUID>
    <abc:UnknownAttributes>
      <abc:CarriedOverAttribute TargetAttributeType="xs:anyURI">
        <abc:SourceCredentialInfo Alias="xs:anyURI"
                                  AttributeType="xs:anyURI"/>
      </abc:CarriedOverAttribute>*
      <abc:JointlyRandomAttribute TargetAttributeType="xs:anyURI"/>*
    </abc:UnknownAttributes>?
  </abc:CredentialTemplate>
</abc:IssuancePolicy>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuancePolicy`

This element describes an issuance policy.

`/abc:IssuancePolicy/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:IssuancePolicy/abc:PresentationPolicy`

This optional element specifies which token has to be presented by the user in order to be issued a credential. See the `/abc:PresentationPolicyAlternatives/abc:PresentationPolicy` element in Section 4.4.1 for a description of the schema. The main goal of this policy and the issuance token returned in response of it is to carry over features from the existing credentials used to generate the presentation token into the newly issued credential.

Note that the presentation policy can also request for a self-signed of self-stated credential; see the `IssuerParametersUID` element in the `PresentationPolicy` for details. Using this feature, the Issuer can have self-signed and self-claimed attributes to be carried over into the newly issued credential. These attribute values will be visible to the Issuer if the issuance policy explicitly specifies that they must be revealed, or will be invisible to the Issuer otherwise.

`/abc:IssuancePolicy/abc:CredentialTemplate/`

This element provides a template for the to-be-issued credential. In case of simple issuance, it will only specify the credential specification and the issuer parameters.

`/abc:IssuancePolicy/abc:CredentialTemplate/@SameKeyBindingAs`

When present, this XML attribute causes the newly issued credential to be bound to the same key as one of the credentials or pseudonyms in the presentation policy. The value of the attribute refers to the `Alias` attribute of the `Pseudonym` or `Credential` from which the key must be carried over.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:CredentialSpecUID`

This element contains the unique identifier of the credential specification of the newly issued credential.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:IssuerParametersUID`

This element contains the unique identifier of the issuer parameters of the newly issued credential.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:UnknownAttributes`

This element specifies the attributes that are unknown to the Issuer and that will either be carried over from another credential or jointly generated at random.

`.../abc:CredentialTemplate/abc:UnknownAttributes/abc:CarriedOverAttribute`

This element describes how an unknown attribute is established.

`.../abc:UnknownAttributes/abc:CarriedOverAttribute/@TargetAttributeType`

This attribute indicates to which attribute in the to-be-issued credential this template information applies to.

`.../abc:UnknownAttributes/abc:CarriedOverAttribute/abc:SourceCredentialInfo`

This element contains information about the source credential to transfer the info from.

`.../abc:CarriedOverAttribute/abc:SourceCredentialInfo/@Alias`

This attribute indicates the alias of the presented credential from which to carry-over the attribute value.

`.../abc:CarriedOverAttribute/abc:SourceCredentialInfo/@AttributeType`

This attribute indicates the attribute type of the presented credential from which to carry-over the attribute value (which could be different than the target attribute type, e.g., from the `LastName` attribute of the `DriverLicense` credential to the `GivenName` attribute of the `StudentCard` credential).

`.../abc:UnknownAttributes/abc:JointlyRandomAttribute`

This element indicates that a specific attribute of the newly issued credential must be generated jointly at random, i.e., so that the Issuer does not learn the value of the attribute, but so that the User cannot bias the uniform distribution of the value.

`.../abc:UnknownAttributes/abc:JointlyRandomAttribute/@TargetAttributeType`

The attribute type of the newly issued credential that must be assigned a jointly generated random value.

4.5.2 Issuance Token

In case of advanced issuance, the User responds with an issuance token, that contains a presentation token and credential template satisfying the issuance policy of the Issuer. In order to satisfy the policy, the credential template in the issuance token must be the same as in the received issuance policy. See Section 4.4.2 for the schema of the presentation token and Section 4.5.1 for the schema of the credential template.

```
<abc:IssuanceToken Version="1.0">
  <abc:IssuanceTokenDescription>
    <abc:PresentationTokenDescription>
    ...
    </abc:PresentationTokenDescription>
    <abc:CredentialTemplate SameKeyBindingAs="xs:anyURI"?>
   ...
    </abc:CredentialTemplate>
  </abc:IssuanceTokenDescription>
  <abc:CryptoEvidence>
   ...
  </abc:CryptoEvidence>
</abc:IssuanceToken>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceToken`

This element describes an issuance token.

`/abc:IssuanceToken/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:IssuanceToken/abc:IssuanceTokenDescription`

This element contains a technology-agnostic description of the revealed information and the new credential.

`.../abc:IssuanceTokenDescription/abc:PresentationTokenDescription`

This element contains a technology-agnostic description of the revealed information.

`.../abc:IssuanceTokenDescription/abc:CredentialTemplate/`

This element provides a template for the to-be-issued credential.

`/abc:IssuanceToken/abc:CryptoEvidence/`

This element provides the cryptographic evidence for the issuance token.

4.5.3 Issuance Messages

Any message that will be exchanged in the course of an issuance protocol is wrapped in an `IssuanceMessage`. That includes the issuance policy and issuance token (if requested by the issuer), as well as the subsequent interactions between the User and Issuer to execute the cryptographic protocol. The message contents in the remaining flows of the issuance protocol are mechanism-specific and therefore treated as opaque pieces of information that are exchanged between the Issuer and the User.

To allow the linkage of the different legs of a protocol, each message includes a Context attribute, which must have the same value on all legs (including the possible preceding issuance policy/token exchange).

```
<abc:IssuanceMessage Context="...">
    ...
</abc:IssuanceMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceMessage`

This element contains either an issuance policy, issuance token or mechanism-specific cryptographic issuance data.

`/abc:IssuanceMessage/@Context`

The message MUST contain a context attribute and its value MUST match the one from the initial IssuanceMessage (if any).

4.5.4 Issuance Log Entries

To keep track of all issued credentials, the issuance log is stored on the issuer side. The issuance log entry contains the verified issuance token (if requested by the issuer), as well as the attribute values specified by the issuer.

```
<abc:IssuanceLogEntry Version="1.0">
    <abc:IssuanceLogEntryUID>...</abc:IssuanceLogEntryUID>
    <abc:IssuerParametersUID>...</abc:IssuerParametersUID>
    <abc:IssuanceToken> ... </abc:IssuanceToken>?
    <abc:IssuerAttributes>
     <abc:Attribute @Type="xs:anyURI">
        <abc:AttributeValue>...</abc:AttributeValue>
     </abc:Attribute>*
    </abc:IssuerAttributes>?
</abc:IssuanceLogEntry>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceLogEntry`

This element contains the verified issuance token (if requested by the issuer), as well as the attribute values specified by the issuer.

`/abc:IssuanceLogEntry/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:IssuanceLogEntry/abc:IssuanceLogEntryUID`

This element contains the identifier of the log entry.

`/abc:IssuanceLogEntry/abc:IssuerParametersUID`

This element contains the identifier of the Issuer's parameters of the issued credential.

`/abc:IssuanceLogEntry/abc:IssuanceToken`

The is optional element contains the verified issuance token.

`/abc:IssuanceLogEntry/abc:IssuerAttributes`

This element contains the description of the attributes (if any) provided by the issuer in an issued credential.

`/abc:IssuanceLogEntry/abc:IssuerAttributes/abc:Attribute`

This element contains the description of an attribute provided by the issuer in an issued credential.

`/abc:IssuanceLogEntry/abc:IssuerAttributes/abc:Attribute/@Type`

This attribute contains the unique identifier of the attribute type of this credential. The attribute type is a URI, to which a semantics is associated by the definition of the attribute type. The definition of attribute types is outside the scope of this document; we refer to Section 7.5 in [Sta09a] for examples. The attribute type (e.g., `http://example.com/firstname`) is *not* to be confused with the data type (e.g., `xs:string`) that is specified by the `DataType` attribute in the `CredentialSpecification`.

`.../abc:IssuerAttributes/abc:Attribute/abc:AttributeValue`

This element contains the actual value of the issued credential attribute provided by the issuer.

4.5.5 Revocation History

To keep track of the revocation process on the upper level, the revocation history is stored on the revocation authority side. Revocation history contains information, including cryptographic data that is used by the revocation authority to support revocation (non-revocation evidence/revocation handle/revocation information generation and updates, keeping track of revocable credentials).

Credentials that are a subject for the verifier-driven revocation are also called revocable in this context. Registering a revocable credential means adding it to the list of the credentials that can be revoked by the revocation authority. This can also include generating fresh revocation handle and/or non-revocation evidence and updating revocation information, if required by the revocation mechanism. In case of verifier-driven revocation, registration of valid attribute values is optional.

```
<abc:RevocationHistory Version="1.0">
   <abc:RevocationHistoryUID>...</abc:RevocationHistoryUID>
   <abc:RevocationAuthorityParametersUID>...
   </abc:RevocationAuthorityParametersUID>
   <abc:CurrentState>...</abc:CurrentState>?
   <abc:RevocationLogEntry @Revoked="xs:boolean">
  <abc:RevocationLogEntryUID>...</abc:RevocationLogEntryUID>
      <abc:RevocableAttribute @Type="xs:anyURI">
          <abc:AttributeValue>...</abc:AttributeValue>
    </abc:RevocableAttribute>*
     <abc:DateCreated>...</abc:DateCreated>
     <abc:CryptoParameters>...</abc:CryptoParameters>?
   </abc:RevocationLogEntry>?
</abc:RevocationHistory>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:RevocationHistory`

This element contains the information that is used by the revocation authority to support revocation and keep track of revocable credentials.

`/abc:RevocationHistory/@Version`

The version attribute has been made optional. It will be removed but currently is kept for backwards compatibility. The version information should be taken from the schema itself rather than from this field. For the current version of the schema, it must be set to `"1.0"`.

`/abc:RevocationHistory/abc:RevocationHistoryUID`

This element contains the identifier of the revocation history.

`/abc:RevocationHistory/abc:RevocationAuthorityParametersUID`

This element contains the identifier of the revocation authority parameters.

`/abc:RevocationHistory/abc:CurrentState`

This optional element contains the information (can also contain cryptographic and revocation mechanism specific data) that is used by the revocation authority to register and revoke credentials.

`/abc:RevocationHistory/abc:RevocationLogEntry`

This element contains information about credentials that were registered and revoked by the revocation authority and the corresponding cryptographic data.

`/abc:RevocationHistory/abc:RevocationLogEntry/@Revoked`

This attribute indicates whether the revocation authority registered a new revocable credential or revoked an existing one.

`/abc:RevocationHistory/abc:RevocationLogEntry/abc:RevocationLogEntryUID`

This element contains the identifier of the revocation log entry.

`/abc:RevocationHistory/abc:RevocationLogEntry/abc:RevocableAttribute`

This element contains the description of an attribute that is used to revoke the credential.

`/abc:RevocationHistory/abc:RevocationLogEntry/abc:RevocableAttribute/@Type`

This attribute contains the unique identifier of the attribute type of the credential attribute that is used to revoke the credential. The attribute type is a URI, to which a semantics is associated by the definition of the attribute type. The definition of attribute types is outside the scope of this document; we refer to Section 7.5 in [Sta09a] for examples. The attribute type (e.g., `http://example.com/firstname`) is *not* to be confused with the data type (e.g., `xs:string`) that is specified by the `DataType` attribute in the `CredentialSpecification`.

`.../abc:RevocationLogEntry/abc:Attribute/abc:AttributeValue`

This element contains the actual value of the credential attribute that is used to revoke the credential. (In case of issuer-driven revocation it contains a value of the revocation handle).

`/abc:RevocationHistory/abc:RevocationLogEntry/abc:DateCreated`

This element contains a timestamp when the credential was registered or revoked by the revocation authority.

`/abc:RevocationHistory/abc:RevocationLogEntry/abc:CryptoParameters`

This element contains mechanism-specific cryptographic data that is used to register or revoke credentials.

4.5.6 Credential Description

At the end of an issuance protocol, the User obtains a new credential. The contents of the new credential are reported back through a `CredentialDescription` element that adheres to the following schema:

```
<abc:CredentialDescription RevokedByIssuer="xs:boolean"?>
  <abc:CredentialUID>...</abc:CredentialUID>
  <abc:FriendlyCredentialName lang="xs:language">
   xs:string
  </abc:FriendlyCredentialName>*
  <abc:ImageReference>xs:anyURI</abc:ImageReference>?
  <abc:CredentialSpecificationUID>...</abc:CredentialSpecificationUID> <abc:IssuerParametersUID>...</abc:
      IssuerParametersUID>
  <abc:SecretReference>...</abc:SecretReference>?
  <abc:Attribute>
    <abc:AttributeUID>...</abc:AttributeUID>
    <abc:AttributeDescription @Type="xs:anyURI" @DataType="xs:anyURI"
      @Encoding="xs:anyURI">
      <abc:FriendlyAttributeName lang="xs:language">
       xs:string
      </abc:FriendlyAttributeName>*
      <abc:AttributeValue>...</abc:AttributeValue>
    </abc:AttributeDescription>
  </abc:Attribute>*
</abc:CredentialDescription>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:CredentialDescription`

This element contains the description of an issued credential in a User's credential portfolio.

`/abc:CredentialDescription/@RevokedByIssuer`

This flag indicates whether this credential was revoked by the issuer. This flag should be set to true as soon as the user knows that this credential was revoked. This flag should be set to false (or omitted) for non-revocable credentials. The default value of this flag is false.

The user's credential store may treat revoked credentials differently than non-revoked ones, in particular it may chose not to store them at all. Revoked credentials will also be skipped by the *PolicyCredentialMatcher*.

`/abc:CredentialDescription/abc:CredentialUID`

This element contains a unique local identifier (formatted as a URI) of the issued credential in the User's credential portfolio. This identifier acts solely as a local reference within the User's system; it is never included in a presentation token or in other artefacts sent across the network for obvious reasons of linkability.

`/abc:CredentialDescription/abc:FriendlyCredentialName`

This optional element provides a friendly textual name for the credential. The content of this element MUST be localized in a specific language.

`/abc:CredentialDescription/abc:FriendlyCredentialName/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyCredentialName` element have been localized.

`/abc:CredentialDescription/abc:ImageReference`

This optional element contains a reference to the endpoint where the image for the credential can be obtained.

When implementing a Privacy-ABC system downloading images from the identity providers should be handled carefully. The reference to the external image resource must not be used every time the credential is presented. To avoid linkability when using the credential, the corresponding image must be downloaded and stored locally at the User's side during the issuance.

`/abc:CredentialDescription/abc:CredentialSpecificationUID`

This element contains the identifier of the credential specification (formatted as a URI) to which the issued credential adheres.

`/abc:CredentialDescription/abc:IssuerParametersUID`

This element contains a reference to the issuer parameters of the Issuer who issued the credential.

`/abc:CredentialDescription/abc:SecretReference`

This optional element contains a unique local identifier (formatted as a URI) of the secret key to which the credential is bound, in case key binding is enabled for this credential. A User may have multiple secret keys; this reference helps in finding the key to which this credential is bound.

This identifier is just a reference to the secret key, not the secret key itself. It acts solely as a local reference within the User's system; it is never included in a presentation token or in other artefacts sent across the network for obvious reasons of linkability.

`/abc:CredentialDescription/abc:Attribute`

This element contains the description of an attribute in an issued credential.

`/abc:CredentialDescription/abc:Attribute/AttributeUID`

This element contains a unique local identifier (formatted as a URI) of this attribute in this credential in the User's credential portfolio. This identifier acts solely as a local reference within the User's system; it is never included in a presentation token or in other artefacts sent across the network for obvious reasons of linkability.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeDescription`

This element contains the generic description of the attribute, as specified in the `/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription` element for this attribute in the credential specification.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeDescription/@Type`

This attribute contains the unique identifier of the attribute type of this credential. The attribute type is a URI, to which a semantics is associated by the definition of the attribute type. The definition of attribute types is outside the scope of this document; we refer to Section 7.5 in [Sta09a] for examples. The attribute type (e.g., `http://example.com/firstname`) is *not* to be confused with the data type (e.g., `xs:string`) that is specified by the `DataType` attribute.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeDescription/@DataType`

This attribute contains the data type of the credential attribute. The supported attribute data types are a subset of XML Schema data types. We refer to Section 4.2.1 for an overview of the supported data types.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeDescription/@Encoding`

To be embedded in a Privacy-ABC, credential attribute values must typically be mapped to fixed-length integers. The `Encoding` XML attribute specifies how the value of this credential attribute is mapped to such an integer. We refer to 4.2.1 for an overview of the supported encoding algorithms.

`/abc:CredentialDescription/abc:Attribute/abc:FriendlyAttributeName`

This optional element provides a friendly textual name for the attribute in the credential. The content of this element MUST be localized in a specific language.

`/abc:CredentialDescription/abc:Attribute/abc:FriendlyAttributeName/@lang`

A required language identifier, using the language codes specified in [Alv01], in which the content of `abc:FriendlyAttributeName` element have been localized.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeValue`

This element contains the actual value of the issued credential attribute.

4.6 Identity Selection and Credential Management

As mentioned in 3.1, the *IdentitySelection* component supports a User in choosing a preferred combination of credentials and/or pseudonyms if there are different possibilities to satisfy a given presentation policy or issuance policy. Also, this component is used to obtain User consent whenever personal data is revealed during presentation or issuance.

In this section, we specify the formats for data that the ABC engine sends to the *IdentitySelection* component, as well as the data formats that it expects in return. While the communication between the ABC engine and the user interface is internal to the architecture, application developers may find it useful to develop dedicated identity selection interface for their particular use case. That's why we document the data formats here.

The formats for data that are sent to the *IdentitySelection* component comprise a part that is common to both credential presentation and credential issuance (see the `<abc:data >` XML element). This common format is also suitable for data being sent to a (graphical) credential management component that allows a User to display the content of her credential repository.

4.6.1 Presentation

4.6.1.1 Arguments sent to the UI for Presentation

```
<abc:UiPresentationArguments>
  <abc:data>
    <abc:credentialSpecifications>
      <abc:credentialSpecification uri="xs:ID">
       <abc:spec>...</abc:spec>
      </abc:credentialSpecification>*
    </abc:credentialSpecifications>?
    <abc:issuers>
      <abc:issuer uri="xs:ID">
        <abc:revocationAuthorityUri>xs:URI
        </abc:revocationAuthorityUri>
        <abc:description>
          <abc:description>...</abc:description>*
        </abc:description>?
      </abc:issuer>*
    </abc:issuers>?
    <abc:revocationAuthorities>
      <abc:revocationAuthority uri="xs:ID">
        <abc:description>
          <abc:description>...</abc:description>*
        </abc:description>?
      </abc:revocationAuthority>*
    </abc:revocationAuthorities>?
    <abc:credentials>
      <abc:credential uri="xs:ID">
        <abc:desc>...</abc:desc>
        <abc:revocationAuthority ref="xs:IDREF" />
        <abc:spec ref="xs:IDREF" />
        <abc:issuer ref="xs:IDREF" />
      </abc:credential>*
    </abc:credentials>?
    <abc:pseudonyms>
      <abc:pseudonym uri="xs:ID">
        <abc:pseudonym Exclusive="xs:boolean" Scope="xs:string"
                       PseudonymUID="xs:anyURI">
          <abc:PseudonymValue>xs:base64Binary</abc:PseudonymValue>?
          <abc:SecretReference>xs:anyURI</abc:SecretReference>
        </abc:pseudonym>
        <abc:metadata>
          <abc:HumanReadableData>xs:string</abc:HumanReadableData>
          <abc:FriendlyPseudonymDescription>...
          </abc:FriendlyPseudonymDescription>
```

```
          <abc:Metadata>...</abc:Metadata>
        </abc:metadata>
      </abc:pseudonym>*
    </abc:pseudonyms>?
    <abc:inspectors>
      <abc:inspector uri="xs:ID">
        <abc:description>
          <abc:description>...</abc:description>*
        </abc:description>?
      </abc:inspector>*
    </abc:inspectors>?
  </abc:data>
  <abc:tokenCandidatesPerPolicy>
    <abc:tokenCandidatePerPolicy policyId="xs:int">
      <abc:policy>...</abc:policy>
      <abc:tokenCandidates>
        <abc:tokenCandidate candidateId="xs:int">
          <abc:tokenDescription>...</abc:tokenDescription>
          <abc:credentials>
            <abc:credential ref="xs:IDREF" />*
          </abc:credentials>?
          <abc:pseudonymCandidates>
            <abc:pseudonymCandidate candidateId="xs:int">
              <abc:pseudonyms>
                <abc:pseudonym ref="xs:IDREF" />*
              </abc:pseudonyms>?
            </abc:pseudonymCandidate>+
          </abc:pseudonymCandidates>
          <abc:revealedFacts>
            <abc:revealedFact>
              <abc:descriptions>
                <abc:description>...</abc:description>*
              </abc:descriptions>?
            </abc:revealedFact>*
          </abc:revealedFacts>?
          <abc:revealedAttributeValues>
            <abc:revealedAttributeValue>
              <abc:descriptions>
                <abc:description>...</abc:description>*
              </abc:descriptions>?
            </abc:revealedAttributeValue>*
          </abc:revealedAttributeValues>?
          <abc:inspectableAttributes>
            <abc:inspectableAttribute>
              <abc:credential ref="xs:IDREF" />*
              <abc:attributeType>xs:string</abc:attributeType>
              <abc:dataHandlingPolicy>xs:string
              </abc:dataHandlingPolicy>
              <abc:inspectionGrounds>xs:string
              </abc:inspectionGrounds>
              <abc:inspectorAlternatives>
                <abc:inspectorAlternative ref="xs:IDREF" />*
              </abc:inspectorAlternatives>?
            </abc:inspectableAttribute>*
          </abc:inspectableAttributes>?
        </abc:tokenCandidate>+
      </abc:tokenCandidates>
    </abc:tokenCandidatePerPolicy>+
  </abc:tokenCandidatesPerPolicy>
</abc:UiPresentationArguments>
```

`/abc:UiPresentationArguments`

This XML root Element is sent by the ABC Engine to the user interface to perform identity selection for presentation. The user interface must then choose which combination of credentials and/or pseudonyms, all satisfying the policy, should be used to complete the presentation proof.

`/abc:UiPresentationArguments/abc:data`

This element contains information about all credential specifications, issuers, revocation authorities, credentials, pseudonyms and inspectors that are used in this XML. Data under this element must not appear twice. All data in this element should be referenced at least once in this XML.

`/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications`

The wrapper for the list of credential specification.

`/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications/abc:credentialSpecification`

An entry in the list of credential specifications.

`/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications/abc:credentialSpecification/@uri`

This element must contain the `specificationUid` of the credential specification in the `spec` element. The subsequent XML code must refer to this credential specification by this uri.

`/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications/abc:spec`

This element contains the actual `credentialSpecification` element, as output by the *Key Manager*. The contents MUST be of the type `/abc:CredentialSpecification`.

`/abc:UiPresentationArguments/abc:data/abc:issuers`

Wrapper for the list of issuers.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer`

An entry in the list of issuers.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/@uri`

This element must contain the `parametersUid` of the issuer parameters of this particular issuer. The subsequent XML code must refer to this issuer by this uri.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/abc:revocationAuthorityUri`

This element must contain a copy of the `revocationParametersUID` element of the issuer parameters of this particular issuer.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/abc:description`

Wrapper for the list of friendly issuer descriptions. The contents of this list must be a copy of the list of `friendlyIssuerDescriptions` in the issuer parameters of this particular issuer.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/abc:description/abc:description`

An entry in the list of friendly issuer descriptions. It must be a copy of the corresponding entry of `friendlyIssuerDescriptions` in the issuer parameters of this particular issuer. The contents MUST be of the type `/abc:CredentialSpecification/abc:FriendlyCredentialName`.

`/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/abc:spec/@ref`

This is a reference to the credential specification associated with this issuer. It must be equal to the credentialSpecUID element of the issuer parameters of this particular issuer. It refers to `/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications/abc:credentialSpecification/@uri`.

`/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities`

Wrapper for the list of revocation authorities.

`/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities/abc:revocationAuthority`

An entry in the list of revocation authorities.

`/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities/abc:revocationAuthority/@uri`

This element must contain the `parametersUid` of the revocation authority parameters of this particular revocation authority. The subsequent XML code must refer to this revocation authority by this uri.

`/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities/abc:revocationAuthority/abc:description`

Wrapper for the list of friendly revocation authority descriptions. Since revocation authorities don't have a friendly description yet, this element is currently unused. In the future, the contents of this list should be a copy of the list of friendly descriptions in the revocation authority parameters of this particular revocation authority.

`/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities/abc:revocationAuthority/abc:description/abc:description`

An entry in the list of friendly revocation authority descriptions. Currently, this element is unused. In the future, it should be a copy of the corresponding entry of the friendly description in the revocation authority parameters of this particular revocation authority. The contents MUST be of the type `/abc:CredentialSpecification/abc:FriendlyCredentialName`.

`/abc:UiPresentationArguments/abc:data/abc:credentials`

Wrapper for the list of credentials.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential`

An entry in the list of credentials.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/@uri`

This element must contain the `credentialUid` of the credential description of this particular credential. The subsequent XML code must refer to this credential by this uri.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:desc`

This element contains the actual `credentialDescription` element corresponding to this credential, as output by the Credential Manager. The contents MUST be of the type `/abc:CredentialDescription`.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:revocationAuthority`

Wrapper for the reference to the revocation authority responsible for issuer-driven revocation for this credential.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:revocationAuthority/@ref`

This is a reference to the revocation authority responsible for issuer-driven revocation for this credential. It must be equal to the `revocationParametersUID` element of the issuer parameters associated with this credential. It refers to `/abc:UiPresentationArguments/abc:data/abc:revocationAuthorities/abc:revocationAuthority/@uri`.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:spec`

Wrapper for the reference to the credential specification of this credential.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:spec/@ref`

This is a reference to the credential specification associated with this credential. It must be equal to the `credentialSpecificationUI` element of the credential description of this credential. It refers to `/abc:UiPresentationArguments/abc:data/abc:credentialSpecifications/abc:credentialSpecification/@uri`.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:issuer`

Wrapper for the reference to the issuer associated with this credential.

`/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/abc:issuer/@ref`

This is a reference to the issuer associated with this credential. It must be equal to the `issuerParametersUID` element of the credential description of this credential. It refers to `/abc:UiPresentationArguments/abc:data/abc:issuers/abc:issuer/@uri`.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms`

Wrapper for the list of pseudonyms. This list contains:

- pseudonyms that were retrieved from the Credential Manager
- each time that the policy allows the creation of a new pseudonym, this list will contain entries corresponding to the newly created pseudonyms. If the policy does not restrict the secret these new pseudonyms are bound to, then one pseudonym will be created for each secret in the Credential Manager.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym`

An entry in the list of pseudonyms.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/@uri`

This element must contain the `pseudonymUID` of this pseudonym. The subsequent XML code must refer to this pseudonym by this uri.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym`

This element contains a description of the actual pseudonym.

For newly created pseudonyms, the fields `SecretReference`, `Exclusive`, `Scope`, and `PseudonymUID` will be set automatically; the `PseudonymValue` field will be left out.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym/@Exclusive`

A Boolean flag indicating whether this is a scope-exclusive pseudonym.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym/@Scope`

The scope of the pseudonym.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym/@PseudonymUID`

The UID of the pseudonym. This value must be exactly equal to `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/@uri`. The reason this element is there is because the client code might treat `xs:ID` and `xs:anyURI` types differently (in particular, the xs:ID type might be hidden to the client code).

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym/abc:PseudonymValue`

This field contains the value of that pseudonym.

This field is mandatory for existing pseudonyms. For not-yet-created pseudonyms, this field MAY be absent.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:pseudonym/abc:SecretReference`

This field contains a reference to the secret used for key binding for this pseudonym.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata`

This element contains the metadata of the pseudonym. The UI is allowed to change the metadata of any pseudonym.

For newly created pseudonyms, this element will consist of dummy values.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata/abc:HumanReadableData`

This field is deprecated, you should use `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata/abc:FriendlyPseudonymDescription` instead.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata/abc:FriendlyPseudonymDescription`

Wrapper for the list of friendly descriptions for the pseudonyms. Each friendly description is of the same type as `/abc:IssuerParameters/abc:FriendlyIssuerDescription`.

`/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata/abc:Metadata`

This element contains arbitrary data.

`/abc:UiPresentationArguments/abc:data/abc:inspectors`

Wrapper for the list of inspectors.

`/abc:UiPresentationArguments/abc:data/abc:inspectors/abc:inspector`

An entry in the list of inspectors.

`/abc:UiPresentationArguments/abc:data/abc:inspectors/abc:inspector/@uri`

This element must contain the publicKeyUID of the public key of this inspector. The subsequent XML code must refer to this inspector by this uri.

`/abc:UiPresentationArguments/abc:data/abc:inspectors/abc:inspector/abc:description`

Wrapper for the list of friendly inspector descriptions. The contents of this list must be a copy of the list of `friendlyInspectorDescriptions` in the inspector public key of this inspector.

`/abc:UiPresentationArguments/abc:data/abc:inspectors/abc:inspector/abc:description/abc:description`

An entry in the list of friendly inspector descriptions. It must be a copy of the corresponding entry of `friendlyInspectorDescriptions` in the inspector public key of this particular inspector. The contents MUST be of the type `/abc:CredentialSpecification/abc:FriendlyCredentialName`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy`

Wrapper for the list of token candidates per policy.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy`

An entry in the list of token candidates per policy. Each entry refers to one of the policy alternatives. Policy alternatives which cannot be satisfied are skipped.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/@policyId`

An identifier for the tokencandidatePerPolicy. It is assigned sequentially, and is needed in the return value.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:policy`

A copy of the presentation policy to which this tokenCandidatePerPolicy refers to. The contents MUST be of the type `/abc:PresentationPolicyAlternatives/abc:PresentationPolicy`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates`

Wrapper for the list of token candidates for this policy.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate`

An entry in the list of token candidate for this policy. One token candidate is established for each acceptable credential assignment.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/@candidateId`

An identifier for this token candidate. It is assigned sequentially, and reset for each policy. It is needed in the return value.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:tokenDescription`

A partially filled out presentation token description for this candidate token. The pseudonym choice and the inspector choice are not yet set. The contents MUST be of the type `/abc:PresentationToken/abc:PresentationTokenDescription`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:credentials`

Wrapper for the list of credentials for this credential assignment of this candidate token. If no credentials need to be shown in this policy, then this list will be empty.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:credentials/abc:credential`

An entry in the list of credentials for the credential assignment of this candidate token. The n-th item in this list corresponds to the n-th credential in the policy. Each entry is a wrapper for a reference to a credential.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:credentials/abc:credential/@ref`

A reference to a credential. This refers to `/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/@uri`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates`

A wrapper for a list of alternative pseudonym assignments for this candidate token. This list also includes pseudonyms assignments containing newly established pseudonyms.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate`

An entry in the list of alternative pseudonym assignments for this candidate token. The user interface has to chose one alternative among the ones proposed. If no pseudonyms need to be shown in this policy, then the list will contain exactly one pseudonym candidate (consisting of an empty list of pseudonyms).

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate/@candidateId`

An identifier for this pseudonym candidate. It is assigned sequentially, and reset for each token candidate. It is needed in the return value.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate/abc:pseudonyms`

A wrapper for the list of pseudonyms in this pseudonym candidate. If no pseudonyms need to be shown in this policy, then the list will be empty.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate/abc:pseudonyms/abc:pseudonym`

An entry in the list of pseudonyms for this pseudonym candidate. The n-th item in this list corresponds to the n-th pseudonym in the policy. Each entry is a wrapper for a reference to a pseudonym.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate/abc:pseudonyms/abc:pseudonym/@ref`

A reference to a pseudonym. It refers to `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/@uri`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedFacts`

A wrapper for the list of revealed facts for this token candidate. One or more revealed facts may be created for each predicate in the presentation token, and describe what is being revealed on the cryptographic layer (which might be more information than can be deduced from the presentation token description alone).

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedFacts/abc:revealedFact`

An entry in the list of revealed facts for this token candidates.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedFacts/abc:revealedFact/abc:descriptions`

A wrapper for a list of human-readable descriptions of this revealed fact. The entries all contain the same description, with each entry being in a different language.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedFacts/abc:revealedFact/abc:descriptions/abc:description`

An entry in the list of human-readable descriptions of this revealed fact. The contents MUST be of the type `/abc:CredentialSpecification/abc:FriendlyCredentialName`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedAttributeValues`

A wrapper for the list of revealed attribute values for this token candidate. There will be exactly one entry for each attribute whose value is being revealed to the verifier by the crypto engine (which might be more attributes than can be deduced from the presentation token description alone).

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedAttributeValues/abc:revealedAttributeValue`

An entry in the list of revealed attribute values for this token candidate.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedAttributeValues/abc:revealedAttributeValue/abc:descriptions`

A wrapper for a list of human-readable descriptions of this revealed attribute value. The entries contain the same description, with each entry being in a different language.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:revealedAttributeValues/abc:revealedAttributeValue/abc:descriptions/abc:description`

An entry in the list of human-readable descriptions of this revealed attribute. The contents MUST be of the type `/abc:CredentialSpecification/abc:FriendlyCredentialName`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes`

A wrapper for the list of inspectable attributes in this token candidate.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute`

An entry in the list of inspectable attributes in this token candidate.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:credential`

A wrapper for the reference to the credential which contains this inspectable attribute.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:credential/@ref`

The reference to the credential which contains this inspectable attribute. It refers to `/abc:UiPresentationArguments/abc:data/abc:credentials/abc:credential/@uri`.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:attributeType`

The attribute type of this inspectable attribute.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:dataHandlingPolicy`

A copy of the data handling policy for this inspectable attribute.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectionGrounds`

A copy of the inspection grounds of this inspectable attribute.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectorAlternatives`

A wrapper for the list of inspector alternatives for this inspectable attribute. For each inspectable attribute, the user interface has to choose one inspector among this list.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectorAlternatives/abc:inspectorAlternative`

An entry in the list of inspector alternatives for this inspectable attribute. This entry is a wrapper to a reference to an inspector.

`/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectorAlternatives/abc:inspectorAlternative/@ref`

Reference to an inspector for this inspectable attribute among the list of possible alternatives. It refers to `/abc:UiPresentationArguments/abc:data/abc:inspectors/abc:inspector/@uri`.

4.6.1.2 Return Value sent by the UI for Presentation

```
<abc:UiPresentationReturn>
  <abc:chosenPolicy>xs:int</abc:chosenPolicy>
  <abc:chosenPresentationToken>xs:int</abc:chosenPresentationToken>
  <abc:metadataToChange>
    <abc:entry>
      <abc:key>xs:string</abc:key>
      <abc:value>...</abc:value>
    </abc:entry>*
  </abc:metadataToChange>
  <abc:chosenPseudonymList>xs:int</abc:chosenPseudonymList>?
  <abc:chosenInspectors>xs:string</abc:chosenInspectors>*
</abc:UiPresentationReturn>
```

`/abc:UiPresentationReturn`

This XML root Element that the user interface sends back to the ABC Engine to complete identity selection for presentation. It contains the choice of credentials and pseudonyms that should be used to complete the presentation proof.

`/abc:UiPresentationReturn/abc:chosenPolicy`

The ID of the policy chosen by the user interface. It refers to `/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/@policyId`.

`/abc:UiPresentationReturn/abc:chosenPresentationToken`

The ID of the presentation token candidate (within the selected policy) chosen by the user interface. It refers to `/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/@candidateId`.

`/abc:UiPresentationReturn/abc:metadataToChange`

This element contains a list of entries (key-value pairs) of `PseudonymMetadata` that the user interface wishes to change. It should contain an entry for all newly created pseudonyms which were selected.

`/abc:UiPresentationReturn/abc:metadataToChange/abc:entry`

A key-value pair.

`/abc:UiPresentationReturn/abc:metadataToChange/abc:entry/abc:key`

The key corresponds to the `pseudonymUID` of the pseudonym whose metatdata the user interface wishes to change. It refers to `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/@uri`.

`/abc:UiPresentationReturn/abc:metadataToChange/abc:entry/abc:value`

The value corresponds to the new metadata of the pseudonym. The ABC Engine will instruct the *Credential Manager* to replace the old metadata of that pseudonym by the given value. The user interface should take the value in `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata` as a template for creating the new metadata. The contents MUST be of the type `/abc:UiPresentationArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata`.

`/abc:UiPresentationReturn/abc:chosenPseudonymList`

The ID of the chosen pseudonym candidate list (for the chosen candidate token). It refers to `/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseudonymCandidate/@candidateId`. If the policy does not require showing pseudonyms, then this field may be left out.

`/abc:UiPresentationReturn/abc:chosenInspectors`

The list of inspectors that the user interface chose. This list should contain one entry per inspectable attribute (for the chosen candidate token). For each inspectable attribute, one inspector should be chosen among the list of alternatives. The list entries must refer to `/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectorAlternatives/abc:inspectorAlternative/@ref`.

4.6.2 Issuance

4.6.2.1 Arguments sent to the UI for Issuance

```
<abc:UiIssuanceArguments>
  <abc:data>...</abc:data>
  <abc:tokenCandidates>...</abc:tokenCandidates>
  <abc:policy>...</abc:policy>
</abc:UiIssuanceArguments>
```

`/abc:UiIssuanceArguments`

This XML root Element is sent by the ABC Engine to the user interface to perform identity selection for issuance. The user interface must then choose which combination of credentials and/or pseudonyms, all satisfying the policy, should be used to complete the issuance proof.

`/abc:UiIssuanceArguments/abc:data`

See `/abc:UiPresentationArguments/abc:data`.

`/abc:UiIssuanceArguments/abc:tokenCandidates`

The semantics of this element are analogous to `/abc:UiPresentationArguments/abc:tokenCandidatesPerPolicy/abc:tokenCandidatePerPolicy/abc:tokenCandidates`, except that they refer to the unique issuance policy instead of one alternative of the presentation policies. References therein point to `/abc:UiIssuanceArguments/abc:data` and not to `/abc:UiPresentationArguments/abc:data`.

`/abc:UiIssuanceArguments/abc:policy`

This element contains a copy of the issuance policy. The contents MUST be of the type `/abc:IssuancePolicy`.

```
<abc:UiIssuanceReturn>
  <abc:chosenIssuanceToken>xs:int</abc:chosenIssuanceToken>
  <abc:metadataToChange>
    <abc:entry>
      <abc:key>xs:string</abc:key>
      <abc:value>...</abc:value>
    </abc:entry>*
  </abc:metadataToChange>
  <abc:chosenPseudonymList>xs:int</abc:chosenPseudonymList>?
  <abc:chosenInspectors>xs:string</abc:chosenInspectors>*
</abc:UiIssuanceReturn>
```

`/abc:UiIssuanceReturn`

This XML root element that the user interface sends back to the ABC Engine to complete identity selection for issuance. It contains the choice of credentials and pseudonyms that should be used to complete the issuance proof.

`/abc:UiIssuanceReturn/abc:chosenIssuanceToken`

The ID of the issuance token candidate chosen by the user interface. It refers to `/abc:UiIssuanceArguments/abc:tokenCandidates/abc:tokenCandidate/@candidateId`.

`/abc:UiIssuanceReturn/abc:metadataToChange`

See `/abc:UiPresentationReturn/abc:metadataToChange`.

`/abc:UiIssuanceReturn/abc:metadataToChange/abc:entry`

See `/abc:UiPresentationReturn/abc:metadataToChange/abc:entry`.

`/abc:UiIssuanceReturn/abc:metadataToChange/abc:entry/abc:key`

The key corresponds to the pseudonymUID of the pseudonym whose metatdata the user interface wishes to change. It refers to `/abc:UiIssuanceArguments/abc:data/abc:pseudonyms/abc:pseudonym/@uri`.

`/abc:UiIssuanceReturn/abc:metadataToChange/abc:entry/abc:value`

The value corresponds to the new metadata of the pseudonym. The ABC Engine will instruct the Credential Manager to replace the old metadata of that pseudonym by the given value. The user interface should take the value in `/abc:UiIssuanceArguments/abc:data/abc:pseudonyms/abc:pseudonym/abc:metadata` as a template for creating the new metadata. The contents MUST be of the type `abc:PseudonymWithMetadata/abc:PseudonymMetadata`.

`/abc:UiIssuanceReturn/abc:chosenPseudonymList`

The ID of the chosen pseudonym candidate list (for the chosen candidate token). It refers to `/abc:UiIssuanceArguments/abc:tokenCandidates/abc:tokenCandidate/abc:pseudonymCandidates/abc:pseu`

donymCandidate/@candidateId. If no pseudonym needs to be shown for this policy, this field may be left out.

/abc:UiIssuanceReturn/abc:chosenInspectors

The list of inspectors that the user interface chose. This list should contain one entry per inspectable attribute (for the chosen candidate token). For each inspectable attribute, one inspector should be chosen among the list of alternatives. The list entries must refer to /abc:UiIssuanceArguments/abc:tokenCandidates/abc:tokenCandidate/abc:inspectableAttributes/abc:inspectableAttribute/abc:inspectorAlternatives/abc:inspectorAlternative/@ref.

4.7 Formats Used By the Webservice API

Since the webservices can only take a single XML root element as input, several elements have been constructed to combine previously defined elements.

4.7.1 CredentialSpecificationAndSystemParameters

```
<abc:CredentialSpecificationAndSystemParameters>
  <abc:CredentialSpecification>...</abc:CredentialSpecifion>
  <abc:SystemParameters>...</abc:SystemParameters>
</abc:CredentialSpecificationAndSystemParameters>
```

/abc:CredentialSpecificationAndSystemParameters

This XML root Element contains a credential specification and a set of system parameters.

/abc:CredentialSpecificationAndSystemParameters/abc:CredentialSpecification

Must be of type /abc:CredentialSpecification.

/abc:CredentialSpecificationAndSystemParameters/abc:SystemParameters

Must be of type /abc:SystemParameters.

4.7.2 IssuancePolicyAndAttributes

```
<abc:IssuancePolicyAndAttributes>
  <abc:IssuancePolicy>...</abc:IssuancePolicy>
  <abc:Attribute>...</abc:Attribute>*
</abc:IssuancePolicyAndAttributes>
```

/abc:IssuancePolicyAndAttributes

This XML root Element contains an issuance policy and a number of attributes.

/abc:IssuancePolicyAndAttributes/abc:IssuancePolicy

Must be of type /abc:IssuancePolicy.

/abc:IssuancePolicyAndAttributes/abc:Attribute

Must be of type /abc:Attribute.

4.7.3 IssuanceMessageAndBoolean

```
<abc:IssuanceMessageAndBoolean>
  <abc:IssuanceMessage>...</abc:IssuanceMessage>
  <abc:LastMesage>xs:boolean</abc:LastMessage>
  <abc:IssuanceLogEntryURI>xs:anyURI</abc:IssuanceLogEntryURI>
</abc:IssuanceMessageAndBoolean>
```

`/abc:IssuanceMessageAndBoolean`

This XML root Element contains an issuance message, a `boolean` indicating if this is the last step of issuance and an URI pointing to the relevant log entry.

`/abc:IssuanceMessageAndBoolean/abc:IssuanceMessage`

Must be of type `/abc:IssuanceMessage`.

`/abc:IssuanceMessageAndBoolean/abc:LastMessage`

`Boolean` indicating if this is the last message of the issuance protocol.

`/abc:IssuanceMessageAndBoolean/abc:IssuanceLogEntryURI`

URI pointing to the relevant `IssuanceLogEntry` in the issuer log.

4.7.4 RevocationReferences

```
<abc:RevocationReferences>
  <abc:RevocationInfoReference>...</abc:RevocationInfoReference>
  <abc:NonRevocationEvidenceReference>
    ...
  </abc:NonRevocationEvidenceReference>
  <abc:NonRevocationEvidenceUpdate>...</abc:NonRevocationEvidenceUpdate>
</abc:RevocationReferences>
```

`/abc:RevocationReferences`

This element contains 3 References, describing an URL where revocation information can be obtained.

`/abc:RevocationReferences/abc:RevocationInfoReference`

Must be of type `/abc:Reference`.

`/abc:RevocationReferences/abc:NonRevocationEvidenceReference`

Must be of type `/abc:Reference`.

`/abc:RevocationReferences/abc:NonRevocationEvidenceUpdateReference`

Must be of type `/abc:Reference`.

4.7.5 PresentationPolicyAlternativesAndPresentationToken

```
<abc:PresentationPolicyAlternativesAndPresentationToken>
  <abc:PresentationPolicyAlternatives>
    ...
  </abc:PresentationPolicyAlternatives>
  <abc:PresentationToken>...</abc:PresentationToken>
</abc:PresentationPolicyAlternativesAndPresentationToken>
```

`/abc:PresentationPolicyAlternativesAndPresentationToken`

This element contains a `PresentationPolicyAlternatives` and a `PresentationToken` element.

`/abc:PresentationPolicyAlternativesAndPresentationToken/abc:PresentationPolicyAlternatives`

Must be of type `/abc:PresentationPolicyAlternatives`.

`/abc:PresentationPolicyAlternativesAndPresentationToken/abc:PresentationToken`

Must be of type `/abc:PresentationToken`.

4.7.6 AttributeList

```
<abc:AttributeList>
  <abc:Attributes>...</abc:Attributes>*
</abc:AttributeList>
```

`/abc:AttributeList`

This element contains a list of `Attributes`, corresponding to `List <Attribute >`.

`/abc:AttributeList/abc:Attributes`

Must be of type `/abc:Attribute`.

4.7.7 ABCEBoolean

```
<abc:ABCEBoolean value="xs:boolean"/>
```

`/abc:ABCEBoolean`

This element is used to indicate `boolean` value, that is, either the value true or false.

`/abc:ABCEBoolean/@value`

This attribute states the value of the `boolean`.

4.7.8 URISet

```
<abc:URISet>
  <abc:URI>xs:anyURI</abc:URI>*
</abc:URISet>
```

`/abc:URISet`

This element contains a set of URIs, corresponding to `Set <URI >`.

`/abc:URISet/abc:URI`

This element contains a URI.

4.7.9 IssuerParametersInput

```
<abc:IssuerParametersInput Version="1.0">
  <abc:ParametersUID>xs:anyURI</abc:ParametersUID>
  <abc:FriendlyIssuerDescription lang="xs:language">
    xs:string
  </abc:FriendlyIssuerDescription>*
  <abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
  <abc:HashAlgorithm>xs:anyURI</abc:HashAlgorithm>
  <abc:RevocationParametersUID>xs:anyURI</abc:RevocationParametersUID>
</abc:IssuerParametersInput>
```

`/abc:IssuerParametersInput`

This element contains a subset of the elements that the element `/abc:IssuerParameters` contains.

`/abc:IssuerParametersInput/abc:ParametersUID`

See `/abc:IssuerParameters/abc:ParametersUID`.

`/abc:IssuerParametersInput/abc:FriendlyIssuerDescription`

See `/abc:IssuerParameters/abc:FriendlyIssuerDescription`.

`/abc:IssuerParametersInput/abc:AlgorithmID`

See `/abc:IssuerParameters/abc:AlgorithmID`.

`/abc:IssuerParametersInput/abc:HashAlgorithm`

See `/abc:IssuerParameters/abc:HashAlgorithm`.

`/abc:IssuerParametersInput/abc:RevocationParametersUID`

See `/abc:IssuerParameters/abc:RevocationParametersUID`.

4.7.10 IssuanceReturn

```
<abc:IssuanceReturn>
  <abc:IssuanceMessage>...</abc:IssuanceMessage>
  <abc:CredentialDescription>...</abc:CredentialDescription>
  <abc:UiIssuanceArguments>...</abc:UiIssuanceArguments>
</abc:IssuanceReturn>
```

`/abc:IssuanceReturn`

This element contains an issuance message, a credential description, and a `UiIssuanceArguments` element.

`/abc:IssuanceReturn/abc:IssuanceMessage`

See Section 4.5.3, Issuance Messages.

`/abc:IssuanceReturn/abc:CredentialDescription`

See Section 4.5.6, Credential Description.

`/abc:IssuanceReturn/abc:UiIssuanceArguments`

See Section 4.6.1.1, Arguments sent to the UI for Issuance.

5 API for Privacy-ABCs

This chapter describes the application programming interfaces (API) of the ABCE layer, focusing solely on the API that the ABCE layer exposes to the upper layers, in particular, to the application layer. This information is mainly intended for application developers who want to build applications that make use of ABCE technology.

The interfaces are described in an object-oriented fashion as a list of methods that take input parameters of certain types and that produce an output of a certain return type. The data types of the input and return types either refer to XML artifacts as defined in Chapter 4 or to simple XML Schema datatypes such as boolean or string.

For ease of integration with applications built on top of our ABCE layer, the actual implementation offers the top-level ABCE interfaces described below also as web services. The descriptions below must therefore be mapped to descriptions in the Web Services Description Language (WSDL). Doing so is straightforward, so for the sake of readability we stick to an object-oriented notation here.

5.1 ABCE methods for Users

`boolean canBeSatisfied(PresentationPolicyAlternatives p)`

On input of presentation policy alternatives, this method determines whether the user has the necessary credentials and established pseudonyms to create a presentation token that satisfies the policy. If so, this method returns true, otherwise, it returns false.

Path	/user/canBeSatisfied/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	PresentationPolicyAlternatives
Output Type	text/xml
Output Format	ABCEBoolean

`UiPresentationArguments createPresentationToken(PresentationPolicyAlternatives p)`

On input of presentation policy alternatives, this method determines whether the user has the necessary credentials and established pseudonyms to create a presentation token that satisfies the policy.

If there is at least one way in which the policy can be satisfied with the user's credentials and pseudonyms, this method returns an object that encodes the different alternatives. The caller then lets the user choose her preferred way of satisfying the policy or let her cancel the transaction, for example by displaying the relevant information in a (graphical) user interface called the *identity selector*. If the presentation policy alternatives cannot be satisfied, this method returns an error.

Path	/user/createPresentationToken/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	PresentationPolicyAlternatives
Output Type	text/xml
Output Format	UiPresentationArguments

`PresentationToken createPresentationToken(UiPresentationReturn upr)`

After the user has chosen her preferred way of satisfying the presentation policy in the identity selector, she calls this method on input of the object encoding her choice.

This method generates a presentation token that reflects this choice, and which satisfies the respective presentation policy alternatives. The generated presentation token consists of two parts: (1) a description of the token's content, which largely repeats the information of the corresponding alternative in the presentation policy; and (2) cryptographic evidence, which mainly consists of a non-interactive zero-knowledge proof (using the Fiat-Shamir heuristic) that the user owns all the credentials and pseudonyms referenced in the token, that all revocable credentials are not revoked, that all inspectable attributes were encrypted correctly, and that all predicates hold. Furthermore, the presentation token contains a reference to the policy alternative that the user chose to fulfil.

This method returns the generated presentation token.

Path	/user/createPresentationTokenUi/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	UiPresentationReturn
Output Type	text/xml
Output Format	PresentationToken

`IssuanceReturn issuanceProtocolStep(IssuanceMessage im)`

This method performs one step in an interactive issuance protocol. It takes as input an issuance message received from an issuer.

The method has exactly one of the following three return values: (1) an issuance message, which has to be sent to the issuer; (2) a description of the newly issued credential — this return value indicates that the protocol was completed successfully and that the newly issued credential was stored in the user's credential manager; (3) an object of type UiIssuanceArguments which encodes the user's choices to satisfy the issuance policy (and which, for example, is forwarded to an identity selection user interface). In the latter case, the user must then call the other `issuanceProtocolStep()` method with an object that reflects the selected choice.

During simple issuance, this method never returns an object of type of type `UiIssuanceArguments`.

Path	/user/issuanceProtocolStep/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	IssuanceMessage
Output Type	text/xml
Output Format	IssuanceReturn

`IssuanceMessage issuanceProtocolStep(UiIssuanceReturn uir)`

This method is called during the issuance protocol after the other `issuanceProtocolStep()` method returned an object of type `UiIssuanceArguments`, and after the user has made her choice (in the identity selector) on how to satisfy the issuance policy.

The input to this method is an object which encodes the user's choice on how to satisfy the issuance policy. The method returns an issuance message, which has to be sent to the issuer.

Path	/user/issuanceProtocolStepUi/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	UiIssuanceReturn
Output Type	text/xml
Output Format	IssuanceMessage

`void updateNonRevocationEvidence()`

This method updates the non-revocation evidence associated to all credentials in the credential store. Calling this method at regular time intervals reduces the likelihood of having to update non-revocation evidence at the time of presentation, thereby not only speeding up the presentation process, but also offering improved privacy as the Revocation Authority is no longer "pinged" at the moment of presentation.

Path	/user/updateNonRevocationEvidence/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	None
Output Type	text/xml
Output Format	None

`URI[] listCredentials()`

This method returns an array of all unique credential identifiers (UIDs) available in the Credential Manager.

Path	/user/listCredentials/
HTTP Method	GET
Output Type	text/xml
Output Format	URISet

`CredentialDescription getCredentialDescription(URI credUid)`

This method returns the description of the credential with the given unique identifier. The unique credential identifier `credUid` is the identifier which was included in the credential description that was returned at successful completion of the issuance protocol.

Path	/user/getCredentialDescription/{credentialUid}
HTTP Method	GET
Output Type	text/xml
Output Format	CredentialDescription

Path Parameter	Parameter Type
credentialUid	URI

`boolean deleteCredential(URI credUid)`

This method deletes the credential with the given identifier from the credential store. If deleting is not possible (e.g. if the referred credential does not exist) the method returns false, and true otherwise.

Path	/user/deleteCredential/
HTTP Method	DELETE
Output Type	text/xml
Output Format	ABCEBoolean

Query Parameter	Parameter Type
credUid	URI

5.2 ABCE methods for Verifiers

`PresentationTokenDescription verifyTokenAgainstPolicy (PresentationPolicyAlternatives p, PresentationToken t, boolean store)`

This method, on input of a presentation policy `p` and a presentation token `t`, checks whether the token `t` satisfies the policy `p` and checks the validity of the cryptographic evidence included in token `t`. If both checks succeed and `store` is set to true, this method stores the token in a dedicated store and returns a description of the token that includes a unique identifier by means of which the token can later be retrieved from the store. If both checks succeed and `store` is set to false, this method does not store the token and returns the description of the token as-is. If one of the checks fails, this method returns a list of error messages.

Path	/verification/verifyTokenAgainstPolicy/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	PresentationPolicyAlternativesAndPresentationToken
Output Type	text/xml
Output Format	PresentationTokenDescription

Query Parameter	Parameter Type
store	String (“true” or “false)

`PresentationToken getToken(URI tokenUid)`

This method looks up a previously verified presentation token. The unique token identifier `tokenUid` is the identifier that was included in the token description that was returned when the token was verified.

Path	/verification/getToken/
HTTP Method	GET
Output Type	text/xml
Output Format	PresentationToken

Query Parameter	Parameter Type
tokenUID	URI

`boolean deleteToken(URI tokenUid)`

This method deletes the previously verified presentation token referenced by the unique identifier `tokenuid`. It returns true in case of successful deletion, and false otherwise.

Path	/verification/deleteToken/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	None
Output Type	text/xml
Output Format	Boolean

Query Parameter	Parameter Type
tokenUID	URI

5.3 ABCE methods for Issuers

`SystemParameters setupSystemParameters(int keyLength, URI cryptoMechanism)`

This method generates a fresh set of system parameters for the given key length, expressed as the bitlength of an asymmetric key (e.g., 1024 or 2048 bits). Issuers can generate their own system parameters, but can also reuse system parameters generated by a different entity. More typically, a central party (e.g., a standardization body) will generate and publish system parameters for a number of different key lengths that will be used by many Issuers. Security levels 1024 and 2048 MUST be supported; other values MAY also be supported.

Currently, the supported mechanism URIs are `urn:abc4trust:1.0:algorithm:idemix` for Identity Mixer and `urn:abc4trust:1.0:algorithm:uprove` for U-Prove.

Path	/issuer/setupSystemParameters/
HTTP Method	GET
Output Type	text/xml
Output Format	SystemParameters

Query Parameter Name	Query Parameter Type
securityLevel	int
cryptoMechanism	URI

`IssuerParameters setupIssuerParameters(IssuerParametersInput ipi)`

This method generates a fresh issuance key and the corresponding Issuer parameters. The issuance key is stored in the Issuer's key store, the Issuer parameters are returned as output of the method. The input to this method specify the maximal number of attributes `maxatts` that credentials issued with these parameters can contain, the system parameters `syspars`, the unique identifier `uid` of the generated parameters, the hash algorithm identifier `hash`, and, optionally, the parameters identifier for any Issuer-driven Revocation Authority.

Currently, the only supported hash algorithm is SHA-256 with identifier `urn:abc4trust:1.0:hashalgorithm:sha-256`.

Path	/issuer/setupIssuerParameters/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	IssuerParametersInput
Output Type	text/xml
Output Format	IssuerParameters

`(IssuanceMessage, boolean, URI) initIssuanceProtocol(IssuancePolicy ip, Attribute[] atts)`

This method is invoked by the issuer to initiate an issuance protocol based on the given issuance policy, where the given list of attributes contains the values that are to be certified in the new credential.

This method returns an issuance message, which the issuer subsequently sends to the user. Recall that all issuance messages contain a Context XML-attribute that uniquely references the instance of the issuance protocol.

This method also returns a boolean value, which indicates whether this is the last message of the issuance protocol. If this value is false, the issuer subsequently invokes the

`issuanceProtocolStep()` method on the next incoming issuance message from the user. If this value is true, the issuance protocol consisted of a single message: this is possible if the underlying credential technology requires no interaction for issuance (e.g., Camenisch-Lysyanskaya) and if simple issuance is used.

This method also returns the uid of the stored issuance log entry that contains an issuance token together with the attribute values provided by the issuer to keep track of the issued credentials.

Path	/issuer/initIssuanceProtocol/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	IssuancePolicyAndAttributes
Output Type	text/xml
Output Format	IssuanceMessageAndBoolean

`(IssuanceMessage, boolean, URI) issuanceProtocolStep(IssuanceMessage m)`

This method performs one step in the interactive issuance protocol. On input of an issuance message received from a user, it returns an issuance message that is to be sent back to the user, a boolean indicating whether this is the last message in the protocol, and (optionally) the uid of a newly-created issuance log entry.

Path	/issuer/issuanceProtocolStep/
HTTP Method	POST
Input Type	application/xml or text/xml
Input Format	IssuanceMessage
Output Type	text/xml
Output Format	IssuanceMessageAndBoolean

`IssuanceLogEntry getIssuanceLogEntry(URI issuanceEntryUid)`

This method looks up an issuance log entry of previously issued credentials that contains a verified issuance token together with the attribute values provided by the issuer. The issuance log entry identifier`issuanceEntryUid` is the identifier that was included in the issuance token description that was returned when the token was verified.

Path	/issuer/getIssuanceLogEntry/
HTTP Method	GET
Output Type	text/xml
Output Format	IssuanceLogEntry

Query Parameter Name	Query Parameter Type
issuanceEntryUid	URI

5.4 ABCE methods for Revocation Authorities

```
RevocationAuthorityParameters setupRevocationAuthorityParameters(int
keyLength, URI cryptoMechanism, URI uid, RevocationInfoReference infoRef,
NonRevocationEvidenceReference evidenceRef, RevocationUpdateReference updateRef)
```

For a given security level, expressed as the bitlength of an asymmetric RSA key, and revocation mechanism, this method generates a fresh secret key for the Revocation Authority and corresponding public Revocation Authority parameters, as well as the initial revocation information. The secret key is stored in trusted storage. Also included in the returned Revocation Authority parameters are the given identifier `uid` as well as the endpoints where Users, Verifiers and Issuers can obtain the latest revocation information (`infoRef`), initial non-revocation evidence (`evidenceRef`), and updates to their non-revocation evidence (`updateRef`). Security levels 1024 and 2048 MUST be supported; other values MAY also be supported.

Currently, the only supported revocation mechanism is based on Camenisch-Lysyanskaya accumulators (cryptoMechanism is `urn:idmx:3.0.0:block:revocation:cl`).

Path	/revocation/setupRevocationAuthorityParameters/
HTTP Method	POST
Input Type	Application/xml or text/xml
Input Format	RevocationReferences
Output Type	text/xml
Output Format	RevocationAuthorityParameters

Query Parameter	Parameter Type
keyLength	int
cryptoMechanism	URI
uid	URI

`NonRevocationEvidence generateNonRevocationEvidence(URI revParsUid, List<Attribute> attributes)`

This method creates up-to-date non-revocation evidence with respect to the given revocation authority parameters and the given list of attribute values.

In the special case of issuer-driven revocation, the list of attributes must contain exactly one item: the revocation handle. When the issuer calls this method during credential issuance, he leaves the attribute value of the revocation handle blank; the non-revocation evidence will then contain the attribute value to use in the new credential.

This method may also be queried by users who wish to update their non-revocation evidence. In contrast to the `generateNonRevocationEvidenceUpdate()` method, this method is potentially more efficient, but the user making the query will not be anonymous.

Depending on the revocation technology, this method may update the revocation information. If it does, verifiers may need to fetch the latest revocation information after this method is called in order to accept the non-revocation evidence in new credentials.

Path	/revocation/generatenonrevocationevidence/{revParsUid}
HTTP Method	POST
Input Type	Application/xml or text/xml
Input Format	AttributeList
Output Type	text/xml
Output Format	NonRevocationEvidence

Path Parameter	Parameter Type
revParsUid	URI

`NonRevocationEvidenceUpdate generateNonRevocationEvidenceUpdate(URI revParsUid, int epoch)`

This method generates information that allows a user to update the non-revocation evidence of one of her credentials. This will allow her to prove non-revocation of her credential against the latest revocation information. The user's anonymity is preserved when calling this method.

The inputs to this method specify the identifier of the revocation authority parameters corresponding to the non-revocation evidence to update; and the epoch of the revocation information that user's non-revocation evidence currently verifies against.

To ensure that the users' certificates are reasonably up-to-date, they will have to call this method for all their credentials at regular intervals (but at the latest when doing a presentation).

Path	/revocation/generatenonrevocationevidenceupdate/{revParsUid}
HTTP Method	POST
Output Type	text/xml
Output Format	NonRevocationEvidenceUpdate

Path Parameter	Parameter Type
revParsUid	URI

Query Parameter	Parameter Type
epoch	int

`RevocationInformation updateRevocationInformation(URI revParsUid)`

This method retrieves the latest revocation information associated with the given revocation authority.

To ensure that verifiers can detect revoked certificates in a timely manner, they will call this method on all revocation authorities they know at regular intervals.

Path	/revocation/getrevocationinformation/{revParsUid}
HTTP Method	POST
Output Type	text/xml
Output Format	RevocationInformation

Path Parameter	Parameter Type
revParsUid	URI

`RevocationInformation revoke(URI revParUid, List<Attribute>attributes)`

This method revokes the attribute values specified by the given list of attributes with respect to the given revocation authority parameters. If the list contains multiple attributes (and if the revocation technology supports this), then the conjunction of these attribute values is revoked. That is, all credentials that contain the combination of attribute values specified in the list are revoked.

In the special case of issuer-driven revocation, the list contains only a single attribute: the revocation handle.

Verifiers have to obtain the latest revocation information from the respective revocation authority in order to detect that the given combination of attributes was revoked.

Path	/revocation/revoke/{revParsUid}
HTTP Method	POST
Input Type	Application/xml or text/xml
Input Format	AttributeList
Output Type	text/xml
Output Format	RevocationInformation

Path Parameter	Parameter Type
revParsUid	URI

5.5 ABCE methods for Inspectors

`InspectorPublicKey setupInspectorPublicKey(int keyLength, URI mechanism, URI uid)`

This method generates a fresh decryption key and corresponding encryption key for the given security level, expressed as the keylength of an asymmetric RSA key with comparable security, and cryptographic mechanism. It stores the decryption key in the trusted storage and returns the inspector public key with the given identifier `uid`. The identifier associated with the key will be used in presentation/issuance policies as the unique reference to a particular Inspector.

Security levels 1024 and 2048 MUST be supported; other values MAY also be supported. The only currently supported mechanism identifier is `urn:abc4trust:1.0:inspectionalgorithm:camenisch-shoup03`.

Path	/inspector/setupInspectorPublicKey/
HTTP Method	POST
Output Type	text/xml
Output Format	InspectorPublicKey

Query Parameter	Parameter Type
keyLength	int
cryptoMechanism	URI
uid	URI

`Attribute[] inspect(PresentationToken t)`

This method takes as input a presentation token with inspectable attributes and returns the decrypted attribute type-value pairs for which the Inspector has the inspection secret key.

Path	/inspector/inspect/
HTTP Method	POST
Input Type	Application/xml or text/xml
Input Format	PresentationToken
Output Type	text/xml
Output Format	AttributeList

6 Crypto Architecture

6.1 Overview of Cryptographic Architecture

The main responsibilities of the Cryptographic Engine are to generate cryptographic key material, issue new credentials by means of a two-party protocol, generate the cryptographic evidence for a Presentation Token to prove that a user satisfies a Presentation Policy, and verify this evidence.

The Cryptographic Engine is shipped as a separate library and can operate without the Privacy-ABC Engine. It replaces version 2 of IBM's Identity Mixer (Idemix) Library. Its main advantage compared to the old Idemix library is the increased modularity of its design. This modularity allowed us to implement additional features, such as supporting U-Prove credentials, and a predicate for checking linear combinations among attributes.

Structure In the Crypto Engine, we have made a clear distinction between the *building blocks*, which implement the actual cryptographic algorithms, and the *framework* code, which is mostly cryptography-agnostic. The Building Blocks interact with the framework and with each other through implementation-agnostic interfaces. This clean separation allows one to easily substitute one implementation of a cryptographic primitive with another—or to provide a new implementation of a existing cryptographic primitive—and only minimally affect the framework code.

The framework comprises the following components:

- The Key Generation Orchestration, responsible for generating cryptographic key material.
- The Proof Generation Orchestration, which, with the help of the Proof Engine, is responsible for generating the cryptographic evidence of a Presentation Token.
- The Proof Verification Orchestration, which, with the help of the Proof Engine, is responsible for verifying the cryptographic evidence contained in a Presentation Token.
- The Issuance Orchestration, which is responsible for the whole process of issuing a credential. It also uses the Proof Engine. This component operates in two modes: Issuer and Recipient.
- A Proof Engine, which is tasked with generating, and later verifying, a non-interactive zero-knowledge proof. This component also operates in two modes: Prover and Verifier.
- A Building Block Factory, which keeps track of all known building blocks and is responsible for returning the appropriate block or list of blocks for a given task.
- State Storage, for keeping the intermediate state during the issuance protocol.

The framework of the Crypto Engine also accesses several other components of the Privacy-ABC Engine, such as the Credential Manager, the Key Manager, the Smartcard Manager, and the Revocation Proxy. In case the Crypto Engine runs without a Privacy-ABC Engine, an alternative implementation of these components must be provided.

In what follows, we describe how the various Orchestration components work. We then describe the Proof Engine and the proof interface of the Building Blocks.

6.1.1 Key Generation Orchestration

Let us describe the generation of parameters/keys such as the System Parameters, the Issuer Key Pair, Inspector Key Pair, and Revocation Authority Key Pair. For the Crypto Engine, this is stateless two-step process: first, upon receiving a request from a user, the Key Generation (KG-) Orchestration generates a *Configuration Template* and returns it to the user; second, the user submits the completed configuration to the KG-Orchestration, which then initiates the generation of the actual parameters/keys.

In Figure 7 we depict the setup of an Issuer Key Pair as an example of the parameter/keys generation process. The generation of System Parameters, or of a key pair of another entity is similar.

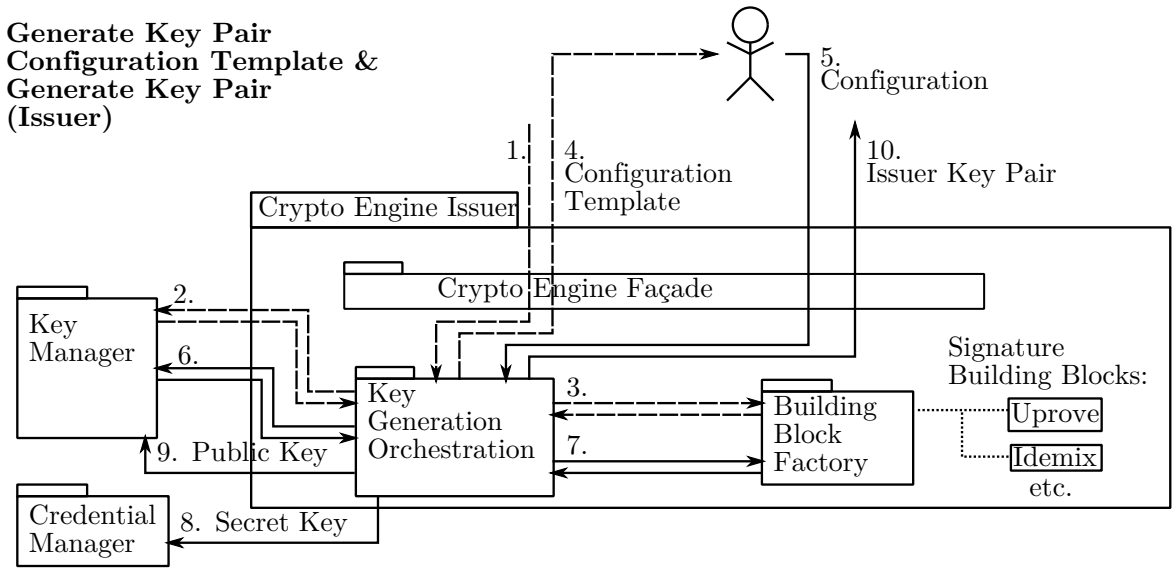

Figure 7: Example of generating an Issuer Key Pair including the creation of the intermediate Key Pair Configuration Template.

Template Generation The user starts by requesting a Key Pair Configuration Template (1) from the Crypto Engine. This request is forwarded to the Key Generation Orchestration. The latter requests the System Parameters from the Key Manager (2) as well as all signature Building Blocks from the Building Block Factory (3). Further, it adds a few default entries to the Configuration Template, and asks each signature Building Block in turn to add its own implementation-specific entries to the Configuration Template. The Configuration Template is then returned to the user (4).

A template is configured with default values that serve as suggestion for a general purpose use; the actual settings (including which implementation of a specific cryptographic primitive) must be set manually and in accordance with the planned use. While all implementations of a given cryptographic primitive can add parameters to the template; the user only needs to fill out the general entries and the entries corresponding to the chosen implementation—the entries corresponding to non-chosen implementations will be ignored.

Parameter Generation After completing the configuration by overriding the appropriate default settings of the Template Configuration, the Crypto Engine must be called again to request the generation of an Issuer Key Pair (5). The KG-Orchestration queries the Key Manager for the System Parameters again (6), and the Building Block Factory for the chosen implementation of the signature Building Block (7). It then asks that Building Block to generate an Issuer Key Pair based on the configuration. It then stores the secret key of the pair in the Credential Manager (8), and the public key in the Key Manager (9), before returning the whole key pair (10).

6.1.2 Presentation Orchestration

Let us now illustrate the generation of a Presentation Token as depicted in Figure 8.

When a user wants to create a Presentation Token she needs to pass the Presentation Token Description, a list of credential URIs, and a list of pseudonym URIs to the Crypto Engine (recall that the credential and pseudonym URIs have meaning only on the user's machine, and might de-anonymize the user if exposed;

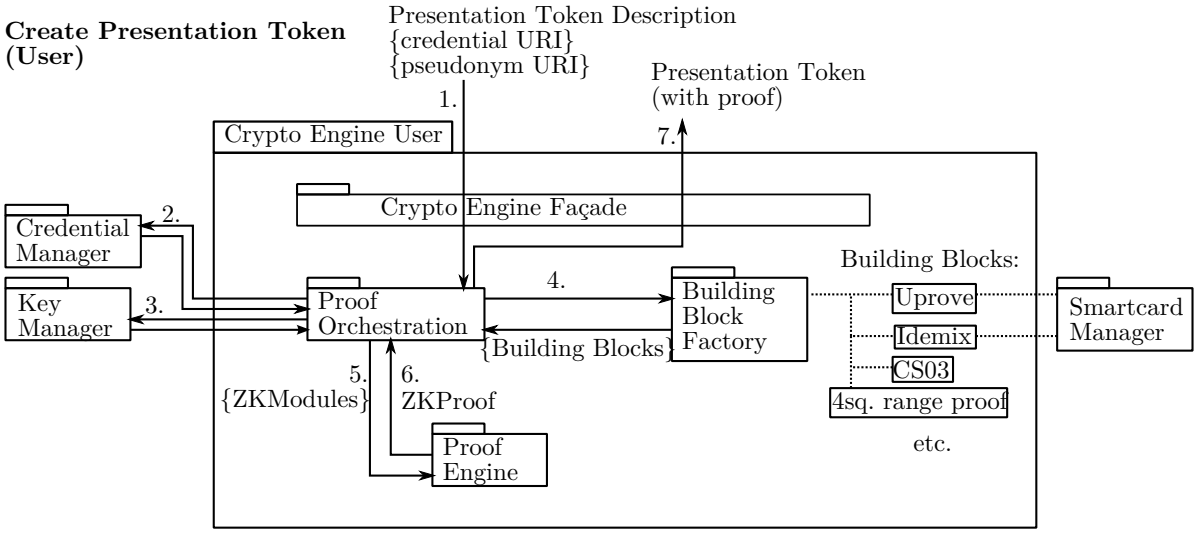

Figure 8: Creation of a Presentation Token.

these URIs must therefore not be included in the Presentation Token Description). These elements get forwarded to the Proof Orchestration (1). The Proof Orchestration first fetches the credentials and pseudonyms based on their URI from the Credential Manager (2). Second, it loads the System Parameters, Issuer Public Keys, Credential Templates, Inspector Public Keys, and Revocation Authority Public Keys from the Key Manager (3). Third, it queries the Building Block Factory for the Building Blocks required for the Presentation Token at hand (4). For Building Blocks that have several implementations, the Proof Orchestration may mandate a specific implementation, or it may ask the Building Block Factory for any implementation that is supported by the verifier. For example, in Figure 8, the building blocks for U-Prove and Identity Mixer signatures are displayed, as well as Camenisch-Shoup verifiable encryption and the four-squares range proofs. In any case, the prover must record his choice of implementation so that the verifier can retrieve the exact same Building Block.

The Proof Orchestration asks these Building Blocks each generate one or more Zero-knowledge–proof Modules (*ZkModules*) (5), and configures each ZkModule with the appropriate parameters such as the keys, credentials, or pseudonyms. These ZkModules will be used later inside the Proof Engine. Each ZkModule will independently perform one part of the overall zero-knowledge proof and encapsulates needed algorithms and state while exposing a uniform interface to the Proof Engine. We point out that ZkModules responsible for proving ownership of a credential or a pseudonym receive a reference to the Smartcard Manager, as they may delegate part of the proof process to the Smartcard Manager, which in turn interacts with a smartcard to generate the proof elements needed during proof creation.

The Proof Orchestration then asks the Proof Engine to generate a Zero-knowledge Proof (6) supporting the validity of the Presentation Token based on this list of ZkModules. The Proof Orchestration finally updates the Presentation Token Description and then combines the former with the Zero-knowledge Proof to form the final Presentation Token (7).

6.1.3 Verification Orchestration

In Figure 9 we show the verification of a Presentation Token.

After the verifier has matched the Presentation Token to the Presentation Policy, he sends the Presentation Token to the Crypto Engine for cryptographic verification (1). The Crypto Engine forwards the Presentation Token to the Proof Verification (PV-) Orchestration. The PV-Orchestration fetches the relevant System Parameters, Issuer Public Keys, Credential Templates, Inspector Public Keys, and

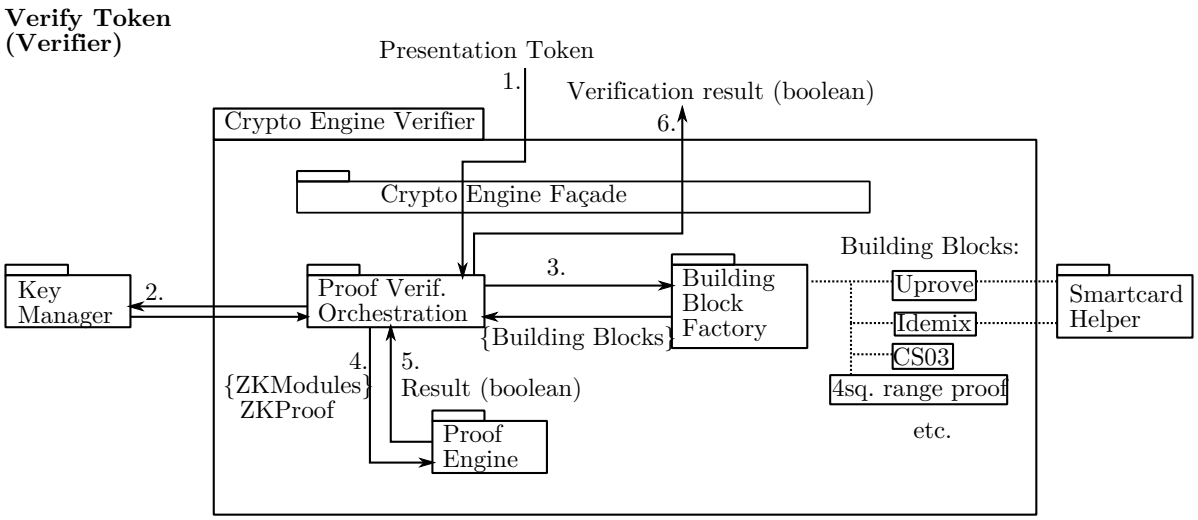

Figure 9: Verification of a Presentation Token.

Revocation Authority Public Keys from the Key Manager (2). It then needs to fetch the same set of Building Blocks from the Building Block Factory (3) as the prover did. Thereafter, it can generate a list of ZkModules using these Building Blocks, where the ZkModules correspond to the ones generated by the prover (4). The ZkModules for credentials and pseudonyms receive a reference to a Smartcard Helper, which provides the functionality required to verify the part of the proof generated by a smartcard. All ZkModules together with the Zero-knowledge Proof in the token are sent to the Proof Engine for verification. The result of the verification (5) is then forwarded to the verifier (6).

6.1.4 Issuance Orchestration

We now describe the issuance protocol in the case of advanced issuance of a revocable credential where the signing of the credential needs only one round (as is the case for CL signatures) and where no jointly-random attributes are present. We point out that the issuance protocol continues for as many rounds as the used signature building block specifies. Figures 10 and 12 show the issuance process on the issuer's side and Figures 11 and 13 show the process on the recipient of the credential's side.

Issuer: First Issuance Message with Issuance Policy The first step of the issuance protocol is shown in Figure 10.

The issuer invokes the Crypto Engine with an Issuance Policy and a list of issuer-set attributes (1). The Crypto Engine forwards those to the Issuance Orchestration. The Issuance Orchestration saves the Issuance Policy and list of attributes in the State Storage (2), wraps the Issuance Policy in an Issuance Message, and returns that message to the issuer (3). The issuer should then transmit the message to the recipient.

Recipient: Generate Issuance Token The second step of the issuance protocol is shown in Figure 11.

The recipient must choose how to satisfy the Issuance Policy contained in the issuer's Issuance Message similar to when performing a presentation. The recipient then calls the Crypto Engine with the Issuance Token Description, a list of credential URIs, a list of Pseudonym URIs, and the original Issuance Message (4). These elements are forwarded to the Issuance Orchestration. The latter first checks with the State

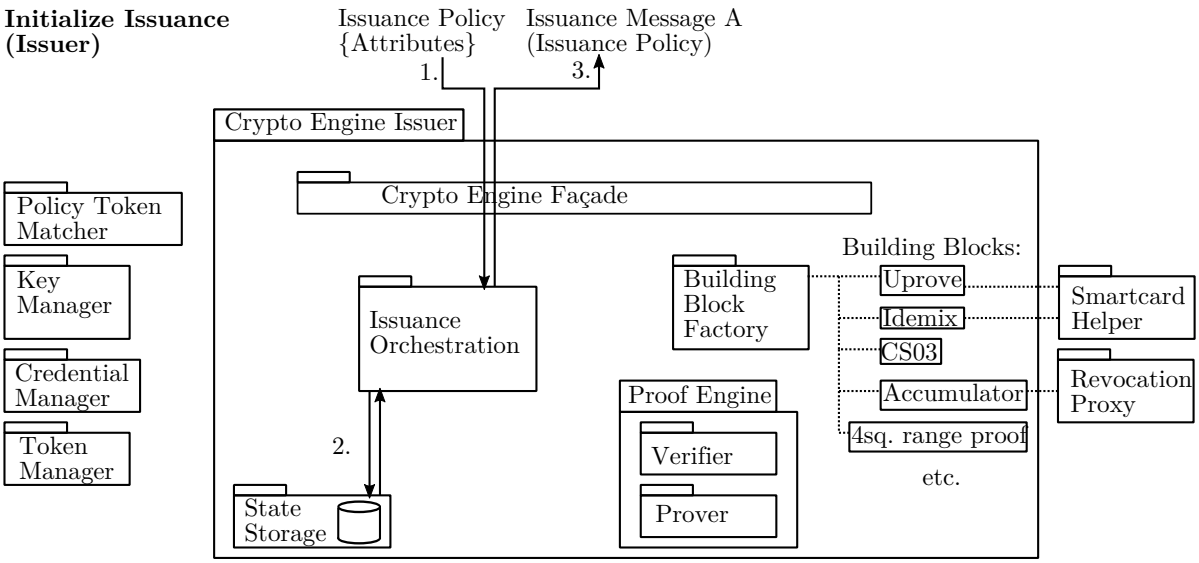

Figure 10: Initiation of the issuance protocol on the issuer's side.

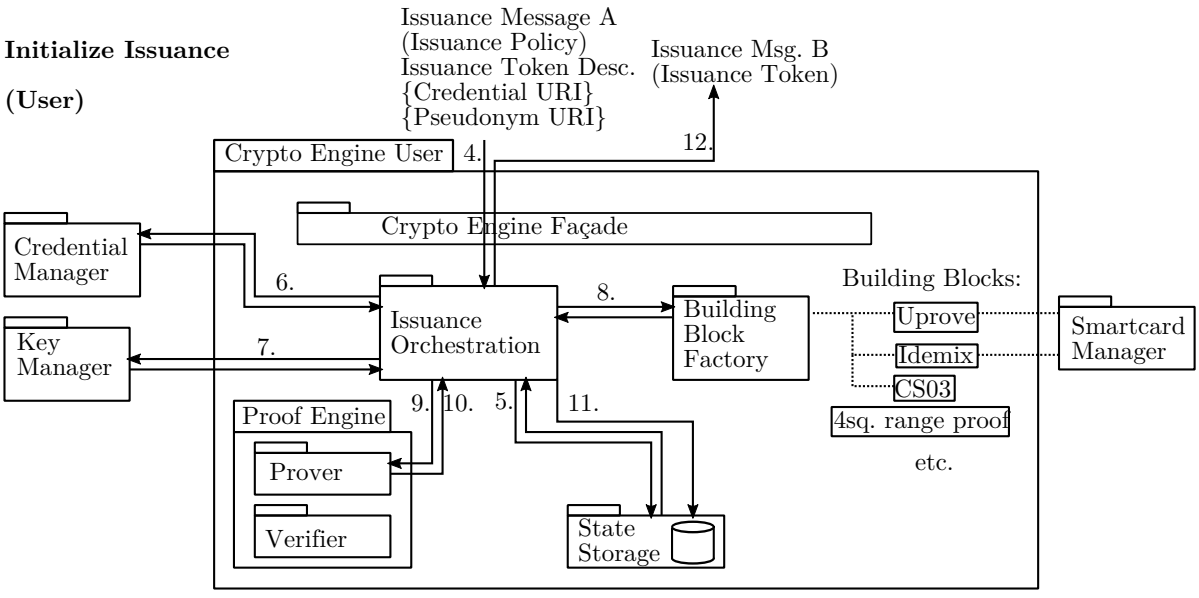

Figure 11: Recipient computes an Issuance Token proving properties used for the credential to be issued.

Storage that the Issuance Context (contained in the Issuance Message) has never been seen before (5). Steps (6) to (10) are similar to a presentation proof (see Section 6.1.2), with the exception that the Issuance Orchestration additionally generates a ZkModule from the signature building block that enables the carry-over of attributes.

The Issuance Orchestration also retrieves state pertaining to the carry-over of attributes from the aforementioned ZkModule after the Proof Engine finished the proof generation. It then completes the Issuance Token Description with data generated during the proof, generates an Issuance Token from the proof and the Issuance Token Description, wraps the Issuance Token in an Issuance Message, saves its current state in the State Storage (11), and returns the Issuance Message to the recipient (12).

Issuer: Create Signature The third step of the issuance protocol is shown in Figure 12.

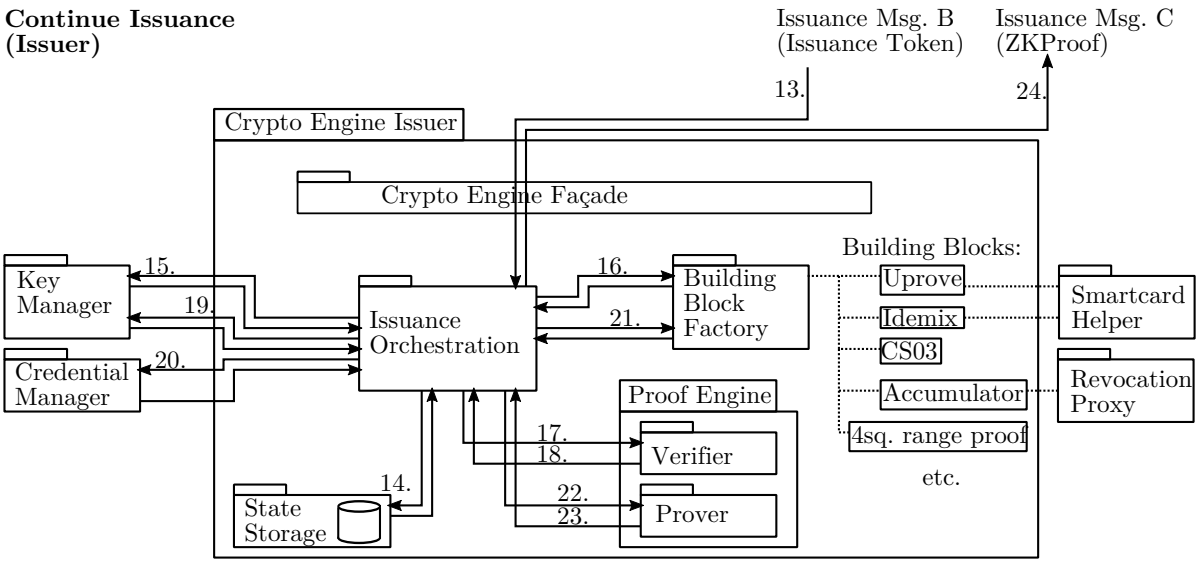

Figure 12: Issuer creates signature.

The issuer should forward the recipient's Issuance Message (containing the Issuance Token) to his Crypto Engine directly. The latter forwards it to the Issuance Orchestration (13). The Issuance Orchestration first recovers the state associated with the Issuance Context from the State Storage (14). Then, it checks the proof contained in the recipient's Issuance Token similar to the PV-Orchestration (see Section 6.1.3) (15)–(18).

If the verification succeeded, the Issuance Orchestration then recovers the Revocation Authority's Public Key from the Key Manager (19), the issuer's Secret Key from the Credential Manager (20), and a Building Block for revocation of the correct implementation from the Building Block Factory (21). It then recovers state from the ZkModule for carry-over and uses that state to initialize a ZkModule for issuance from the signature Building Block; that ZkModule is also initialized with the issuer-set attributes, and the Issuer Secret Key. It also generates a ZkModule for issuance from the Building Block for revocation. During creation time, the ZkModule contracts the Revocation Authority (through the Revocation Proxy) to retrieve a new Revocation Handle and the associated Non-revocation Evidence. The Issuance Orchestration then passes these two ZkModules to the Proof Engine (22).

During the proof generation, the ZkModule for signature issuance will blindly sign the new credential. The Proof Engine returns the created zero-knowledge proof to the Issuance Orchestration (23). This zero-knowledge proof also contains the issuer's blind signature on the credential, the issuer-set attributes,

the value of the Revocation Handle, and the Non-revocation Evidence. The Issuance Orchestration then queries the ZkModule for signature issuance for the list of attribute values it knows about (including the revocation handle), and generates an issuance log entry containing that list. Finally, it wraps the zero-knowledge proof into an Issuance Message, and returns it to the issuer (24).

Recipient: Complete Credential The last step of the issuance protocol is shown in Figure 13.

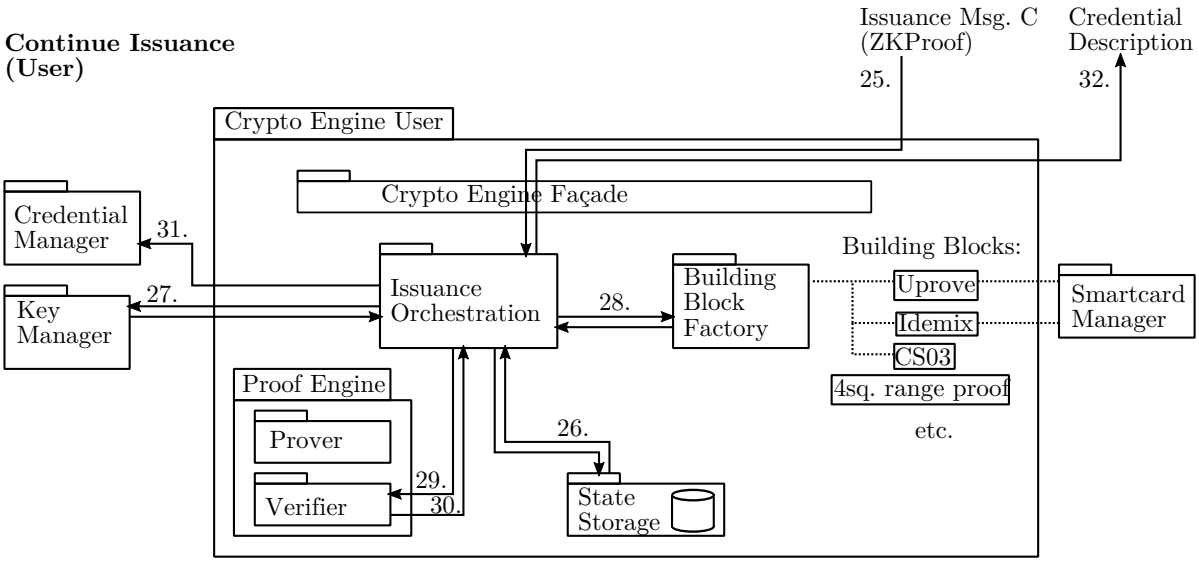

Figure 13: Recipient finishes signature and stores credential.

The recipient should forward the issuer's Issuance Message (containing the zero-knowledge proof) to her Crypto Engine directly. The latter forwards it to the Issuance Orchestration (25). The Issuance Orchestration first recovers the state associated with the Issuance Context from the State Storage (26), retrieves the necessary parameters, specifications, and keys from the Key Manager (27). It then queries for a Building Block for signatures and a Building Block for revocation of the appropriate implementation from the Building Block Factory (28).

The Issuance Orchestration creates a ZkModule for signature issuance from the first Building Block, initializing it with the issuer's Public Key and state from the ZkModule for carry-over from last round. It also creates a ZkModule for revocation issuance from the second Building Block. It then sends these two ZkModules and the zero-knowledge proof to the Proof Engine for verification (29). After the Issuance Orchestration gets back the results of the proof verification (30), it extracts the issuance state from the ZkModule for signature issuance. It asks the signature Building Block to combine the states of the ZkModule for issuance and the ZkModule for carry over from last round to generate the complete credential, including signature and Non-revocation Evidence. It then saves the credential in the Credential Manager (31) and returns the Credential Description to the recipient (32).

6.1.5 Building Blocks

The Building Blocks are singleton classes that implement the actual cryptographic algorithms. We have defined an interface for Building Blocks for each of the cryptographic primitives (signatures, pseudonyms, inspection, revocation, range proofs, not-equal proofs, set-membership proofs) plus a few interfaces for helper Building Blocks (e.g., reveal attribute, attribute equality, add message to proof). There may be

several implementations of a given cryptographic primitive (e.g., CL-signatures and U-Prove signatures), but the Building Blocks corresponding to the various implementations expose the same interface to the rest of the library. This strong encapsulation ensures the modularity of the library.

The Building Block for signatures for example has the following interfaces:

- Functions to populate a template for an Issuer Key Pair and to generate an Issuer Key Pair.
- One *proof interface* for presenting the signature (i.e., proving possession of the signature).
- One proof interface for carry-over, in which it is proven that a freshly generated commitment contains all the user-specified or carried over attributes, and which allows the prover and verifier to extract that commitment and later use the commitment for issuance.
- One proof interface for issuance, in which it is proven that a blind signature was performed correctly, and which allows the verifier to extract the blind signature.
- Functions to continue with the issuance process after the issuance proof (used only for implementations that have a multi-step issuance process, like U-Prove).
- A function that extracts a complete signature from the issuance proof.
- Bookkeeping functions, for example one that returns the identifier of the cryptographic primitive, and one that returns the name of the implementation.

Other Building Blocks have interfaces that are adapted to the needs of the specific cryptographic primitive they implement. The interface of helper Building Blocks typically comprises only a single proof interface and the bookkeeping functions.

Proof Interfaces and ZkModules The proof interface of a Building Block consists of two functions: one factory for so called prover *ZkModule* (zero-knowledge–proof modules) objects and one factory for verifier ZkModule objects. These ZkModules are the actual objects that are sent to the Proof Engine. Each ZkModule is responsible for performing one part of a zero-knowledge proof (for example the proof of a single cryptographic commitment) without needing to explicitly care about interaction with other ZkModules—it the Proof Engine's responsibility to coordinate the ZkModules behind the scenes.

All prover ZkModules in the library expose the same interface towards the Proof Engine, allowing the latter to handle them uniformly. (Some specialized ZkModules also have additional functions, for example to retrieve values after the proof is completed, but those functions are not visible to the Proof Engine.) This interface consists of four callback functions that are called sequentially by the Proof Engine during the course of the proof:

- `initializeModule`, in which the ZkModule must tell the ZkBuilder (a component of the Proof Engine) the name of all attributes it intends to use, the acceptable range of values each attribute can take, whether each attribute should be revealed or not, whether it knows the value of that attribute or not, whether this attribute is an external attribute (i.e., one which resides on a smartcard), and whether it needs to know the value of some other attribute (provided by another ZkModule) before it can set the value of that attribute. In this phase, the ZkModule may also declare that an attribute is equal to another attribute (possibly an attribute that appears in another ZkModule).
- `collectAttributes`, in which the ZkModule must provide the value of all attributes for which it knows that value (and where the ZkModule is allowed to query for the value of the attributes it requested in the initialize phase).
- `firstRound`, in which the ZkModule must help the ZkBuilder perform the first phase of the zero-knowledge proof, meaning, determine all the values that will be used to compute the challenge: the ZkModule can ask if any of its attributes is revealed, query for the value of all revealed attributes, and query for the R-Value (randomizers—see Table 6) of all unrevealed non-external attributes; and must generate T-Values (commitment values). At this point the ZkModule may also add data to its hash contribution by adding D-Values (which are delivered to the verifier) or N-Values (which are not delivered to the verifier as he is supposed to know the value already).

- `secondRound`, in which the ZkModule receives the value of the challenge from the ZkBuilder and must provide the S-Value (response values) for all external attributes.

Table 6: Glossary for values inside a zero-knowledge proof. More details are given in Section 6.2.2.

Name	Explanation
R-value	Randomizers. The random values that replace the attribute values when computing the T-values.
T-value	Commitment values. Values that are derived from the R-Values and the statement to be proven, and which will be used to compute the challenge (together with the D-values, N-values, and revealed attributes).
D-value	Delivered values. Values that are sent to the verifier together with the proof, and which are also used to compute the challenge.
N-value	Context values. Values that the prover and the verifier agree on during the proof and that don't need to be transmitted to the verifier. These values are also used to compute the challenge.
S-value	Response values. Values that are computed based on the challenge.

Similarily, all verifier ZkModules expose the same interface towards the Proof Engine. This interface consists of two callback functions that are called sequentially by the Proof Engine during the course of a proof verification:

- `initializeModule`, in which the ZkModule must tell the ZkVerifier (a component of the Proof Engine) the name of all attributes it intends to use, the acceptable range of values each attribute can take, whether each attribute should be revealed or not, and whether it knows the value of that attribute or not. In this phase, the ZkModule may also declare that an attribute is equal to another attribute (possibly an attribute that appears in another ZkModule). The verifier ZkModule must make the equivalent calls as the corresponding prover ZkModule in this function.
- `verify`, in which the ZkModule receives the value of the challenge, the S-Values of all of its attributes, and the value of all revealed attributes; and must re-compute the T-Values. The ZkModule must also provide the N-Values and may perform additional checks in this function (for example by doing implementation-specific checks on the S-Values and D-Values).

6.1.6 Proof Engine

The Proof Engine is responsible for orchestrating the construction of a zero-knowledge proof following the Fiat-Shamir heuristic on input a list of ZkModules. We designed the Proof Engine according to the Builder pattern: the zero-knowledge proof is build step-by-step by the ZkBuilder (the builder in the Builder pattern) following the directions of a ZkDirector class and of the individual ZkModules (both collectively fulfilling the role of the director in the Builder pattern).

In Figure 14, we show the sequence diagram of the construction of a proof in the Proof Engine. The ZkDirector's role is simply to call the methods of the ZkModules and the ZkBuilder in the right order:

- First, it calls `initializeModule` on all ZkModules (with a reference to the ZkBuilder), so that the latter can register their attributes with the ZkBuilder. The ZkBuilder needs to keep track of which attributes are equal (and handle all equivalences implied by transitivity); and for each disjoint set of attributes, it needs to keep track of the properties, such as acceptable range, whether the attributes are external. The ZkBuilder will later also need to keep track of the attribute values, R-Values and S-Values.
- Second, if some ZkModules need to know the value of attributes of other ZkModules, the ZkDirector asks the ZkBuilder to topologically sort the ZkModules.
- Third, the ZkDirector calls `collectAttributes` on all ZkModules. In this phase, the ZkBuilder will learn the value of all non-external attributes.

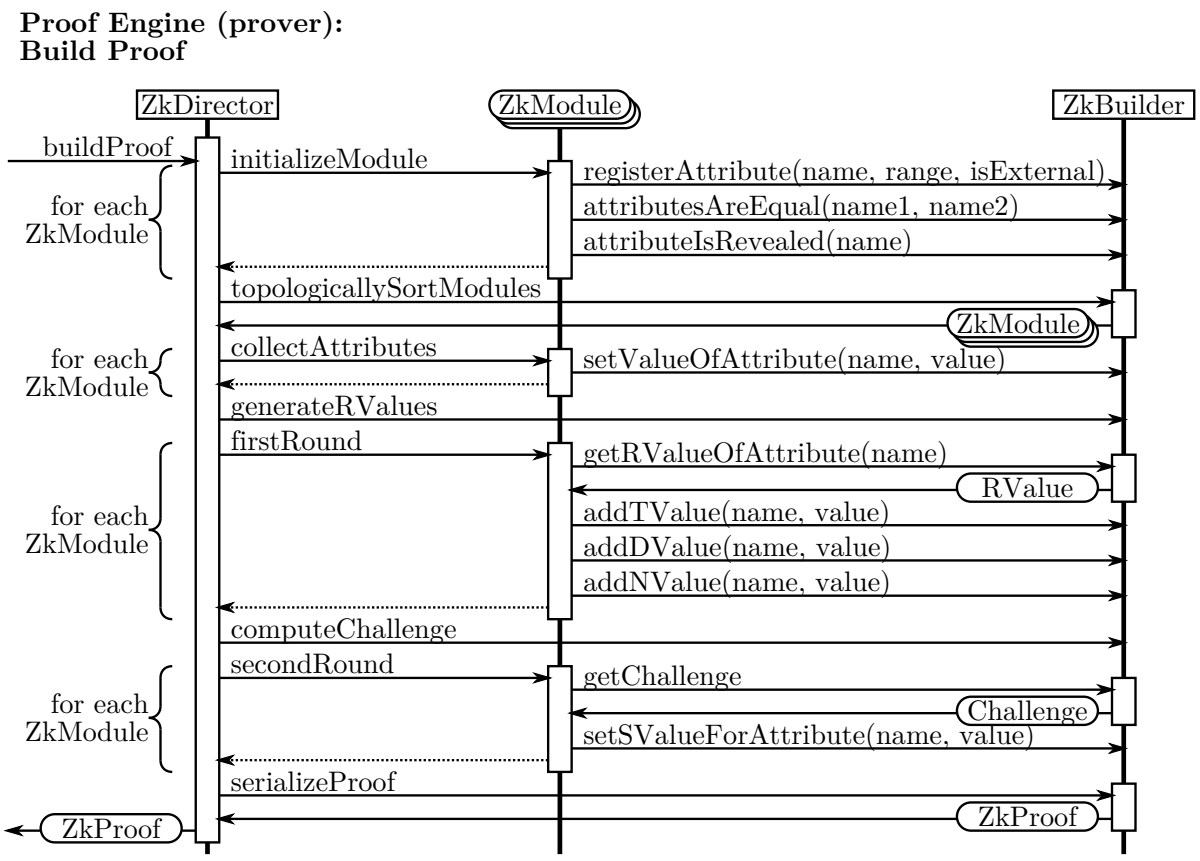

Figure 14: Sequence diagram for the construction of a proof in the Proof Engine.

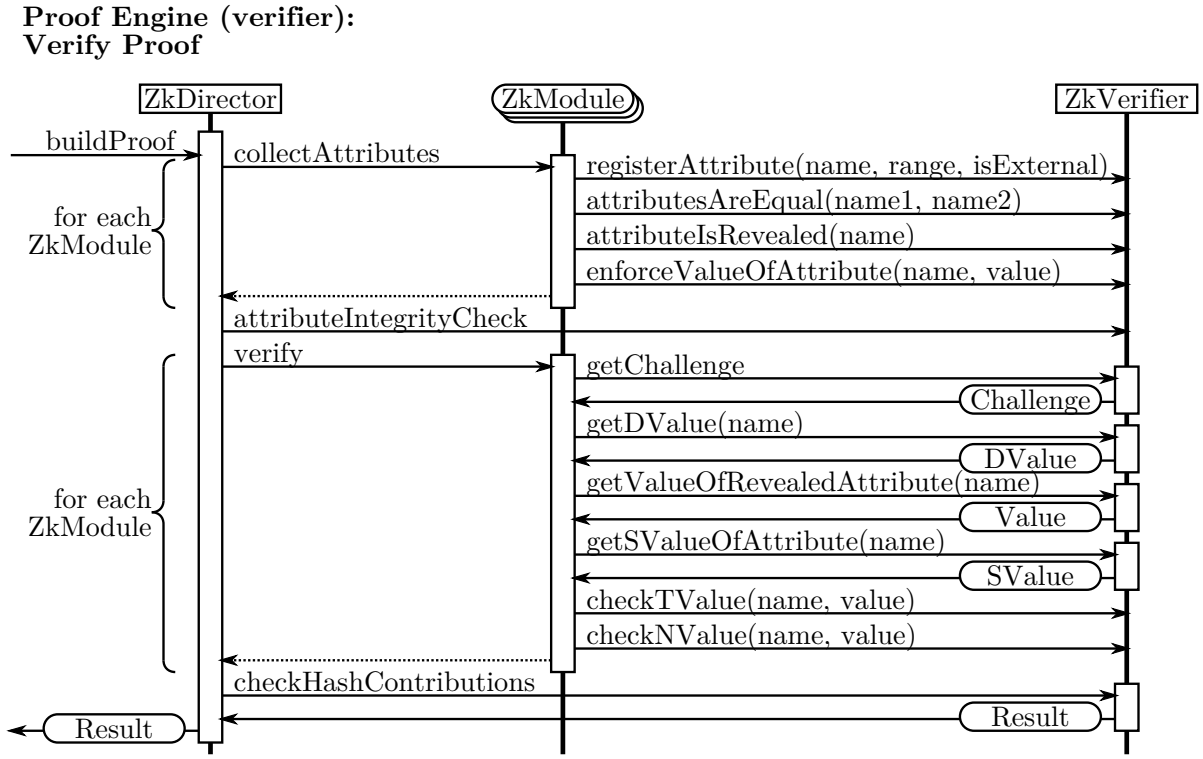

Figure 15: Sequence diagram for the verification of a proof in the Proof Engine.

- Fourth, it asks the ZkBuilder to compute R-Values for all unrevealed non-external attributes.
- Fifth, it calls `firstRound` on all ZkModules. In this phase, the ZkBuilder learns the T-Values of all equations in the proof and collects the D-Values and N-Values from the ZkModules.
- Sixth, it asks the ZkBuilder to compute the value of the challenge of the proof. This computation requires two steps: the ZkBuilder computes a *hash contribution* for each ZkModule that includes all the T-, N-, and D-Values (including revealed attributes) used by that ZkModule; and finally the overall challenge is computed by hashing all hash contributions. After the ZkBuilder computed the challenge, it also computes the S-Value of all unrevealed non-external attributes.
- Seventh, it calls `secondRound` on all ZkModules. In this phase, the ZkModules tell the ZkBuilder the S-Values of all external attributes.
- Finally, it asks the ZkBuilder to build the zero-knowledge proof object from: the list of ZkModule names, the list of ZkModule hash contributions, the list of D-Values (including revealed attributes), and the list of S-Values.

The Proof Engine uses a similar construction for verifying a proof. In Figure 15, we show the sequence diagram for the verification of a proof in the Proof Engine. The ZkDirector does the following:

- First, it calls `collectAttributes` on all ZkModules, so that the latter can register their attributes with the ZkVerifier. The ZkVerifier needs to re-construct a similar attribute database as the ZkBuilder during the construction of the proof. We note that for revealed attributes, the ZkModules may choose to request a specific value of an attribute—if no ZkModule provides such a value, the corresponding D-Value from the proof object is taken.
- Second, it asks the ZkVerifier to check that all S-Values are within their acceptable range. At this point, the ZkVerifier also re-computes the value of the challenge from the list of hash contributions.

- Third, it calls `verify` on all ZkModules. The ZkModules now have access to the value of the challenge, the D-Values, and the S-Values; and must re-compute the T-Values and provide the N-Values to the ZkVerifier. ZkModules may also perform additional checks with the D-Values and S-Values (e.g., for the U-Prove signatures, the verifier must check that the U-Prove token —a D-Value—sent by the prover is valid).
- Finally, it asks the ZkVerifier to check the hash contributions of all ZkModules. If the list of re-computed hash contributions matches the list in the proof, the ZkVerifier reports that the proof verification was successful.

6.2 Cryptographic Primitives

After describing the cryptographic architecture implemented within this project, we next give a detailed summary of the diverse building blocks that are used. In particular, all the building blocks mentioned in Figures 8, 9, 10 and 11 will be discussed in the following.

For each building block, we give a high-level description of the primitive and the security properties that need to be satisfied, as well as a cryptographic description of the instantiations we use. However, we refrain from giving implementation specific details for any of the building blocks, but refer the interested reader to the documentation of the implementation [BCD+13].

6.2.1 Algebraic Background

We now briefly explain the mathematical background that is required for the rest of this chapter, as well as the cryptographic hardness assumptions that underly the security proofs of the given instantiations. Below, let $\mathbb{Z}_n = \{0, \ldots, n-1\}$ denote the set of integers modulo n, and let $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : \gcd(x, n) = 1\}$.

Groups The concept of groups is central for all the primitives and protocols presented in the following. Informally, a group is a set of elements, on which one can operate as one is used to from addition on the integers: combining two elements yields another element of the group (the sum of two integers is an integer), the order in which elements are combined does not matter (parentheses are not important for addition), there is an element which does not change the value of any other element when combined with that element (adding zero to any integer yields the very same integer), and for every element there is an inverse element (there exists the negative inverse for every integer).

The following now formally defines this idea:

Definition 6.1. A pair $(\mathcal{G}, \otimes)$, where $\mathcal{G}$ is a set and $\otimes$ is a binary operation, is called a *group* if the following properties are satisfied:

Closure. For all $a, b \in \mathcal{G}$ the result $a \otimes b \in \mathcal{G}$.

Associativity. For all $a, b, c \in \mathcal{G}$ it holds that $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.

Identity element. There exists $e \in \mathcal{G}$ such that for all $a \in \mathcal{G}$ it holds that $a \otimes e = e \otimes a = a$.

Inverse element. For all $a \in \mathcal{G}$ there exists an element $b \in \mathcal{G}$ such that $a \otimes b = b \otimes a = e$, where e is the identity element.

Now and in the following, we will typically omit the binary operation when denoting the group, i.e., we will just write $\mathcal{G}$ instead of $(\mathcal{G}, \otimes)$.

In cryptography, we are mainly concerned with finite groups, i.e., groups where $\mathcal{G}$ only contains a finite number of elements, which we will assume for the rest of this chapter.

A group is called *cyclic* if there exists an element $a \in \mathcal{G}$ such that any element $b \in \mathcal{G}$ can be written as $b = a \otimes \cdots \otimes a = a^n$ for some positive integer n. In this case, a is called a *generator* of $\mathcal{G}$. The smallest positive integer such that $e = a^n$, where e is the identity element, is called the *order of* a, and the number of elements in $\mathcal{G}$ is called the *order of* $\mathcal{G}$.

Finally, for two elements $a, b \in \mathcal{G}$ we say that an integer n is the *discrete logarithm of b in base a*, if it holds that $b = a^n$.

Hardness Assumptions The security of most cryptographic primitives cannot be proved directly using information-theoretic arguments, but can only be proved assuming that solving some mathematical task is computationally infeasible. Here, computationally infeasible means that no algorithm whose running time is bounded by a polynomial in the length of its input, can solve the given task with more than negligible probability, where a function is negligible if it vanishes faster than any inverse polynomial.

The following hardness assumptions have been analyzed for decades, and are widely believed to be satisfied for the groups that we are going to use in the following descriptions.

Definition 6.2. Let $\mathcal{G}$ be a cyclic group, and let a be a generator of $\mathcal{G}$. Given a random $b \in_R \mathcal{G}$, the *discrete logarithm problem* is to compute the discrete logarithm of b in base a, i.e., to find an integer n such that $b = a^n$. The *discrete logarithm assumption* holds for $\mathcal{G}$ if no efficient (i.e., polynomial time) algorithm can solve the discrete logarithm problem with more than negligible probability.

Definition 6.3. Let $\mathcal{G}$ be a cyclic group of order q, and let a be a generator of $\mathcal{G}$. Let further be $x, y, z \in_R \mathbb{Z}_q$. The *decisional Diffie-Hellman problem* is to distinguish (a, a^x, a^y, a^z) from (a, a^x, a^y, a^{xy}). The *decisional Diffie-Hellman assumption* (DDH) holds for $\mathcal{G}$ if no efficient (i.e., polynomial time) algorithm can solve the decisional Diffie-Hellman problem with non-negligibly higher success probability than 1/2.

The next assumption is related to factoring large integers, and is also commonly believed to be computationally hard. It is a generalization of the RSA assumption [RSA78] and was introduced by Fujisaki and Okamoto [FO97] and Barić and Pfitzmann [BP97].

Definition 6.4. Let n be a random safe RSA modulus, i.e., $n = pq$ where $p := 2p' + 1, q := 2q' + 1$ and p, q, p', q' are all primes, and p and q are about the same size. Then the *strong RSA problem* is to find, given n and a random $b \in_R \mathbb{Z}_n^*$, an element $a \in_R \mathbb{Z}_n^*$ and a positive integer $e > 1$ such that $b = a^e \bmod n$. The *strong RSA assumption* says that no efficient (i.e., polynomial time) algorithm can solve the strong RSA problem with more than negligible probability.

The following assumption was first introduced by Paillier [Pai99].

Definition 6.5. Let n, p, q, p', q' be as in Definition 6.4. Let further $\mathcal{G}$ be the subgroup of $\mathbb{Z}_{n^2}^*$ consisting of all n^{th} powers of elements in $\mathbb{Z}_{n^2}^*$, i.e., $\mathcal{G} := \{a^n \ : \ a \in \mathbb{Z}_{n^2}^*\}$. The *decisional composite residuosity problem* is to, given n, distinguish random elements of $\mathbb{Z}_{n^2}^*$ from random elements of $\mathcal{G}$. The *decisional composite residuosity assumption* says that no efficient (i.e., polynomial time) algorithm can solve the decisional composite residuosity problem with more than negligible probability.

6.2.2 Zero-Knowledge Proofs of Knowledge

Zero-knowledge proofs of knowledge are a fundamental primitive for privacy-enhancing cryptography. They are two-party protocols between a prover and a verifier, where the prover claims to know some secret piece of information and needs to convince the verifier about this fact in a private manner.

Zero-knowledge proofs of knowledge have to satisfy the following security properties. First, *correctness* says that honest provers can always convince honest verifiers. Furthermore, they have to satisfy the following seemingly contradictory goals: On the one hand, *soundness* guarantees that a prover that can convince the verifier really knows the claimed secret, except for a negligibly small probability. On the other hand, the *zero-knowledge* property says that the proof does not reveal any information about the secret to the verifier, except for what is already revealed by the claim itself.

What is typically being proved in our applications is knowledge of discrete logarithms or similar statements. Now and in the following we use the notation introduced by Camenisch and Stadler [CS97] to denote such proof in an abstract way. For instance, an expression like:

$$ZKP\left[(\alpha, \beta, \gamma) \ : \ y_1 = g_1^\alpha g_2^\beta \ \wedge \ y_2 = y_1^\alpha g_3^\gamma \ \wedge \ \alpha > 0\right]$$

denotes a zero-knowledge proof of knowledge of integers α, β, γ such that the relations on the right hand side are satisfied. We stick to the convention that knowledge of values denoted by Greek letters has to be proved, while all other values are assumed to be publicly known.

We next show how such proof goals are compiled to real-world protocols on hand of the following simple example proof goal:

$$ZKP\left[(\alpha, \beta) \ : \ y = g^{\alpha} h^{\beta}\right] , \tag{1}$$

where g and h are generators of a group $\mathcal{G}$ of prime order q, and y is the public image. Let further be H a collision resistant hash function such as, e.g., SHA-2, and $\mathsf{desc}_{\mathcal{G}}$ be a description of the group $\mathcal{G}$. Then Figure 16 illustrates the protocol run.

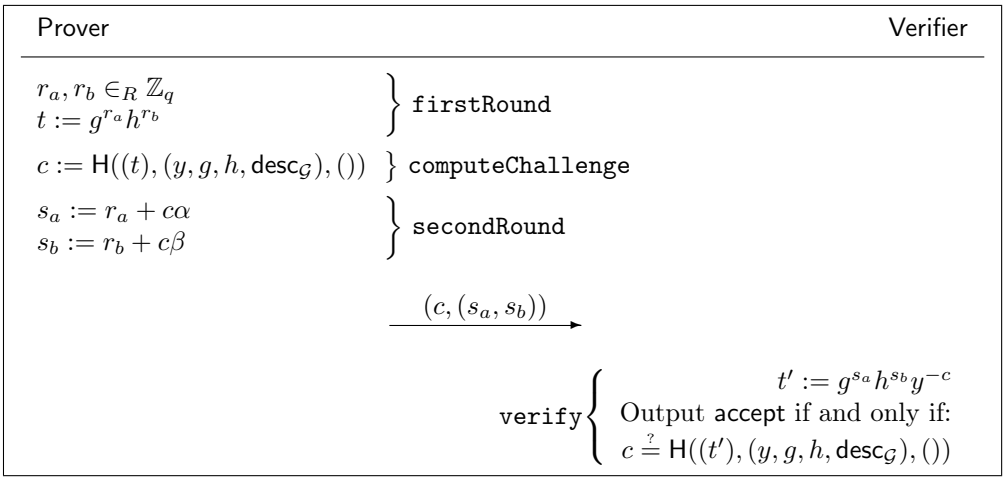

Figure 16: Protocol flow of the zero-knowledge proof of knowledge specified in (1). The given method names correspond to those discussed in Section 6.1.6.

On hand of this example, we next explain the concepts of T-values, etc. that were introduced in Table 6:

R-values. These are the internal random coins the prover draws. In Figure 16, the R-values are given by r_a and r_b. R-values are never revealed to the verifier in the clear.

T-values. Using the R-values the T-values are computed, essentially by evaluating the proof goal on the randomnesses instead of the secrets. T-values do not need to be sent to the verifier, but the verifier can re-compute them himself. They are hashed in order to compute the challenge c for the proof. In our example, the only T-value is t.

S-values. S-values are derived from the R-values, the challenge and the secrets. They are always computed as a sum of an R-value and the product of the challenge and the respective secret value, cf. Figure 16. S-values are sent to the verifier together with the challenge c. In our example, the S-values are given by s_a and s_b.

N-values. N-values contain all public values that are required to perform the proof, and that are already known to both parties before the proof starts. They typically specify the entire algebraic setting such as the groups and group elements being used, as well as the public images for which the prover claims to know the corresponding secrets. In Figure 16, the algebraic setting is given by $g, h, \mathsf{desc}_{\mathcal{G}}$ and the public image is given by y.

D-values. Finally, D-values are public values that are required to perform the proof, but that are not already known to both parties before the proof starts. Such values often arise when algebraic claims about secrets are to be proved, such as, e.g., $\alpha > 0$ or $\alpha = \beta\gamma$ or the like. Technically, such proof goals are realized by first reformulating such claims as claims related to discrete logarithms, i.e., to claims of the form $z = v^{\mu} w^{\nu}$, etc.. In this case often temporary public values need to be computed,

which are then added as D-values and which are also used when computing the challenge c. In our example no D-values are required, and therefore the empty list is hashed in Figure 16.

For a deeper discussion of the design of efficient zero-knowledge proofs of knowledge for practically relevant proof goals we refer the interested reader to the original literature, in particular Schnorr [Sch91] and generalizations [FO97, DF02, CKY09], and Fiat and Shamir [FS87].

Four Square Range Proof Many practically relevant proof goals require a user to prove that same secret value is larger (or smaller) than some threshold value. For instance, when claiming some senior citizen discount, a user has to prove that his secret birth date was before some public reference date. In this case, the proof goal contains an algebraic claim of the form $\alpha < \mathsf{date}$, where α is the secret attribute specifying the birth date of the user. As stated before, such claims need to be rewritten to discrete logarithm based claims before they can be proved efficiently.

By Lagrange's Four Square Theorem, every non-negative integer can be written as the sum of four squares, and this is obviously not the case for negative integers. Furthermore, efficient algorithms for computing this decomposition are known in the literature [RS86]. Therefore, a proof goal of the form:

$$ZKP\left[(\alpha,\rho)\ :\ y = g^{\alpha}h^{\rho}\ \wedge\ \alpha > \mathsf{date}\right]$$

can be rewritten to:

$$ZKP\left[(\chi_1,\chi_2,\chi_3,\chi_4,\rho)\ :\ y = g^{\chi_1^2+\chi_2^2+\chi_3^2+\chi_4^2+\mathsf{date}}h^{\rho}\right],$$

where $\chi_1^2+\chi_2^2+\chi_3^2+\chi_4^2 = \alpha - \mathsf{date}$.

Now, standard techniques found in the literature allow this to be rewritten to a conjunction of the form $z = v^{\mu}w^{\nu}$, which can then be proved as discussed before. The complexity of such a proof is roughly nine times the complexity of a proof for a statement of the form $z = v^{\mu}w^{\nu}$.

We refer the interested reader to the original literature [Lip03] for details.

6.2.3 Commitment Schemes

Informally, a commitment scheme can be seen as the digital equivalent of a sealed envelope: A party can commit to a chosen value, while keeping it secret from others. The committing party can later reveal (or *open*) the commitment to another party, which can verify the correctness of this opening.

There are three security requirements a commitment scheme has to satisfy. First, if an honest party commits to a message and later opens the commitment to another party, the latter will always be convinced that the opening is correct. This property is referred to as *correctness*. Second, the *hiding* property guarantees that only given the commitment, one cannot learn any information about the contained message. Third, it is infeasible to open a commitment to two different messages, i.e., to convince the receiver that two different openings are correct for the same commitment. This property is known as *binding*.

In our implementation, commitments are used in multiple places. They are used for advanced issuance to make an issuance protocol depend on a preceding credential presentation proof. At presentation, a user shows that he knows a credential, and additionally proves that the same attributes are contained in a freshly computed commitment. Then, for issuance, the value contained in this commitment is injected into the new credential. The hiding property guarantees that the issuer does not learn the attribute value, while the binding property guarantees the issuer that he issues a credential on an attribute that was already contained in a previous credential. Furthermore, we use commitments to realize protocol extensions such as inequality proofs, i.e., to prove that an attribute is larger than some (potentially public) other value. For this to be possible, we need commitment schemes that allow one to commit to arbitrarily large integer values, which is the case for the scheme presented in the following.

Pedersen/Damgård-Fujisaki Commitments Our implementation uses the so-called Damgård-Fujisaki-Okamoto scheme [DF02], which is a generalization of the Pedersen commitment scheme [Ped91] to messages of arbitrary length.

Key generation. A commitment key is computed by drawing a random safe RSA modulus n, S randomly in $\mathbb{Z}_n^*$ and $R_1, \ldots, R_L$ randomly in $\langle S \rangle$.

Message space. The commitment scheme supports arbitrary messages in $\mathbb{Z}^L$.

Committing to a message. Given a message $m = (m_1, \ldots, m_L)$, the commitment is computed as follows:

1. Choose a random $r \in_R [0, \lfloor n/4 \rfloor]$ and
2. compute the commitment as $com := \prod_{i=1}^{L} R_i^{m_i} S^r \bmod n$.

Verifying a commitment. Given a commitment com, a message m and an opening r, the validity can be checked by checking whether the following equation is satisfied:

$$com \stackrel{?}{=} \prod_{i=1}^{L} R_i^{m_i} S^r \bmod n\,.$$

We note that it is important that the committing entity is not privy of the factorization of n. For our instantiation, we can use $R_1, \ldots, R_L$, S and n from the public key of an issuer.

Theorem 6.1 ([DF02])**.** *Under the strong RSA assumption, the above commitment scheme is correct, statistically hiding and computationally binding.*

6.2.4 Blind Signature Schemes

A blind signature scheme allows a user to obtain signatures on messages (or attributes) from a signer, without the signer learning the attributes he signed. As a non-digital example, one could think of the following scenario. A voter privately makes his choice in a voting booth, and then puts his ballot into a carbon paper envelope. He then authenticates himself towards the voting authorities, proving that he is indeed eligible to vote, e.g., by showing his passport. The authorities approve this by signing the envelope, and therefore also the ballot. The signed envelope is then put into the ballot box. Now, when counting the votes, it can be verified whether or not a ballot was voted by an eligible voter, but the authorities never learned the choice of any specific citizen.

Informally, a blind signature scheme should satisfy the following security properties. First, an honest user should always be able to obtain a signature on messages of his choice. This property is referred to as *correctness*. Second, *blind issuance* ensures that the signer does not learn any information about the messages he signed. Third, *untraceability* guarantees that when proving possession of a blindly obtained signature, this cannot be linked to a specific issuance session. Finally, the *unforgeability under chosen message attacks* property says that only the signer is able to produce valid signatures, i.e., no other party is able to produce a signature on a message that has not been signed by the signer before, even if it can request signatures on arbitrary message of its choice.

Blind signatures are at the heart of all known anonymous credential systems. The scheme underlying IBM's idemix are so-called CL-signatures [CL02a], whereas the scheme underlying Microsoft's U-Prove is Brand's blind signature scheme [Bra93].

CL-Signatures CL-signatures were first proposed by Camenisch and Lysyanskaya [CL02a]. They are at the heart of IBM's identity mixer (aka *idemix*), and many other real world applications such as Direct Anonymous Attestation that allows one to remotely authenticate a machine while preserving privacy.

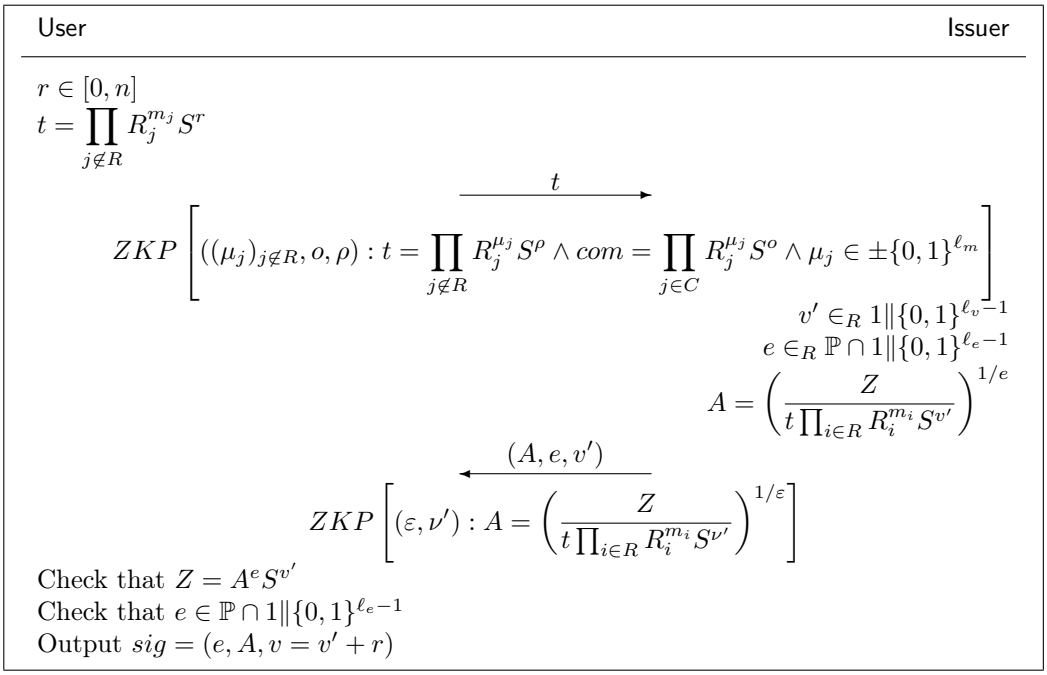

Figure 17: Issuance of a signature for attributes $(m_1, \ldots, m_L)$. In the first zero-knowledge proof, the user acts as the prover, while in the second proof the issuer acts as the verifier. If any of the checks fails or proofs fails, the protocol aborts.

Key generation. On input ℓ_n, choose an ℓ_n-bit RSA modulus n such that $n := pq$, $p := 2p' + 1$, $q := 2q' + 1$, where p, q, p', and q' are primes. Choose, uniformly at random, $R_1, \ldots, R_L, S, Z \in \mathbb{QR}_n$, where $\mathbb{QR}_n$ is the set of quadratic residues modulo n.

Output the public key $(n, R_1, \ldots, R_L, S, Z)$ and the secret key p.

Message space. Let ℓ_m be a parameter. The message space is the set

$$\{(m_1, \ldots, m_L) : m_i \in \pm\{0,1\}^{\ell_m}\} \ .$$

Signing. Let ℓ_r, $\ell_e > \ell_m + 2$ and $\ell_v := \ell_n + \ell_m + \ell_r$ be security parameters. A signature on messages $m_1, \ldots, m_L$ is then generated by the protocol depicted in Figure 17. There, $R \subseteq \{0, \ldots, n-1\}$ denotes the set of indices of the messages which are revealed to the signer, and $C \subseteq \{0, \ldots, n-1\} \setminus R$ denotes the set of indices of the messages which are to be carried over from a previous signature. That is, as discussed earlier, the commitment *com* is the output of a preceding presentation proof, and the values of the contained messages are carried over into the new signature, ensuring security to both parties.

Signature presentation. When proving possession of a signature, the user can again decide to reveal a subset R of the messages, and additionally generate a commitment *com* for a subset C of different messages. The latter can then be used in a follow-up issuance session to show that the attributes where actually blindly carried-over correctly as describe above.

Presentation is now done as follows:

1. The user first re-randomizes the signature (e, A, v) by choosing a random $r \in_R [0, n]$ and computing $(e, A' = AS^{-r}, v' = v + er)$. Note that A' is statistically close to the uniform over $\mathbb{Z}_n^*$ and therefore does not leak any information to the verifier.
2. The user computes a commitment *com* to the attributes with indices in C as described in Section 6.2.3.

3. The user then sends A' and *com* to the adversary.
4. The user and the verifier run the following zero-knowledge proof of knowledge:

$$ZKP\Big[((\mu_j)_{j\notin R}, \rho, \nu', \varepsilon) : Z\prod_{i\in R} R_i^{-m_i} = A'^{\varepsilon}\prod_{j\notin R} R_j^{\mu_j} S^{\nu'} \ \wedge \ com = \prod_{j\in C} R_j^{\mu_j} S^{\rho} \wedge \bigwedge_{j\notin R} \mu_j \in \pm\{0,1\}^{\ell_m} \ \wedge \ 2^{\ell_e-1} < \varepsilon < 2^{\ell_e}\Big] .$$

5. The verifier accepts if and only if this proof output accept.

Theorem 6.2 ([CL02a])**.** *Under the strong RSA assumption, the above scheme is secure against adaptive chosen message attacks. Furthermore, for any polynomially bounded number of presentations, it is computationally infeasible to link presentation sessions among each other, or presentation sessions to an issuance session, even if verifiers and issuers collude.*

The original scheme considered messages in the interval $[0, 2^{\ell_m} - 1]$. Here, however, we allow messages to be from $[-2^{\ell_m} + 1, 2^{\ell_m} - 1]$. The only consequence of this is that we need to require that $\ell_e > \ell_m + 2$ holds instead of $\ell_e > \ell_m + 1$.

Brands Signatures The signature scheme presented in the following was introduced by Brands [Bra93]. It is the core building block of Microsoft's U-Prove anonymous credential system.

In the following, let H be a collision resistant hash function, i.e., it is hard to find to different values which map to the same output.

Key generation. Choose random primes p and q such that $q|(p-1)$ and computing discrete logarithms in the subgroup of order q of $\mathbb{Z}_p^*$ is hard for the given security parameter. Choose further a random generator g of this subgroup, and random values $y_i \in_R \mathbb{Z}_q$ for $i = 0, \ldots, L$, and define $g_i := g^{y_i}$ for all i.

Output the public key $(g, p, q, g_0, \ldots, g_L)$ and the secret key y_0.

Message space. Let ℓ_m be such that $2^{\ell_m} < q$. The message space is the set

$$\{(m_1, \ldots, m_L) : m_i \in \{0,1\}^{\ell_m}\} \ .$$

Signing. Using the same notation as for the signing algorithm in Section 6.2.4, Figure 18 shows how messages can blindly be signed.

Signature presentation. Again using the notation from Section 6.2.4, knowledge of a signature is done as follows:

1. The user computes a commitment *com* to the attributes with indices in C as described in Section 6.2.3.
2. The user sends (h, z', c', r') and *com* to the verifier.
3. The user and the verifier run the following zero-knowledge proof of knowledge:

$$ZKP\Big[((\mu_j)_{j\notin R}, \sigma, \rho) : h = \Big(g_0\prod_{j\in R} g_i^{m_i}\prod_{j\notin R} g_i^{\mu_j}\Big)^{\sigma} \ \wedge \ com = \prod_{j\in C} R_j^{\mu_j} S^{\rho}\Big].$$

4. The verifier accepts if and only if $c' \stackrel{?}{=} \mathsf{H}(h, z', g^{r'} g_0^{-c'}, h^{r'}(z')^{-c'})$ and the above proof output accept.

Theorem 6.3. *For any polynomially bounded number of presentations, it is infeasible to link presentation sessions to an issuance session, even if verifiers and issuers collude.*

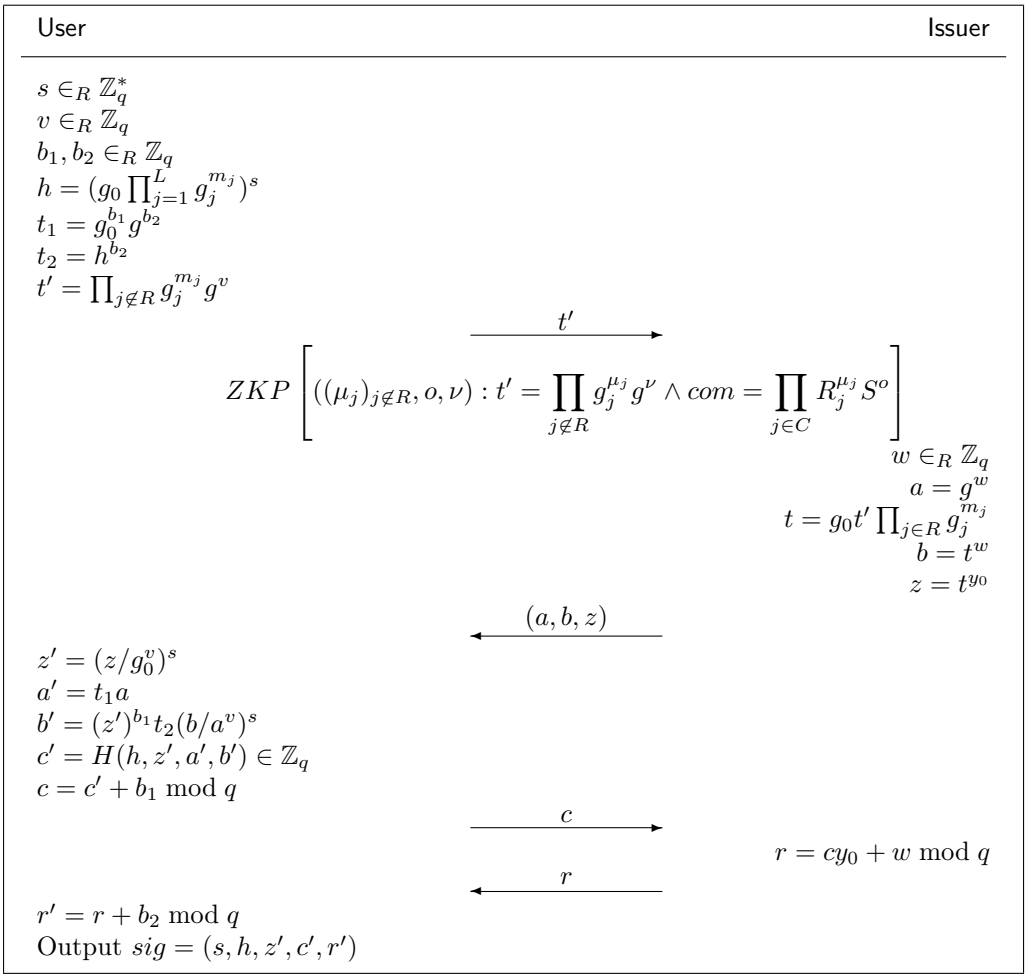

Figure 18: Issuance of a signature for attributes $(m_1, \ldots, m_L)$. In the zero-knowledge proof the user acts as the prover. If any of the checks or proofs fails, the protocol aborts.

Note here that in contrast to CL-signatures it is possible to link multiple presentations of the same signature among each other. This can easily be seen by the way presentations are done: there, the user reveals parts of the signature to the verifier. Therefore, if presentations should be unlinkable, every signature must only be used in a single presentation session.

Furthermore, note that there does not exist a formal proof that Brands signatures are unforgeable under chosen message attacks. However, they have been well studied for almost two decades and are widely believed to be secure.

6.2.5 Verifiable Encryption

Verifiable encryption is the cryptographic building block that makes inspection possible by allowing a user to prove that the encrypted plaintext, the inspectable attribute, is identical to a committed or signed value.

In public key encryption schemes, each user has two keys: a public key, which others can use to encrypt message to this user, and a secret key, which the user can use to decrypt these ciphertexts. Such schemes can be thought of as digital equivalents of standard letter boxes: Each other user can post a letter in this box, but only the legitimate owner of the letter box is able to open it and extract the letter from the box.

Informally, the public key encryption schemes used for anonymous credentials have to satisfy the following security properties. First, they have to be *correct*, i.e., decrypting a ciphertext always yields the message that was originally encrypted. Second, they should be *indistinguishable under chosen-ciphertext attacks*. This means, that given a ciphertext, no adversary knowing the public key but not the secret key can tell which message got encrypted. This has to hold even if the adversary knows that the ciphertext is an encrypted of one out of two adversarially chosen plaintexts, and if the adversary is allowed to obtain decryptions of arbitrary different ciphertexts.

Verifiable encryption schemes are an extension of public key encryption schemes, where the sender is additionally able to prove certain statements about the message he encrypted, without having to reveal the message. In particular, a sender is able to prove that he knows the message contained in a ciphertext, or, e.g., that the ciphertext contains the same message as a given commitment.

The Camenisch-Shoup Encryption Scheme The scheme described here was proposed by Camenisch and Shoup [CS03a] and is a variation of an encryption scheme put forth by Cramer and Shoup [CS02]. The scheme makes use of a keyed hash scheme $\mathcal{H}$ that uses a key hk, chosen at random from an appropriate key space associated with the security parameter. Every hash function $\mathsf{H} \in \mathcal{H}$ maps triples of the form (u, e, L) to integers in the set $[0, 2^\ell - 1]$. The hash functions have to be collision resistant, i.e., given a random hash key hk it is infeasible to find two different triples mapping to the same value.

We further define $\mathsf{abs} : \mathbb{Z}^*_{n^2} \to \mathbb{Z}^*_{n^2}$ as the function mapping $(a \bmod n^2)$ to $(n^2 - a \bmod n^2)$ if $a > n^2/2$, and to $(a \bmod n^2)$, otherwise. Note that $v^2 \equiv (\mathsf{abs}(v))^2$ holds for all $v \in \mathbb{Z}^*_{n^2}$.

Key generation. On input ℓ_n, choose an ℓ_n-bit RSA modulus n such that $n := pq$, $p := 2p' + 1$, $q := 2q' + 1$, where p, q, p', and q' are primes. Let further $n' := p'q'$. Choose random $x_1, x_2, x_3 \in_R [0, \lfloor n^2/4 \rfloor]$, and a random $g' \in_R \mathbb{Z}^*_{n^2}$, and compute $g := (g')^{2n}$, and $y_i := g^{x_i}$, for $i = 1, 2, 3$.

Also, generate a hash key hk from the key space of the hash scheme $\mathcal{H}$ associated with the given security parameter.

The public key is $(\mathsf{hk}, n, g, y_1, y_2, y_3)$. The secret key is $(\mathsf{hk}, n, x_1, x_2, x_3)$.

Message space. The message space is given by $[0, n]$.

Encryption. To encrypt a message m with label $L \in \{0,1\}^*$ under a public key as above, choose a random $r \in_R [0, \lfloor n/4 \rfloor]$ and compute

$$u := g^r \ , \qquad e := y_1^r (n+1)^m \ , \quad \text{and} \qquad v := \mathsf{abs}\left((y_2 y_3^{\mathsf{H}(u,e,L)})^r \right) \ .$$

The ciphertext is (u, e, v).

Decryption. To decrypt a ciphertext $(u, e, v) \in \mathbb{Z}^*_{n^2} \times \mathbb{Z}^*_{n^2} \times \mathbb{Z}^*_{n^2}$ with label L under a secret key as above, first check that $\mathsf{abs}(v) \equiv v$ and $u^{2(x_2 + \mathsf{H}(u,e,L)x_3)} \equiv v^2$. If this does not hold, then output reject and halt. Next, let $t := 2^{-1} \bmod n$, and compute $\hat{m} := (e/u^{x_1})^{2t}$. If $\hat{m}$ is of the form h^m for some $m \in [0, n]$, then output m; otherwise, output reject. This can efficiently be tested using the fact that $h^m \equiv 1 + mn \bmod n^2$ for $0 \le m < n$, and therefore in this case $m = \frac{\hat{m}-1}{n}$.

Theorem 6.4. *Under the decisional composite residuosity assumption and if the deployed hash function is collision resistant, the scheme described above is indistinguishable under chosen-ciphertext attacks.*

In a credential scheme, typically some attribute uniquely identifying the credential and/or the owner of the credential is encrypted for a presentation proof, under the public key of some public authority such as, e.g., a judge, commonly referred to as *inspector*. When presenting the credential, the user additionally shows that the computed ciphertext is indeed the same as the corresponding attribute value in the credential. While the user's privacy is maintained, the verifier is thereby ensured that he has a valid encryption of the user's identity. Upon misbehavior of the user, the verifier can now request the public authority to reveal the identity of the user by decrypting the ciphertext, and thus the user can, e.g., be held accountable for any damage he caused.

6.2.6 Scope-Exclusive Pseudonyms

Pseudonyms are aliases that users assume for particular applications or settings. That is, a user may be known under different pseudonyms to different entities, such that those entities cannot decide whether two pseudonyms belonged to the same user or not. A pseudonym is scope-exclusive, if for a given string specifying the scope of the session, e.g., the URL of a webpage or the name of a service, the user can only generate a unique pseudonym. This means that within a given scope users can be recognized, but that they are still unlinkable across scopes. If for a certain service it is not required to be recognizable, the scope can just be set to a fresh random string every time, thereby becoming fully unlinkable.

Technically, a user is identified with a secret key that is only known to that specific user. A scope-exclusive pseudonym is then derived deterministically from the scope string and the user's secret key. Whenever a user gives a pseudonym to a verifier, he further proves that he knows the secret key that was used to derive the pseudonym without revealing it.

Such a scheme has to satisfy the following security properties. The scheme must be *complete*, meaning that an honest user deriving a pseudonym from his secret key can convince the verifier that he is indeed privy of this secret key, i.e., that he owns the identity hidden behind the given pseudonym. On the other hand *soundness* guarantees that only an honest user can convince a verifier about this fact. The *scope-exclusiveness* property says that for each scope string, every user secret key maps to a unique pseudonym. *Collision resistance* ensures that for every fixed scope, no two different identities map to the same pseudonym. Finally, *unlinkability* says that given pseudonyms to two different scopes, it is infeasible to decide whether or not they were derived from the same user secret key.

Efficient Scope Exclusive Pseudonyms In the following we present the algorithms for an efficient pseudonym system.

Key generation. The public key of the scheme consists of a group $\mathcal{G}$ of prime order q, and a hash key hk specifying a collision resistant hash function H as in Section 6.2.5.

User key generation. A user's secret key is computed by randomly choosing an $x \in_R \mathbb{Z}_q$.

Pseudonym generation. Given a scope string scope and a user secret key x, the pseudonym is given by $\mathsf{nym} := \mathsf{H}(\mathsf{scope})^x$.

Pseudonym presentation. To convince a verifier that the user knows the identity behind a given pseudonym, they perform the following zero-knowledge proof of knowledge:

$$ZKP\left[(\chi) : \mathsf{nym} = \mathsf{H}(\mathsf{scope})^{\chi}\right] .$$

Theorem 6.5. *Under the DDH-assumption for $\mathcal{G}$ and if the deployed hash function is collision resistant, the given scope-exclusive pseudonym system is secure.*

In practice, the user's secret key is embedded as an attribute into a credential. Then, when presenting a credential under a pseudonym, the user shows that the same user secret key was used in the presentation of the credential and to derive the pseudonym.

6.2.7 Revocation

In real-world applications of anonymous credentials it is vital to have efficient means to revoke credentials. On the one hand, users might want to revoke their credentials, e.g., if they loose the device they used to store the credential, or if it gets stolen. On the other hand, service providers might want to revoke credentials upon misbehavior such as credential sharing.

In a revocation scheme, a revocation authority gives secret pieces of information to every user, and publishes some publicly available revocation information. Now, whenever presenting a credential, the user simultaneously proves that his credential has not yet been revoked by showing that he possesses such an unrevoked secret piece of information, and that this data is somehow linked to the presented credential.

Informally, revocation schemes have to satisfy the following security properties. First, *correctness* ensures that honest holders of unrevoked credentials can always convince the verifier that this is indeed the case. Second, by the *soundness* property, verifiers are ensured that holders of revoked credentials cannot make them accept. Finally, to protect the user's privacy, the *zero-knowledge* property guarantees the user that no personal information is leaked to the verifier when proving that the credential has not yet been revoked.

Camenisch-Lysyanskaya Accumulators The scheme described in the following was presented by Camenisch and Lysysanskaya [CL02b]. On a very high level, the scheme works as follows: The revocation authority publishes some revocation information v in $\mathbb{Z}_n^*$, and hands a secret pair (e, u) to the user, where $e > 1$ is an integer, and u is an e^{th} root of v, such that no two users receive the same e. If a user wants to prove that his secret has not yet been revoked, he proves that he knows such a pair (e, u) in a zero-knowledge manner. If a user's secret key is to be revoked, the revocation authority just computes a root of v, obtaining $v^{e^{-1}}$ as the new revocation information. Unrevoked users can update their pairs efficiently, while the revoked user would now have to compute a fresh root of the new revocation information, as his secret exponent was "divided out". However, the latter is impossible under the strong RSA assumption, cf. Definition 6.4.

Key generation. The revocation authority, on input ℓ_n, chooses an ℓ_n-bit RSA modulus n such that $n := pq$, $p := 2p' + 1$, $q := 2q' + 1$, where p, q, p', and q' are primes. It further chooses $v, g, h \in_R \mathbb{QR}_n$.

The public key is given by (u, g, h, n) and the secret key is given by (p, q).

Join. Whenever a new item is added to the accumulator, i.e., whenever a new credential is issued, the revocation authority hands over a random prime e and $u \in \mathbb{QR}_n$ such that $u^e \equiv v \bmod n$. Using the secret key, such a u can always be computed efficiently as $u = v^{e^{-1} \bmod (p-1)(q-1)}$ using the extended Euclidean algorithm.

Revoking a user. If a user's certificate is to be revoked, the revocation authority updates the public revocation information v as $v := v^{e'^{-1} \bmod (p-1)(q-1)} \bmod n$, where e' denotes the exponent the user received when he joined the group. The revocation authority then further published the value e'.

Updating the revocation information. Every time a user's certificate gets revoked, all the remaining users have to update their secret values u and e. Let therefore e' denote the revoked value, $v_{\texttt{new}}$ be the new revocation information published by the revocation authority, and $v_{\texttt{old}}$ be the public revocation information from before e' was revoked.

In a first step, a user uses the extended Euclidean algorithm to compute integers a and b such that $ae + be' = 1$. In a second step, the user then updates his private group element u to $u := u^b v_{\texttt{new}}^a$.

Proving unrevokedness. A user can prove to a verifier that his certificate has not been revoked by performing the following steps:

1. The user first draws $r_1, r_2, r_3 \in_R [0, \lfloor n/4 \rfloor]$.
2. The user then computes $C_e = g^e h^{r_1}$, $C_u = u h^{r_2}$ and $C_r = g^{r_2} h^{r_3}$, which he sends to the verifier.
3. The user and the verifier together run the following zero-knowledge proof of knowledge:

$$ZKP\Big[(\varepsilon, \rho_1, \rho_2, \rho_3, \phi, \psi) \ : C_e = g^{\varepsilon} h^{\rho_1} \ \wedge \ C_r = g^{\rho_2} h^{\rho_3} \ \wedge \\ v = C_u^{\varepsilon} h^{-\phi} \ \wedge \ 1 = C_r^{\varepsilon} g^{-\phi} h^{-\psi}\Big].$$

   Here, the user uses $r_2 e$ for ϕ and $r_3 e$ for ψ.

4. The verifier accepts if and only if this proof output accept.

Theorem 6.6 ([CL02b])**.** *Under the strong RSA assumption, the scheme described above is a secure revocation scheme.*

The above revocation scheme is linked to the credential scheme by embedding the user's revocation key e as an attribute into a credential. The user then shows that he knows an unrevoked revocation key,

and that the very same key is contained in the credential, thereby proving that the credential has not been revoked.

7 A Case Study based on Privacy-ABCs

In this section, we illustrate the different artifacts in the language framework of Section 4 by means of an example scenario of an online library. For the sake of space and readability, the artifact examples for the scenario do not illustrate all the features of Privacy-ABCs. In the following, we will explicitly distinguish between user attributes (as contained in a credential) and XML attributes (as defined by XML schema) whenever they could be confused.

7.1 Example Scenario

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

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

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

7.2 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 below.

```
<CredentialSpecification KeyBinding="true" Revocable="true">
  <SpecificationUID> urn:creds:id </SpecificationUID>
  <AttributeDescriptions MaxLength="256">
    <AttributeDescription Type="urn:creds:id:name" DataType="xs:string" Encoding="xenc:sha256">
      <FriendlyAttributeName lang="EN"> Full Name </FriendlyAttributeName>
    </AttributeDescription>
    <AttributeDescription Type="urn:creds:id:state" DataType="xs:string" Encoding="xenc:sha256"/>
    <AttributeDescription Type="urn:creds:id:bdate" DataType="xs:date" Encoding="date:unix:signed"/>
    <AttributeDescription Type="urn:creds:id:cardnr" DataType="xs:integer" Encoding="integer:unsigned" />
    <AttributeDescription Type="urn:revocationhandle" DataType="xs:integer" Encoding="integer:unsigned" />
  </AttributeDescriptions>
</CredentialSpecification>
```

The XML attribute `KeyBinding` indicates that 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.

In our example, electronic identity cards contain a person’s full name, state, date of birth, and a unique card number. 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. Attributes that may have values longer than

`MaxLength` have to be hashed, as is done here for the name and state using SHA-256. The specification can also define human-readable names for the user attributes in different languages (Line 5).

7.3 Issuer, Revocation, and System Parameters

The government of Utopia acts as issuer and revocation authority for the identity cards. It uses some pre-generated system parameters

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

and generates an issuance key pair and publishes the following issuer parameters.

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

It also generates and publishes the following revocation authority parameters.

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

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.

When a credential is subject to issuer-driven revocation (as indicated in the credential specification), a presentation token related to this credential must always contain a proof that the presented credential has not been revoked. The issuer parameters above specify the revocation authority parameters that define where the most recent revocation information can be fetched to compute and verify the proof.

7.4 Presentation and Issuance Policies with Basic Features

Assume that a user already possesses an identity card from the Republic of Utopia issued according to the credential specification depicted above. 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 following presentation policy. Note that, even though the state is encoded in the credential in hashed form, it is the attribute itself that will be revealed in the presentation.

```
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>
```

The message to be signed is specified in the policy (Lines 2–4). In this case, it only includes a nonce to prevent replay attacks, i.e. to ensure freshness of the presentation token. Note that, 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. The `Pseudonym` element (Line 5) indicates that the presentation token must contain a scope-exclusive pseudonym, 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.

The `Credential` element (Lines 6–14) imposes that an ID card issued by the republic of Utopia must be presented (Lines 7-9 and 10–12) of which the state attribute must be revealed (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. The `SameKeyBindingAs` attribute of the `Credential` element (Line 6) indicates that the identity card must be bound to the same key as the pseudonym in Line 5.

Library cards are key-bound credentials that contain only a single attribute, namely the applicant's identity card number. Their credential specification is given below.

```
1  <CredentialSpecification KeyBinding="true" Revocable="true">
2    <SpecificationUID> urn:utopia:lib </SpecificationUID>
3    <AttributeDescriptions MaxLength="256">
4      <AttributeDescription Type="urn:utopia:lib:idcardnr" DataType="xs:integer" Encoding="integer:unsigned">
5        <FriendlyAttributeName lang="EN"> ID Card Number </FriendlyAttributeName>
6      </AttributeDescription>
7      <AttributeDescription Type="urn:revocationhandle" DataType="xs:integer" Encoding="integer:unsigned" />
8    </AttributeDescriptions>
9  </CredentialSpecification>
```

A library card contains the applicant's ID card number and must be 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 ID card number 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 an issuance policy that contains the presentation policy above 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:idcardnr">
8          <SourceCredentialInfo Alias="id" AttributeType="urn:creds:id:cardnr"/>
9        </CarriedOverAttribute>
10     </UnknownAttributes>
11   </CredentialTemplate>
12 </IssuancePolicy>
```

The credential template (Lines 3–11) first states 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 (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 (Line 8) indicates the credential and the attribute of which the value will be carried over.

7.5 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 between the issuance token below and to the presentation and issuance policy above is that in the returned token the `Pseudonym` now also contains a `PseudonymValue` (Line 6). Similarly, the `DisclosedAttribute` elements (Lines 10–12) in the 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 (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>
```

7.6 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 following presentation policy.

```
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:idcardnr" >
12         <InspectorAlternatives>
13           <InspectorParametersUID> urn:lib:arbitrator </InspectorParametersUID>
14         </InspectorAlternatives>
15         <InspectionGrounds> Late return or damage. </InspectionGrounds>
16       </DisclosedAttribute>
17     </Credential>
18     <Credential Alias="id" >
19       <CredentialSpecAlternatives>
20         <CredentialSpecUID> urn:creds:id </CredentialSpecUID>
21       </CredentialSpecAlternatives>
22       <IssuerAlternatives>
23         <IssuerParametersUID> urn:utopia:id:issuer </IssuerParametersUID>
24       </IssuerAlternatives>
25     </Credential>
26     <VerifierDrivenRevocation>
27       <RevocationParametersUID> urn:lib:blacklist </RevocationParametersUID>
28       <Attribute CredentialAlias ="libcard" AttributeType="urn:utopia:lib:idcardnr" />
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" >
36     <Message> ... </Message>
37     <Credential Alias="libcard" SameKeyBindingAs="id" >
38       <CredentialSpecAlternatives>
39         <CredentialSpecUID> urn:utopia:lib </CredentialSpecUID>
40       </CredentialSpecAlternatives>
41       <IssuerAlternatives>
42         <IssuerParametersUID> urn:utopia:lib:issuer </IssuerParametersUID>
43       </IssuerAlternatives>
44       <DisclosedAttribute AttributeType= "urn:utopia:lib:idcardnr" >
45         <InspectorAlternatives>
46           <InspectorParametersUID> urn:lib:arbitrator </InspectorParametersUID>
47         </InspectorAlternatives>
48         <InspectionGrounds> Late return or damage. </InspectionGrounds>
49       </DisclosedAttribute>
50     </Credential>
51     <VerifierDrivenRevocation>
52       <RevocationParametersUID> urn:lib:blacklist </RevocationParametersUID>
53       <Attribute CredentialAlias ="libcard" AttributeType="urn:utopia:lib:idcardnr" />
54     </VerifierDrivenRevocation>
55   </PresentationPolicy>
56 </PresentationPolicyAlternatives>
```

The above presentation policy contains two `Credential` elements (Lines 4–17, 18–25), one for the library card and one for the identity card. The XML attribute `SameKeyBindingAs` (Line 4) of the library credential referring to the identity card imposes that both credentials are underlain by the *same* secret key. This prevents more senior readers from combining their library card with the identity card of a young reader. No user attributes of the identity card have to be revealed, but the `AttributePredicate` element (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.

To be able to nevertheless reveal the ID card number of an anonymous borrower and to impose a fine when a book is returned late or damaged, the library makes use of inspection. The `DisclosedAttribute` element (Lines 11–16) for the XML attribute `"urn:utopia:lib:idcardnr"` contains `InspectorParametersUID` and `InspectionGrounds` child elements, indicating that the ID card number should not be revealed in the clear to the verifier, but in a verifiably encrypted form to an inspector. 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 user attribute value may be revealed by the arbiter.

The library cards of customers who too often return borrowed books late or damaged must be excluded from borrowing further paper books, but must still be usable for other purposes. In the preceding example, this is taken care of by the `VerifierDrivenRevocation` element (Lines 26–29) that specifies that the library card must be checked against the most recent revocation information from the revocation authority `urn:lib:blacklist`. Revocation is performed based on the customer's ID card number, as indicated by the `Attribute` child element (Line 28). Revocation can also be based on a combination of user attributes from a set of different credentials, in which case there will be multiple `Attribute` child elements. The user then has to prove that a disjunctive combination of attribute values from each `VerifierDrivenRevocation` element is not revoked with respect to `RevocationParametersUID`.

7.7 Interaction with the User Interface

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

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

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.

The `UiPresentationArguments` object is designed to minimize the complexity of the user interface: (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 consists of two parts, as illustrated above: 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 the issuer-driven revocation authorities (Line 8), and credential descriptions (Lines 9–10), and pseudonym descriptions (shown in Line 4 of the issuance example below). 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 of the issuance example below); 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 Privacy-ABC framework will tentatively create new pseudonyms each time and include those in the list; these pseudonyms are then only saved if the user actually selects them for inclusion in the presentation token.

The `UiPresentationReturn` object 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. The issuance example below illustrates that 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.

```
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>
```

```
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>
```

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

8 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?*".

8.1 Definition 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.

8.2 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.

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].

8.3 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 and learn about the necessary trust relationships that need to be established. Figure 19 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 19, the general trust requirements by all the parties are demonstrated.

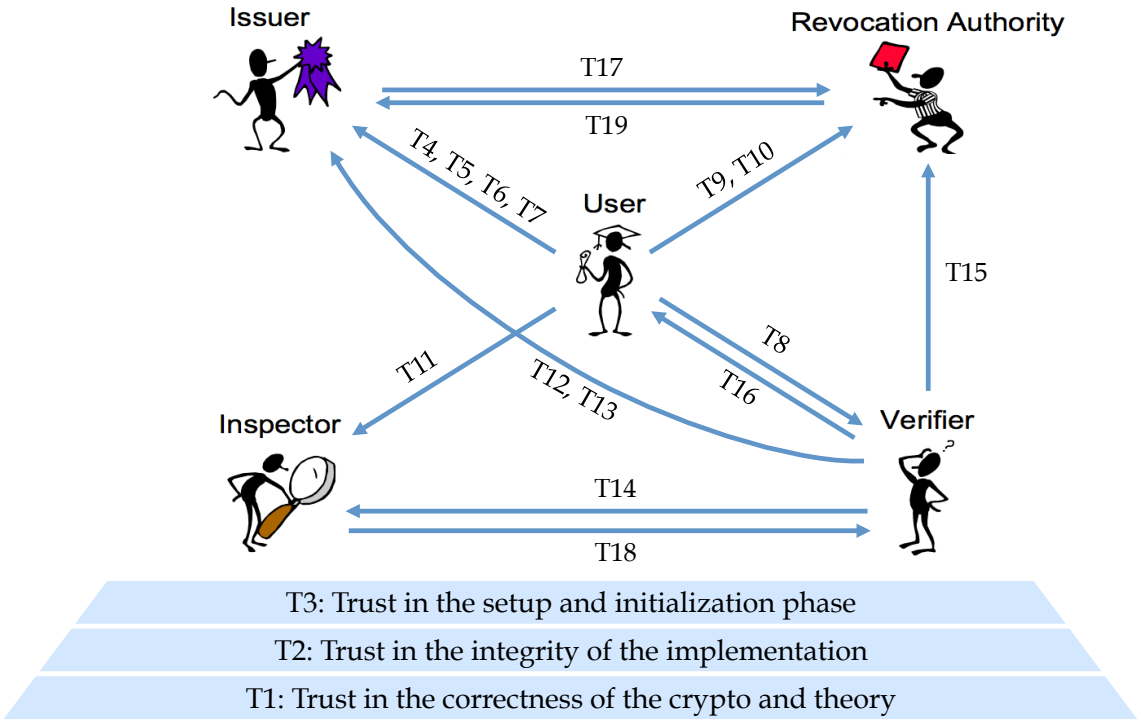

Figure 19: Visualization of the trust relationships

8.3.1 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 employ Privay-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.

8.3.2 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. 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.*

8.3.3 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 actives 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.*

8.3.4 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. The verifiers rely on the information that is certified by the issuers for the authentication phase so 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 limit the usage of the credentials to the 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.*

8.3.5 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.*

8.3.6 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.*

8.3.7 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.*

9 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.[5] 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.

9.1 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.

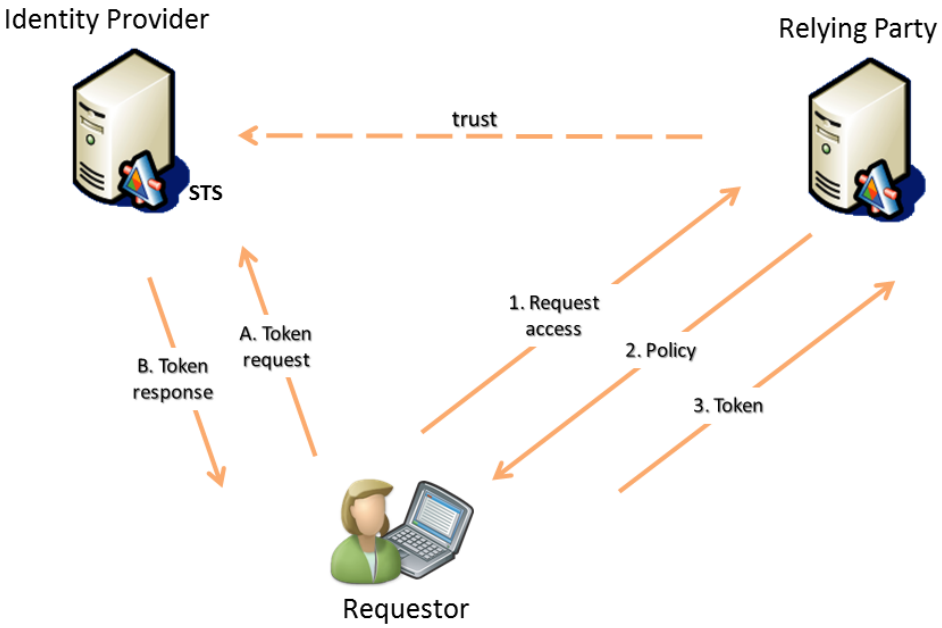

Figure 20: WS-Trust protocol flow

The WS-Trust specification is the main building block that defines how *security tokens* can be obtained

[5] 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.

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 Provider's 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 20.

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 [WST09]). 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 21.

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 5.

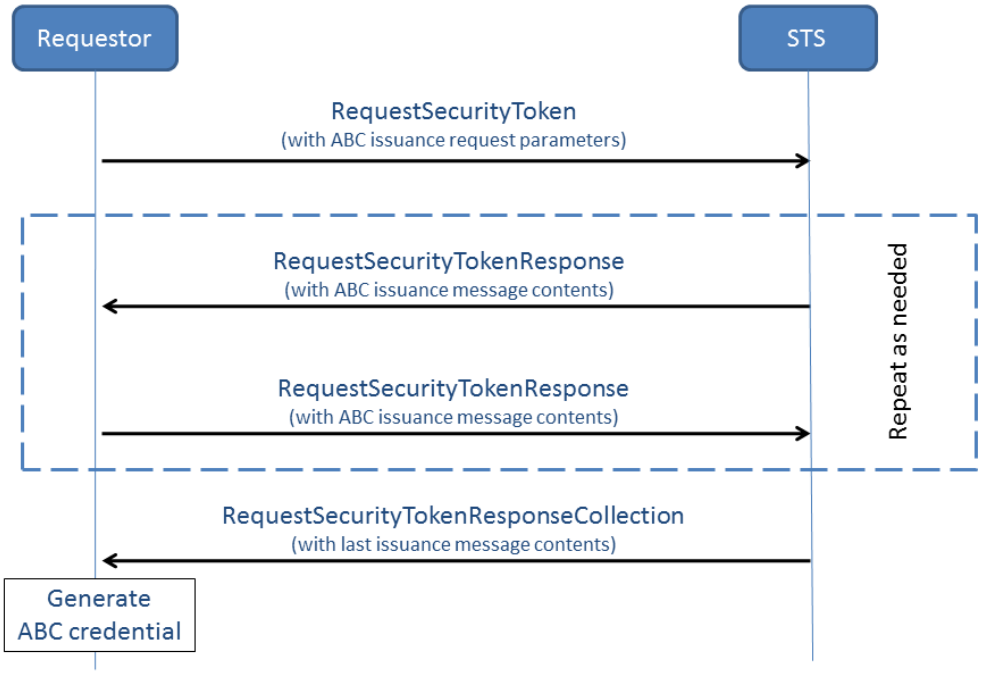

Figure 21: WS-Trust issuance protocol

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 architecture using WS-Federation, the presentation token could be included in a `RequestForSecurity TokenResponse` 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 service's federation metadata (see Section 3 of [WSF09]). Privacy-ABC technology integration into

WS-Trust has been successfully demonstrated; see, e.g., [UPW11].

9.2 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 22 illustrates the protocol flow.

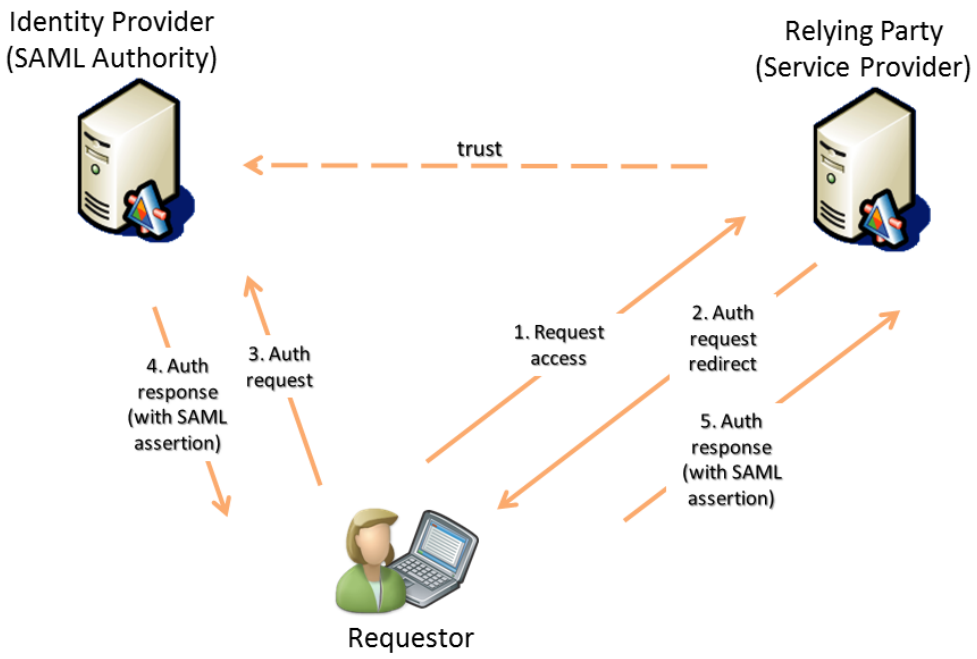

Figure 22: SAML protocol flow

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 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 [BBF+02]. This means that XML Signature must also be profiled to support ABC issuer signatures.[6] The alternative would be to protect the SAML assertion using a custom external signature element.

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

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 Party's authentication request rather than fetching it in real-time from the Identity Provider (replacing steps 3 and 4 in Figure 22). The assertion would contain the disclosed attributes, and encode the presentation token's 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.

9.3 OpenID

OpenID is a federated protocol allowing users to present an identifier[7] 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 23:

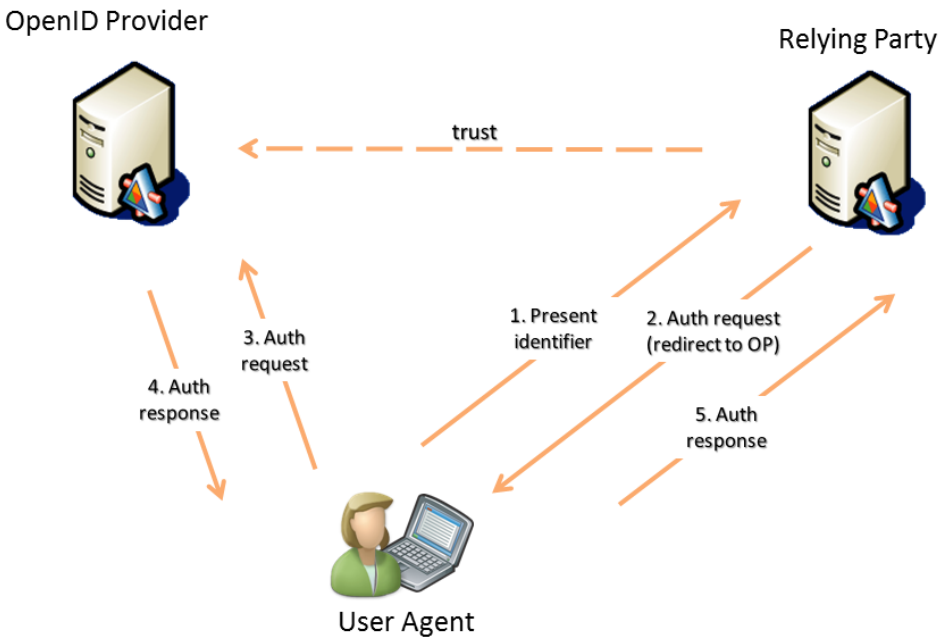

Figure 23: OpenID protocol flow

We assume that the user has an existing OpenID identifier registered with an OpenID Provider.

1. To login to a Relying Party, the user presents her (unverified) OpenID identifier.
2. The Relying Party parses the identifier to discover the User's OpenID Provider and redirects the User Agent to it.

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

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 describes 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 User's spheres of activities at different Relying Parties (using the Relying Partie's 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 Chaum's 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.

9.4 OAuth

OAuth is an authorization protocol that enables applications and devices to access HTTP[8] 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.[9] 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 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 24 :

[8] Using a transport protocol other than HTTP is undefined by the specification.
[9] OAuth 2.0 evolved from the OAuth WRAP [HTEG10] profile which has been deprecated.

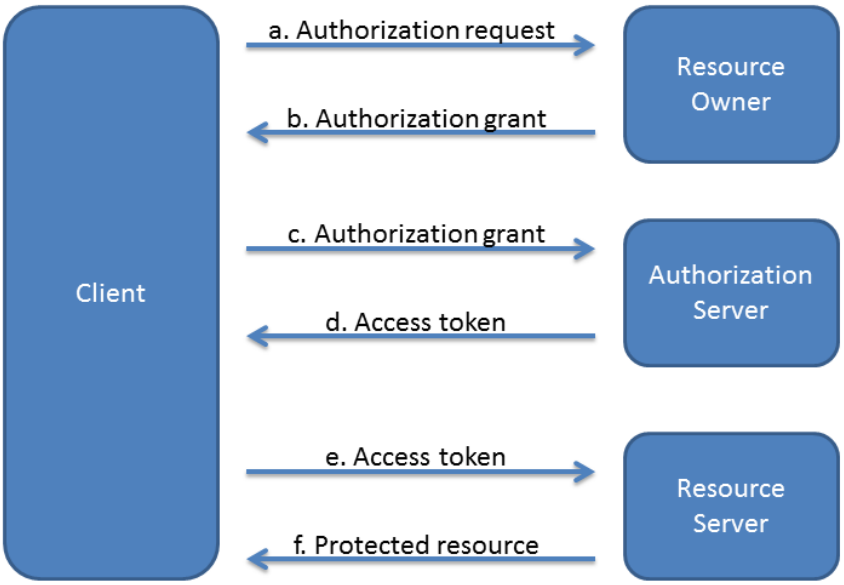

Figure 24: OAuth 2.0 protocol flow

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 sections.[10] 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 client's behaviour.

9.4.1 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.

[10]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.

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=PEFzc2VGlv...
```

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 client's presentation behaviour would be dealt with at the application level (and might be fixed for an application).

9.4.2 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].[11]

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 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.

9.5 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 [X50] is a popular PKI standard[12] 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.

[11] 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.

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

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 certificate's 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 certificate's 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 certificate's serial number does not appear on a Certificate Revocation List (CRL), or by querying an Online Certificate Status Protocol (OCSP) responder.[13] See Figure 25.

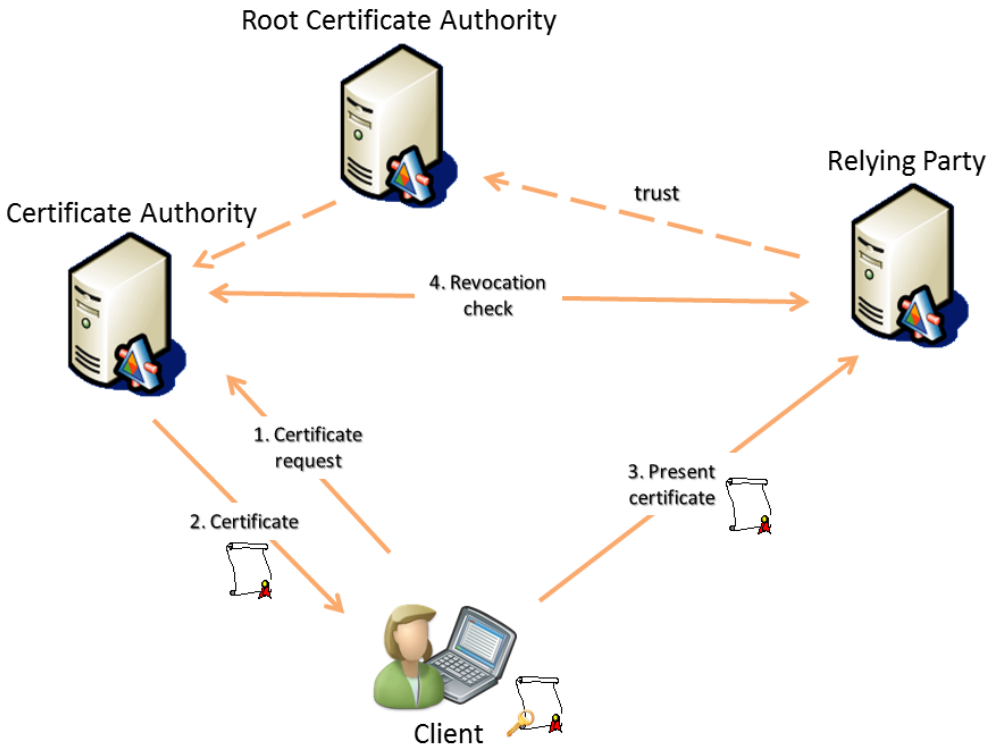

Figure 25: X.509 protocol flow

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 user's 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-ABC's 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.

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

The second and preferred[14] 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 certificate's information into attributes:
   (a) The certificate's expiration date is encoded in an attribute.
   (b) The certificate's serial number is encoded as the revocation handle.
   (c) The revocation information (e.g., the CRL endpoint)[15] 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 identifier[16] 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).[17]
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.

9.6 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 user's identity without the user's 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,

[14] 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.

[15] 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.

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

[17] Alternatively, an ABC Revocation Authority could create an accumulator for the revoked values.

or in some WS-Trust profiles), this is more complicated but still could be possible.[18] 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.
- 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 user's 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 won't 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.

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

Glossary

Attribute	A piece of information, possibly certified by a credential, describing a characteristic of a natural person or entity, or of the credential itself. An attribute consists of an attribute type determining the semantics of the attribute (e.g., first name) and an attribute value determining its contents (e.g., John).
Certified pseudonym	A verifiable pseudonym based on a user secret that also underlies an issued credential. A certified pseudonym is established in a presentation token that also demonstrates possession of a credential bound to the same User (i.e., to the same user secret) as the pseudonym.
Credential	A list of certified attributes issued by an Issuer to a User. By issuing a credential, the Issuer vouches for the correctness of the contained attributes with respect to the User.
Credential specification	A data artifact specifying the list of attribute types that are encoded in a credential.
Inspection	An optional feature allowing a presentation token to be de-anonymized by a dedicated Inspector. At the time of creating the presentation token, the User is aware (through the presentation policy) of the identity of the Inspector and the valid grounds for inspection.
Inspection grounds	The circumstances under which a Verifier may ask an Inspector to trace the User who created a given presentation token.
Inspection Requester	Entity requesting an inspection from the Inspector, asserting that inspection is compliant with the inspection grounds specified or is legally required. In most cases this will be the Verifier, but also may be the police, or other legally authorized entity.
Inspector	A trusted entity that can trace the User who created a presentation token by revealing attributes from the presentation token that were originally hidden from the Verifier.
Issuance key	The Issuer's secret cryptographic key used to issue credentials.
Issuer	The party who vouches for the validity of one or more attributes of a User, by issuing a credential to the User.
Issuer parameters	A public data artifact containing cryptographic and other information by means of which presentation tokens derived from credentials issued by the Issuer can be verified.
Key binding	The property of binding a credential to a user's secret key, so that the credential cannot be used in a presentation without knowing the secret key. Key binding can be used to prevent credential pooling and, by storing the secret key on trusted hardware, to bind the credential to a physical token.
Linkability	See *unlinkability*.

Personal data	"Personal data' shall mean any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity", Art. 2 (a) of Directive 95/46/EC. Within this deliverable personal data is the terminology used for legal considerations. See also *Personally Identifiable Information* .
Presentation policy	A policy created and published by a Verifier specifying the class of presentation tokens that the Verifier will accept. The presentation policy contains, among other things, which credentials from which Issuers it accepts and which information a presentation token must reveal from these credentials.
Presentation token	A collection of information derived from a set of credentials, usually created and sent by a User to authenticate to a Verifier. A presentation token can contain information from several credentials, reveal attribute values, prove that attribute values satisfy predicates, sign an application-specific message or nonce or support advanced features such as pseudonyms, device binding, inspection, and revocation. The presentation token consists of the presentation token description, containing a technology-agnostic description of the revealed information, and the presentation token evidence, containing opaque technology-specific cryptographic parameters in support of the token.
Pseudonym	See *verifiable pseudonym.*
Pseudonym scope	A string provided in the Verifier's presentation policy as a hint to the User which previously established pseudonym she can use, or to which a new pseudonym should be associated. A single User (with a single user secret) can generate multiple verifiable or certified pseudonyms for the same scope string, but can only generate a single scope-exclusive pseudonym.
Revocation	The act of withdrawing the validity of a previously issued credential. Revocation is performed by a dedicated Revocation Authority, which could be the Issuer, the Verifier, or an independent third party. Which Revocation Authorities must be taken into account can be specified by the Issuer in the issuer parameters (Issuer-driven revocation) or by the Verifier in the presentation policy (Verifier-driven revocation).
Revocation Authority	The entity in charge of revoking credentials. The Revocation Authority can be an Issuer, a Relying Party, or an independent entity. Multiple Issuers or Verifiers may rely on the same Revocation Authority.

Revocation information	The public information that a Revocation Authority publishes every time a new credential is revoked or at regular time intervals to allow Verifiers to check that a presentation token was not derived from revoked credentials.
Revocation parameters	The public information related to a Revocation Authority, containing cryptographic information as well as instructions where and how the most recent revocation information and non-revocation evidence can be obtained. The revocation parameters are static, i.e., they do not change every time a new credential is revoked or at regular time intervals like the revocation information and non-revocation evidence (may) do.
Non-revocation evidence	The User-specific or credential-specific information that the user agent maintains, allowing it to prove in presentation tokens that the credential was not revoked. The non-revocation evidence may need to be updated either at regular time intervals or when new credentials are revoked.
Scope	See *pseudonym scope.*
Scope-exclusive pseudonym	A certified pseudonym that is guaranteed to be cryptographically unique per scope string and per user secret. Meaning, from a single user-bound credential, only a single scope-exclusive pseudonym can be generated for the same scope string.
Traceability	See *untraceability.*
Unlinkability	The property that different actions performed by the same User, in particular different presentation tokens generated by the same User, cannot be linked to each other as having originated from the same User.
Untraceability	The property that an action performed by a User cannot be traced back to her identity. In particular, the property that a presentation token generated by a User cannot be traced back to the issuance of the credential from which the token was derived.
User	The human entity who wants to access a resource controlled by a verifier and obtains credentials from Issuers to this end.
User agent	The software entity that represents the human User and manages her credentials.
User secret	A piece of secret information known to a User (either a strong random secret or a human-memorizable password or PIN code) underlying one or more issued credentials or pseudonyms. A presentation token involving a pseudonym or a user-bound credential implicitly proves knowledge of the underlying user secret.
Verifiable pseudonym	A public identifier derived from a user secret allowing a User to voluntarily link different presentation tokens created by her or to re-authenticate under a previously established pseudonym by proving knowledge of the user secret. Multiple unlinkable pseudonyms can be derived from the same user secret.

Verifier	The party that protects access to a resource by verifying presentation tokens to check whether a User has the requested attributes. The Verifier only accepts credentials from Issuers that it trusts.

Acronyms

ABCs	Attribute Based Credentials
ABCE	ABC Engine
CA	Certificate Authority
CE	Crypto Engine
ENISA	European Network and Information Security Agency
FP7	Framework Programme 7
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure (HTTP secured by TLS or SSL)
ID	Identifier
Idemix	IBM Identity Mixer
ICT	Information and Communications Technology
IdM	Identity Manager
ISO	International Organisation for Standardisation
IdSP	Identity Service Provider
PET	Privacy Enhancing Technology
PRIME	Privacy and Identity Management for Europe
PrimeLife	Privacy and Identity Management in Europe for Life
PIN	Personal Identification Number
RP	Relying Party
SCI	Smartcard Interface
SSL	Secure Sockets Layer
STS	Secure Token Service
TLS	Transport Layer Security
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

References

[Alv01] H. Alvestrand. Rfc 3066: Tags for the identification of languages. `http://tools.ietf.org/rfc/rfc3066.txt`, January 2001.

[BBF+02] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML-Signature Syntax and Processing. `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`, February 2002.

[BCD+13] Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Anja Lehmann, Gregory Neven, and Dieter Sommer. H2.3 – abc4trust crypto architecture. ABC4Trust Heartbeat H2.3, 2013. Available from `https://abc4trust.eu`.

[BCGS09] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard java card. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 600–610. ACM, 2009.

[BDDD07] Stefan Brands, Liesje Demuynck, and Bart De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. In *Information Security and Privacy*, pages 400–415. Springer, 2007.

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

[Bra] Stefan Brands. The id corner blog. the problem(s) with openid. `http://www.untrusted.ca/cache/openid.html`.

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

[Bra94] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology – CRYPTO'93*, pages 302–318. Springer, 1994.

[Bra97] Scott Bradner. Rfc 2119: Key words for use in rfcs to indicate requirement levels. `http://www.rfc-editor.org/rfc/rfc2119.txt`, March 1997.

[Bra00] Stefan A Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. MIT Press, 2000.

[Cam06] Jan Camenisch. Protecting (anonymous) credentials with the trusted computing group's tpm v1. 2. In *Security and Privacy in Dynamic Environments*, pages 135–147. Springer, 2006.

[CC+08] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *Advances in Cryptology – ASIACRYPT 2008*, pages 234–252. Springer, 2008.

[CDF+] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. Openpgp message format. `http://www.rfc-editor.org/rfc/rfc4880.txt`.

[CG08] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 345–356. ACM, 2008.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communication of the ACM*, 28(10):1030–1044, 1985.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 201–210. ACM, 2006.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks*, pages 141–155. Springer, 2006.

[CKL+14] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssø e Mikkelsen, Gregory Neven, and Michael østergaard Pedersen. Scientific comparison of ABC protocols part I - formal treatment of privacy-enhancing credential systems. ABC4Trust Deliverable D3.1, 2014. Available from `https://abc4trust.eu`.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography–PKC 2009*, pages 481–500. Springer, 2009.

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

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001*, pages 93–118. Springer, 2001.

[CL02a] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[CL02b] J. Camenisch and A. Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In M. Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[CL02c] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – CRYPTO 2002*, pages 61–76. Springer, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO 2004*, pages 56–72. Springer, 2004.

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

[CS02] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In L. R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *LNCS*, pages 45–64. Springer, 2002.

[CS03a] J. Camenisch and V. Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In D. Boneh, editor, *CRYPTO*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.

[CS03b] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology – CRYPTO 2003*, pages 126–144. Springer, 2003.

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

[DFLP07] Nelly Delessy, Eduardo B. Fernandez, and Maria M. Larrondo-Petrie. A pattern language for identity management. In *Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, ICCGI '07, pages 31–, Washington, DC, USA, 2007. IEEE Computer Society.

[DW10] Arkajit Dey and Stephen Weis. Pseudoid: Enhancing privacy in federated login. In *Hot Topics in Privacy Enhancing Technologies*, pages 95–107, 2010.

[Eck10] Peter Eckersley. How unique is your web browser? In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[Fac] Facebook login. https://developers.facebook.com/products/login/.

[Fid] Fido alliance. http://fidoalliance.org.

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

[FP10] Walter Fumy and Manfred Paeschke. *Handbook of EID Security: Concepts, Practical Experiences, Technologies*. John Wiley & Sons, 2010.

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

[Har04] Russell Hardin. *Trust and trustworthiness*, volume 4. Russell Sage Foundation, 2004.

[Har12] D. 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 Josang and Stephane Lo Presti. Analysing the relationship between risk and trust. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Second International Conference on Trust Management (iTrust 2004)*, volume LNCS 2, pages 135–145. Springer, 2004. Event Dates: March 29 - April 1st 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 *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications*, AINA '09, pages 137–145, Washington, DC, USA, 2009. IEEE Computer Society.

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

[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, 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.

[N.11] Smart N. ECRYPT II Yearly Report on Algorithms and Keysizes (2010-2011). Katholieke Universiteit Leuven (KUL). Deliverable SPA-17. rob. Online: http://www.ecrypt.eu.org/documents/D.SPA.17.pdf, June 2011.

[NA09] European Network and Information Security Agency. Privacy features of european eid card specifications – position paper. http://www.enisa.europa.eu/act/it/ privacy-and-trust/eid/eid-cards-en, February 2009.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology – CT-RSA 2005*, pages 275–292. Springer, 2005.

[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.

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

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

[RS86] Rabin and Shallit. Randomized Algorithms in Number Theory. *Communications in Pure and Applied Math*, 39:239–256, 1986.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[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.

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

[Sta05] OASIS Standard. extensible access control markup language (xacml) version 2.0. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`, February 2005.

[Sta09a] OASIS Standard. Identity metasystem interoperability version 1.0. `http://docs.oasis-open.org/imi/identity/v1.0/identity.html`, July 2009.

[Sta09b] OASIS Standard. Ws-trust 1.4. `http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html`, February 2009.

[UPW11] U-prove ws-trust profile v1.0. `http://www.microsoft.com/u-prove`, March 2011.

[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.

[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.

[X50] X.509 : Information technology - open systems interconnection - the directory : Public/key and attribute certificate frameworks. `http://www.itu.int/rec/T-REC-X.509/en`.