Mixing Identities with Ease

Patrik Bichsel, Jan Camenisch

Mixing Identities with Ease

Patrik Bichsel and Jan Camenisch*

IBM Research, Switzerland
{pbi,jca}@zurich.ibm.com
http://www.zurich.ibm.com

Abstract. Anonymous credential systems are a key ingredient for a secure and privacy protecting electronic world. In their full-fledged form, they can realize a broad range of requirements of authentication systems. However, these many features result in a complex system that can be difficult to use. In this paper, we aim to make credential systems easier to employ by providing an architecture and high-level specifications for the different components, transactions and features of the identity mixer anonymous credential system. The specifications abstract away the cryptographic details but they are still sufficiently concrete to enable all features. We demonstrate the use of our framework by applying it to an e-cash scenario.

Key words: Anonymous Credentials, Architecture, Privacy.

1 Introduction

We all increasingly use electronic services in our daily lives. To do so, we have no choice but to provide plenty of personal information for authorization, billing purposes, or as part of the terms and conditions of service providers. Dispersing all these personal information erodes our privacy and puts us at risk of abuse of this information by criminals. Therefore, these services and their authentication mechanisms should be built in a way that minimizes the disclosed personal information. Indeed, over the past decades, the research community has come up with a large number of privacy-enhancing technologies that can be employed to this end.

A key privacy-enhancing technology are anonymous credential systems [19, 5, 14]. In their basic form, they allow a user to obtain a credential from an issuing authority, attesting to her attributes such as her birth date or access rights. Later, she can use the credential to selectively reveal a subset of the attested attributes, without revealing *any* other information (*selective disclosure*). In particular, even if she uses the same credential repeatedly, the different uses cannot be linked to each other. It has been proven that anonymous credentials can be used in practice today (even on Java Cards [2]) and publicly available implementations exist (e.g., `www.primelife.eu/results/opensource/33-idemix`).

* This work has been funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483.

The literature provides a number of cryptographic building blocks that allow one to expand this basic functionality; in fact, many of them are needed to meet the practical requirements of a modern public key infrastructure. These include:

Property proofs about attributes allow a credential owner to prove properties about her attributes such as that one attribute is larger than another one (even if they are contained in different credentials). This allows an owner to prove, e.g., that her age lies in a certain range [8], or that an attribute is a member of a given set [9].

Usage limitation such as ensuring that an owner can use a credential (i.e., proof ownership of a credential) only a limited number of times (e.g., for e-cash) [11] or a number of times within some context [10, 12] (e.g., twice per hour or once per election). Furthermore, using domain specific pseudonyms enables the implementation of usage restrictions as it makes a user linkable within a given domain.

Revocation of credentials can be implemented using dynamic accumulators [13, 16] or a form of credential revocation lists [3, 6, 21]. This is necessary for instance to withdraw the right associated with the ownership of the credential or after leakage of the master secret of a user.

Revocation of anonymity in case of abuse of (the rights granted by) a credential can be implemented using techniques from [14].

Verifiable encryption of attributes under some third party's public key [17]. This feature constitutes a generalization of anonymity revocation assuming the user's identity is an attribute encrypted for the party in charge of anonymity revocation. It is a means to control the dispersal of attributes using a trusted entity.

These mechanisms can be combined in various ways. Thereby they allow us to build a multitude of privacy-enhancing applications such as anonymous e-cash, petition systems, pseudonymous reputations systems, or anonymous and oblivious access control systems. It is an enormous challenge to find the balance between offering the whole spectrum of functionality and abstracting away from the cryptographic details when implementing an anonymous credential system. Furthermore, when designing the application programming interface we should require no knowledge of cryptography but only familiarity with the concepts that it realizes. However, reducing complexity bears the risk of tailoring the library towards certain application scenarios which we must avoid. In addition, we require our specifications to be extensible and to go along with current standards.

At IBM Research – Zurich, we have implemented most of the protocols and mechanisms described before. This implementation has been growing over the last couple of years and it has been re-designed and re-implemented several times, the current publicly available version is the forth complete iteration. We were fortunate to receive feed back from a considerable number of universities who have used different versions of our code to build various prototypes. Also, our code has been used in the PRIME and PrimeLife projects to build prototypes, which allowed us to test and discuss our implementation. We believe that the

current version provides a good compromise between providing access to the features while ensuring the usability for application developers.

This paper describes the architecture and specification languages for all the interactions of our anonymous credential system called *Identity Mixer*. Due to its generality, the architecture and specification languages also apply to other anonymous credential systems supporting (a subset of) the described features including the one by Brands [5]. In addition, our proposal is extensible, i.e., we allow for the specification of low-level features (e.g., commitments, pseudonyms, and verifiable encryption) that can be utilized to implement a high-level functionality (e.g., reputation system). This fosters the usage of the various functionalities described before and simplifies building applications upon them.

We refer to [22] for the complete set of the specification languages for the components of an anonymous credential system. Here we will discuss the most complex ones and depict them in a human readable pseudo code form rather than providing the XML version used by our implementation. We will incorporate our specification language in the next release of *Identity Mixer* (`www.primelife.eu/results/opensource/33-idemix`), where several examples for each component will be available. We will demonstrate our framework by elaborating the example of building an e-cash scenario.

Related Work. Camenisch and Van Herreweghen [18] describe the basic functions and protocols of an anonymous credential system and define the APIs for them. The system they describe provides only the very basic functionalities (i.e., selective disclosure and anonymity revocation). We provide much more extensive (and less general) specifications at a slightly lower level, i.e., we do not directly specify anonymity revocation but provide the more flexible verifiable encryption primitive that can be used for the same purpose (cf. Section 1).

Bangerter et al. [1] provide a cryptographic framework that allows security researchers to design privacy-protecting systems and protocols. In this work we go further: we describe our (Java) implementation of all the building blocks described by Bangerter et al. and describe the architecture and specification languages that enable the design and realization of privacy-protecting systems based on our *Identity Mixer* library.

There are various approaches to specify cryptographic objects such as credentials or authentication information. We provide a specification that is general enough to allow to incorporate, for example, X.509 certificates. On the other hand our proof specification could be extended to comply with the OASIS SAML standard. Consequently, we align very well with current standards while still extending their current functionality to a full-fledged anonymous credential system.

Finally, Microsoft has recently released the protocol specification for U-Prove [7], the credential scheme by Brands [5]. That document specifies the cryptographic protocol for issuing credentials and proving possession of a credential with selective attribute disclosure. We provide a much more extensive specification as *Identity Mixer* allows for more features compared to U-Prove (cf. Section 5).

Organization of this Paper. In Section 2 we give a high-level description of anonymous credential systems. Next, we describe the architecture in Section 3, which consists of (1) a description of the different components of the *Identity Mixer* (*idemix*) credential system, (2) a detailed analysis of how those components are used in the *idemix* protocols, and (3) the specification language for the components. We give an example showing how we make use of the specifications to realize an e-cash scheme in Section 4. Section 5 provides a comparison to the U-Prove specification finally we provide an outlook on the integration with current authentication technology in Section 6.

2 Overview of an Anonymous Credential System

An anonymous credential system involves the roles of *issuers*, *recipients*, *provers* and *verifiers* (or *relying parties*). Parties acting in those roles execute the issuing protocol, where a credential for the recipient is created by the issuer, or the proving protocol, where the owner creates a proof on behalf of the verifier. An entity (e.g., user, company, government) can assume any role during each protocol run. For instance, a company can act as verifier and run the proof protocol with a user before assuming the role of the issuer and running the issuance protocol (possibly with the same user). Finally, an extended credential system requires the role of trusted third parties who performs tasks such as anonymity revocation, credential revocation, or decryption of (verifiably) encrypted attributes. Usually organizations or governments assume the roles the issuer, verifier and trusted party, and natural persons the ones of recipient and prover.

Note, all parties in an anonymous credential system agree on general system parameters that define the bit length of all relevant parameters as well as the groups that will be used. In practice, these parameters can be distributed together with the code and they must be authenticated.

To participate a user needs to choose her *master secret key* based on the group parameters of the system. This secret allows her to derive pseudonyms, which she can use similar to a session identifier, i.e., it allows the communication partner to link the actions of the user. However, the user can create new pseudonyms at her discretion and all pseudonyms are unlinkable unless the user proves that they are based on the same master secret key. Certain scenarios require one user only having one pseudonym with an organization, where we call such pseudonym a domain pseudonym. In addition to being used for pseudonym generation, the master secret will be encoded into every credential. This constitutes a sharing prevention mechanism as sharing one credential implies sharing all credentials of a user.

The setup procedure for issuers and trusted parties consists of generating public key pairs, create a specification of the services they offer and publish the specification as well as the public key. As an example, an issuer publishes the structure(s) of the credential it issues and its public key.

Let us now elaborate on the issuing and the proving protocol. The credential *issuance protocol* is carried out between an issuer and a recipient with the result

of the recipient having a credential. The credential consists of a set of attribute values as well as cryptographic information that allows the owner of the credential (i.e., the recipient) to create a *proof of possession* (also called 'proof of ownership' or 'proof'). When encoding the values into a credential, the issuer and recipient agree on which values the issuer learns and which will remain unknown to it, i.e., they agree on a credential structure. In addition, they agree on the values that will be encoded.

The *proving protocol* requires a prover and a verifier to interact, i.e., the owner of one or several credentials acts as prover in the communication with a verifier. Firstly, the entities define (interactively) what statement will be proved about which attribute value. Secondly, the prover compiles a cryptographic proof that complies with the statements negotiated before. Thirdly, the verifier checks if the given proof is compiled correctly. The first step is a very elaborate process that is outside of the scope of this paper. To indicate the complexity remember that a proof can range from merely proving possession of a credential issued by some issuer to proving detailed statements about the individual attributes. Our specification focuses on the language that expresses the results from the negotiation phase as well as the second and third step from before. The difficulties here lie in the fact that a proof may be linked to a pseudonym of the user's choice or it may release a verifiable encryption of some attribute value under a third party's public key. In addition, we need to be able to express statements about attributes that will be proved. Finally, the protocols for proving possession of credentials and issuing credentials may be combined. In particular, before issuing a new credential, the issuer may require the recipient to release certified attribute values, i.e., prove that she holds a credential issued by another party.

3 Architecture & Specifications

In this section we first discuss the components of *idemix*, then we show how the components are used in the protocols, and finally we provide the specification of the objects used in those protocols. In particular, we introduce the specification languages for the information that needs to be passed between participants.

3.1 Components of *idemix*

An extended anonymous credential system consists of many components. We will introduce them starting with the attributes that are contained in credentials. Continuing with the credentials we will finish the discussion with the optional components such as commitments and pseudonyms, which are used to implement extensions.

Attributes. We denote an attribute a_i as the tuple consisting of *name*, *value* and *type*, i.e., $a_i = \{n_i, v_i, t_i\}$. The name must be unique within its scope (e.g., a credential structure or a commitment), which will allow us to refer to the attribute using that name and the scope. The value refers to the content of

the attribute, which is encoded as defined by the type. For each type we define a mapping from the content value to a value that can be used in the cryptographic constructions. Currently, *idemix* supports the attribute types *string*, *int*, *date1900s*, and *enum*. Encoding a string to be used in a group $\mathbb{G}$ with generator g can be achieved by use of a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$. Integers do not require such mapping unless they are larger than the order of the group used by *idemix*. In such case, the value will be encoded into several attributes. We chose the granularity of the currently implemented date type as a second and set the origin to 1.1.1900. Enumerated attributes are mapped using a distinct prime according to the description in [9].

Credentials. We denote the set of attributes together with the corresponding cryptographic information as credentials. We classify attributes contained in credentials depending on which party knows the value of an attribute. More concretely, the owner of a credential always knows all attribute values but the issuer or the verifier might not be aware of certain values. During the issuance of a credential we distinguish three sets of attributes as the issuer might *know* a value, have a *commitment* of the value, or the value might be completely *hidden* to him. Let us denote the these sets of attributes by A_k, A_c, and A_h, respectively. Note that the user's master secret, as introduced in Section 2, is always contained in A_h.

When creating a proof of possession of credentials, the user has the possibility to reveal only a selected set of attributes. Therefore, we distinguish the *revealed* attributes, which will be learned by the verifier, from the *unrevealed* attributes. We call the two sets of attributes during the proving protocol A_r and $A_{\overline{r}}$. Note, that each attribute can be assigned to either A_r or $A_{\overline{r}}$ independently of all previous protocols and, in particular, independently of the issuing protocol.

Commitments and Representations of Group Elements. With commitments [20] a user can commit to a value v, which we denote as $C = \mathrm{Comm}(v)$. The commitment has a hiding and a binding property, where hiding refers to the recipient not being able to infer information about v given C and binding refers to the committer not being able to convince a recipient that $C = \mathrm{Comm}(v')$ for a $v' \neq v$. Either of the two properties can be information theoretically achieved where the other will hold computationally.

In our context the bases of a commitment are selected from the bases of the group parameters. When we need the more general version of arbitrarily chosen bases, we call the corresponding object a representation. Where the name is chosen because such objects are representations of group elements w.r.t. other group elements. Representations enable the integration of almost arbitrary proof statements, e.g., they are building blocks for building e-cash schemes or (more generally) cloning prevention for credentials.

Pseudonyms and Domain Pseudonyms. We denote randomized commitments to the master secret as pseudonyms. Thus, a pseudonym is similar to a public key in a traditional PKI and can be used to establish a relation with an

organization, e.g., in case a user wants an organization to recognize her as a returning user. In contrast to an ordinary public-secret key pair, however, the user can generate an unlimited number of pseudonyms based on the same master secret without the link between those pseudonyms (i.e., the master secret key) becoming apparent.

A domain pseudonym is a special kind of pseudonym in the sense that a user can create exactly one pseudonym w.r.t. one domain. The domain is specified by a verifier, which allows it to enforce usage control for its domain. Note that no two pseudonyms (be it domain or ordinary) are linkable unless a user proves that the underlying master secret key is the same.

3.2 Protocols

The basic building block of *idemix* is the Camenisch-Lysyanskaya (CL) signature scheme [14, 15] which largely determines the protocols. The signature scheme supports blocks of messages, i.e., with a single signature many messages can be signed. In a simple credential, thus, each attribute value is handled as a separate message. A more elaborate idea is to use a compact encoding as in [9] to combine several attribute values into one message. The signature scheme also supports "blind" signing, where the recipient provides the issuer only with a commitment of the attribute value that will be included in the credential. This is used for attributes of the set A_c. Credentials are always issued to a recipient authenticated with a pseudonym, which ensures that the user's master secret gets "blindly" embedded into the credential.

The distinguishing feature of a CL signature is that it allows a user to prove possession of a signature without revealing the underlying messages or even the signature itself using efficient zero-knowledge proofs of knowledge. Thus, when a prover wants to convince a verifier that she has obtained a credential from an issuer and selectively reveal some of the messages of the credential, she employs a zero-knowledge proof stating that she "knows" a signature by the issuing organization and messages such that signature is valid. As the proof is "zero-knowledge", the user can repeat such a proof as many times as she wants and still it is not possible to link the individual proofs. This statement even holds if the verifier and the issuer pool their information. Of course, a user can also prove possession of several credentials (acquired from different issuers) at once to a verifier and then prove that these credentials share some messages (without revealing the messages).

Let us specify the inputs of the protocols. The issuance protocol requires two inputs for either participant, namely an issuance specification and a set of values. The former is the same for both participants as it defines the issuance process, i.e., it links to the definition of the structure of the credential to be issued or the system parameters. The latter are the values assigned to the attributes of the newly created credential. As we pointed out already, the issuer may operate on a set of the values that differs from the one used by the receiver as A_h are not know to it and for values in A_c the issuer only knows a commitment. Note, the issuer may additionally input cryptographic components into the protocol. This

is useful when combining the issuance and the proving protocol, e.g., the issuer can input a commitment received during a previous run of the proving protocol. It can use the value "sealed" in the commitment as the value of an attribute from the set A_c.

The proving protocol most notably makes use of the proof specification, which the prover and the verifier both must provide as input to the protocol. This specification defines all details of the proof. In addition, it links to the necessary elements for compiling and verifying such proof. The prover provides all credentials referenced in the proof specification as input and the verifier uses the credential structures (cf. Section 3.3) to verify the proof. The cryptographic proof object will be provided to the verifier during the protocol run.

Extensions to the Issuing Protocol. The issuing protocol has fewer degrees of freedom compared to the proving protocol. This results from the credential structure setting many limitations on the protocol. For instance, the structure defines which attributes belong to which set (i.e., A_k, A_c, or A_h). Still we provide a mechanism for extending the issuing protocol and use it for implementing a feature that enables efficient updates of the attribute values (A_k) contained in a credential.

Credential Updates. As the issuing protocol is interactive (and for security reasons might need to be executed in a particularly protected environment) re-running it would be impractical in many cases. Rather, *idemix* offers an non-interactive method to update credentials where the issuer publishes update information for credentials such that attribute values are updated if necessary.
This feature can, e.g., be used to implement credential revocation. The mechanism that we have implemented employs epochs for specifying the life time. A credential thus expires and can be re-validated when updating the expiration date (given that the issuer provides such) [13].

Extensions to the Proving Protocol. The proving protocol requires the prover and the verifier to agree on the attribute values that will be revealed during the proof, i.e., all attributes a_i are contained in either A_r or $A_{\overline{r}}$ such that $A_r \cap A_{\overline{r}} = \emptyset$. In addition, the verifier may define what partial information about the attributes $a_i \in A_{\overline{r}}$ has to be proved, where partial information denotes:

Equality. A user can prove equality of attribute values, where the values may be contained in different credentials. In particular, equality proofs can be created among values that are contained in any cryptographic object such as credentials or commitments. As an example, a user can compute a commitment to a value v, with $C = \mathrm{Comm}(v)$. Assuming a value v' is contained in a credential, the user can prove that $v = v'$.

Inequality. Allows a user to prove that an attribute value is larger or smaller than a specified constant or another attribute value.

Set Membership. Each attribute that is contained as a compact encoding as described in [9] enables the user to prove that the attribute value does or does not lie in a given set of values.

Pseudonym. A pseudonym allows a user to establish a linkable connection with a verifier. Furthermore, domain pseudonyms allow a verifier to guarantee that each user only registers one pseudonyms w.r.t. his domain.

Verifiable Encryption. A user can specify an encryption public key under which an attribute value contained in a credential shall be (verifiably) encrypted.

3.3 Specification Languages

As pointed out in Section 1, one challenge when designing the specification languages is to abstract from the underlying cryptography while allowing access to flexible primitives that enable developers to build a broad range of systems. The necessity of both parties having certain information (e.g., the credential structure) in order to extract the semantic of a proof presents another difficulty. For instance, a verifier needs to know the issuer of a credential, the attributes names, their order or their encoding within a credential used in a proof. Thus, it is essential to separate the structural information from the data, where the latter may remain unknown to one communication partner. We will not introduce such separation for objects that do not require it (e.g., public keys). Our specifications are in XML and each component uses and XML schema to define its general structure. Note that the information acquired through unsecured channels needs to be authenticated, which can be attained using a traditional PKI.

System and Group Parameters. The system and group parameters are specified as a list of their elements. In addition, the group parameters contain a link to the system parameters. Both system and group parameters need to be authenticated.

Issuer Key Pair. The issuer key pair consists of a public key and a private key, where mostly the specification of the public key is of interest as the private key as it is never communicated. The public key links to the group parameters with respect to which it has been created. Note that apart from the public key, an issuer needs to publish the structures of the credentials it issues. Even though this information might be included in the public key, we suggest to create a designated file.

Credentials. As mentioned earlier, we decompose credentials into a *credential structure*, which is the public part, and the *credential data*, which is private to the owner of the credential. In addition a credential data object is partially populated and sent to the verifier during the proving protocol. This decomposition is needed in the issuing process, when the credential data has not been created, as well as in the verification protocol, where the verifier does only get to know a selected subset of the credential data.

In Fig. 1 we describe the credential structure. It contains (1) references to the XML schema and the issuer public key and (2) information about the structure

```
References{
   Schema = http://www.zurich.ibm.com/security/idemix/credStruct.xsd
   IssuerPublicKey = http:www.ch.ch/passport/ipk/chPassport10.xml
}
Attributes{
   Attribute { FirstName, known, type:string }
   Attribute { LastName, known, type:string }
   Attribute { CivilStatus, known, type:enum }
      { Marriage, Widowed, Divorced  }
   Attribute { Epoch, known, type:int }
}
Features{
   Domain { http://www.ch.ch/passport/v2010 }
   Update  { http://www.ch.ch/passport/v2010/update.xml }
}
Implementation{
   PrimeFactor { CivilStatus:Marriage = 3 }
   PrimeFactor { CivilStatus:Widowed = 7 }
   PrimeFactor { CivilStatus:Divorced = 17 }
   AttributeOrder { FirstName, LastName, CivilStatus, Epoch }
}
```

Fig. 1. Example credential structure where we assume this structure being located at `http://www.ch.ch/passport/v2010/chPassport10.xml` and corresponding to a Swiss passport. For the XML version refer to [22].

of a credential, which is needed to extract the semantics of a proof. We partition the latter into the attribute, feature, and implementation specific information.

The attribute information defines name, issuance mode (cf. Section 3.1), and type (e.g., string, enumeration) of each attribute. The feature section contains all relevant information about extensions such as domain pseudonyms. Finally, the implementation specific information is mapping general concepts to the actual implementation. As an example, enumerated attributes are implemented using prime encoded attributes [9], which requires the assignment of a distinct prime to each possible attribute value.

The credential data most importantly refers to the credential structure that it is based on. In addition, it contains the (randomized) signature and the values of the attributes. Figure 2 shows a credential created according to the structure provided in Fig. 1 and corresponding to the proof specification given in Fig. 3.

```
References{
   Schema = http://www.zurich.ibm.com/security/idemix/cred.xsd
   Structure = http://www.ch.ch/passport/v2010/chPassport10.xml
}
Elements{
   Signature { A:4923...8422, v:3892...3718, e:8439...9239 }
   Features { Update:http://www.ch.ch/passport/v2010/update/7a3i449.xml }
   Values { FirstName:Patrik; LastName:Bichsel; ... }
}
```

Fig. 2. This example shows a Swiss passport credential. Note that the owner who will act as prover knows all the attribute values.

Credential Updates. Credential update information is twofold: it consists of (1) general information detailing, e.g., which attributes will be updated, and (2) the information specific to each credential. The former is linked from the credential structure (see Fig. 1), the latter is referenced from the credential (see Fig. 2). Only attributes from the set A_k can be updated.

Commitment and Representation. A commitment and a representation, similar to a credential, consist of a set of values. We assume that the bases for the commitments are listed in the same file as the group parameters. Thus, they use a reference to link to the corresponding parameters. The representations, however, list their bases in addition to the list of exponents.

Pseudonym and Domain Pseudonym. As pseudonyms are a special case of commitments, they also contain a reference to the group parameters they make use of. In addition, at the user's side pseudonyms contain the randomization exponent value. Domain pseudonyms additionally link to their domain.

Verifiable Encryption. A verifiable encryption is transferred to a verifier and (if necessary) to the trusted party for decryption. It contains the public key used for the encryption as well as the name used in the proof specification, the label and the ciphertext of the encryption.

Protocol Messages. When running the protocols, there are several messages that are passed between the communication partners. The specification of those objects contains the reference to the schema and the cryptographic values. Each cryptographic value is assigned a name such that the communication partner can retrieve the values easily.

Issuance Specification. Issuing a credential most importantly requires a credential structure and a set of attribute values. As introduced in Section 3.1, the set of values from the issuer may differ from the set of the recipient. More specifically, values of attributes in A_k are known to both recipient and issuer and values of attributes $a_i \in A_h$ are only known to the recipient. For each attribute $a_i \in A_c$ the recipient knows the corresponding value v_i and the issuer only knows a commitment $C = \mathrm{Comm}(v_i)$. We define the issuance modes *known*, *hidden*, and *committed* in the credential structure to denote the set an attribute belongs to. The reason for defining the issuance mode in the credential structure is to unify the issuance modes between different recipients.

As the majority of the information used in the issuance protocol is defined by the credential structure, the issuance specification is only needed to implement advanced features (e.g., binding a proving and an issuing protocol).

Proof Specification. The proof specification is more elaborate than the issuing specification as the *idemix* anonymous credential system supports many features that require specification. Thus, even when using a specific credential we can imagine a broad range of different proofs to be compiled. We start by specifying an identifier for each distinct value that will be included in a proof. Also, we specify the attribute type of each identifier, where the protocol aborts if the type

```
Declaration{ id1:unrevealed:string; id2:unrevealed:string;
             id3:unrevealed:int; id4:unrevealed:enum;
             id5:revealed:string}
ProvenStatements{
   Credentials{
      randName1:http://www.ch.ch/passport/v2010/chPassport10.xml =
          { FirstName:id1, LastName:id2, CivilStatus:id4 }
      randName2:http://www.ibm.com/employee/employeeCred.xml =
          { LastName:id2, Position:id5, Band:5, YearsOfEmployment:id3 }
   Enums{           randName1:CivilStatus = or[Marriage, Widowed] }
   Inequalities{    {http://www.ibm.com/employee/ipk.xml, geq[id3,4]} }
   Commitments{     randCommName1 = {id1,id2} }
   Representations{ randRepName = {id5,id2; base1,base2} }
   Pseudonyms{      randNymName; http://www.ibm.com/employee/ }
   VerifiableEncryptions{ {PublicKey1, Label, id2} }
   Message { randMsgName = "Term 1:We will use this data only for ..." }
}
```

Fig. 3. Example proof specification using a Swiss passport and an IBM employee credential.

of the identifier and the type of an attribute that it identifies do not match. In addition to identifiers, we allow for constants in the proof specification.

We start the definition of the statements to be proved with a list of credentials that the user proves ownership of (i.e., the user proves knowledge of the underlying master secret key). Next, we assign attribute identifiers or constants to the attributes, where the constants will cause an equality proof w.r.t. the constant. Using the same identifier several times creates an equality proof among those attributes (e.g., `id2` is used within two credentials). Note that we only need to assign an identifier to attributes that are either revealed or partial information is proved.

Apart from the equality proofs all proofs are specified explicitly. Let us begin with the proofs of set membership for enumerated attributes, where the *idemix* library supports the *and*, *or*, and *not* operators. Those operators can be used on the set of values specified in the credential structure corresponding to the given credential. Similar to set membership proofs, we allow for inequality proofs, i.e., proofs for statements of the form $v_i \circ \hat{v}$, where v_i is an attribute value, $\circ$ is the operator, and $\hat{v}$ can be a constant or another attribute value. Currently, the following operators are implemented: $<$, $>$, $\leq$, and $\geq$. Note that inequality proofs require a reference to group parameters that are to be used, which we provide by linking to an issuer public key.

Relating to the components that we describe in Section 3.1, we specify how commitments, representations, pseudonyms and domain pseudonyms relate to the identifiers. More concretely, the proof specification defines for each exponent of any of those components a corresponding identifier or constant. In addition, all the components of a proof specification are assigned random names, which is mandatory for the identification of the corresponding object in the context of a proof but prevents different proofs from becoming trivially linkable.

4 Example Use Case

In this section we describe how to implement a simple anonymous e-cash scheme with our library to give the reader an idea of how our specifications can be used. We recall the basic idea of anonymous e-cash [4]: The user has an account with the bank under some identity u. To withdraw a coin from the bank, the bank issues the user a credential with the following three attributes $(user_{id}, serial_{num}, randomizer)$ (see Fig. 4). The first one is known to the issuer and is set to U, the other two are not known to the issuer ($serial_{num}, randomizer \in A_h$) and are random values chosen from $\mathbb{Z}_q$ by the user as s and r, where q is the order of the groups used for the pseudonyms (and is part of the system parameters). Let g denote the generator of that group (which is part of the group parameters). The form of the credential can be deducted from Fig. 2.

```
References{
   Schema = http://www.zurich.ibm.com/security/idemix/credStruct.xsd
   IssuerPublicKey = http://www.bank.ch/ecash/ipk/credPK.xml
}
Attributes{
   Attribute { UserId, known, type:int }
   Attribute { SerialNum, hidden, type:int }
   Attribute { Randomizer, hidden, type:int }
}
Implementation{
   AttributeOrder { UserId, SerialNum, Randomizer }
}
```

Fig. 4. The credential structure of the e-coin issued by the bank. Let us assume that this structure is located at `http://www.bank.ch/ecash/coin.xml`.

When the user wants to spend a coin anonymously with a merchant, the user obtains from the merchant a random value $v \in \mathbb{Z}_q$, computes $a = u + rv \pmod{q}$, generates a representation with g^a being the group element, and g and g^v being the bases. Then she generates a proof to show that she owns a credential from the bank where she reveals the attribute $serial_{num}$ and proves that the attributes $user_{id}$ and $randomizer$) are also appearing in the representation. Figure 5 shows the representation object that contains the representation g^a and the bases g and g^v. We provide the proof specification in Fig. 6.

```
References{
   Schema = http://www.zurich.ibm.com/security/idemix/rep.xsd
   Params = http://www.zurich.ibm.com/security/idemix/gp.xml
}
Elements{
   Name = ksdfdsel
   Value = 8483...2939
   Bases { 3342...2059, 4953...3049 }
}
```

Fig. 5. This example shows the representation that the user created.

The user then sends (a, s) along with the proof to the merchant who accepts the coin if the proof verifies and if the representation object was indeed computed correctly. The merchant verifies the latter by re-computing the representation. Later, the merchant will deposit the coin with the bank who debits the merchant if the proof verifies. Also, the bank will check whether s has appeared before. If this is the case it will compute u from the two a and v values present in the two deposits (i.e., solve the two linear equations $a_1 = u + rv_1 \pmod q$ and $a_2 = u + rv_2 \pmod q$ for u) and then punish the user u accordingly (e.g., by charging the user for the extra spending).

```
Declaration{ u1:unrevealed:int; u2:unrevealed:int;
             r1:revealed:int }
ProvenStatements{
   Credentials{
      sfeoilsd:http:www.bank.ch/ecash/coin.xml =
          {UserId:u1, SerialNum:r1, Randomizer:u2};
   Representations{ ksdfdsel = {u1,u2; base1,base2} }
}
```

Fig. 6. The proof specification for the user when spending the e-coin at a merchant. Note that `base1` $= g$ and `base2` $= g^v$ holds.

5 Comparison with the U-Prove Specification

Microsoft has recently released the specification of the U-Prove protocols by Brands and Paquin. The specification describes the interactive issue protocol between the receiver and the issuer and the mechanisms to present and verify tokens to a verifier. The issue specification defines a number of attributes that will be contained in the token. These attributes are known by both the receiver and the issuer. At the end of the protocol, the receiver possesses a signature by the issuer on the attributes. While they call this signature a U-Prove token we would call it a credential. This issuing process is called a blind signature scheme in the literature, i.e., the issuer does not learn the token that the receiver obtains but only learns the attributes. Brands and Paquin then specify a token presentation algorithm (subset presentation proof). The input to the algorithm is the U-Prove token and the subset of the attributes that shall be disclosed (the other attributes remain hidden to the verifier). The output is an augmented U-Prove token that can then be sent to a verifier who runs the verification procedure to assert the validity of the token w.r.t. to the issuer's public key and the disclosed attributes.

Let us compare the U-Prove specifications to the ones presented in this paper. We do not attempt a cryptographic comparison here. The U-Prove issuing specification realizes a subset of our issuance specification, i.e., U-Prove requires that all attributes have to be known by the issuer, whereas in our specification, some attributes can be hidden from the issuer or only be given by commitments. Thus, it is for instance not possible with U-Prove to issue several credentials

(tokens) to the same (pseudonymous) user as this requires all credentials containing a (secret) attribute that is essentially the user's secret key and plays the role of a secret identity.

Similarly to the issuing specification, the proof specification (or token presentation specification) of U-Prove realizes a subset of ours. U-Prove only supports that a subset of the attributes can be disclosed, but does not feature proofs of statements about attributes nor does it provide the possibility to release attributes as commitments or verifiable encryption. Furthermore, U-Prove does not support proving possession of several credentials at the same time and proofs among attributes (be they disclosed or not) contained in different credentials. Furthermore, U-Prove has no support for pseudonyms. Let us finally remark that for cryptographic reasons U-Prove tokens can be presented only once (afterwards the different presentations would become linkable to each other). The *idemix* credentials can be used for an unlimited number of proving protocols without transactions becoming linkable.

Despite the differences in the specifications, it is possible to use U-Prove tokens as part of the framework described in this paper. After all, the U-Prove issuing specification is a means to issue a signature on attributes and it is not hard to extend their specification to cover all the features of our specification. The resulting U-Prove Tokens would still be valid U-Prove Tokens. The same holds for the U-Prove subset presentation proof specification, but of course such extended U-Prove tokens could no longer be verified according to the (unmodified) U-Prove subset presentation proof as the extended proof will necessarily contain new elements.

6 Conclusion

We have provided an architecture and specifications of the components, protocols, and data formats of the *idemix* anonymous credential system. The architecture and specification builds the basis to build a large range of applications that require some form of anonymous authentication. We believe that especially our specification language for the various features of the proving protocol is well-suited for making easy use of the different components such as commitment schemes, verifiable encryption, and representations of group elements. That is, with our specification we enable implementation of systems without having an understanding of the cryptography realizing a feature, in fact, we only require knowledge of the very principle. However, this is the minimal understanding that we can require.

We compared our languages to the U-Prove specification and noticed that the more extensive set of features requires a more powerful language. Our language does not manage to hide all this complexity. Still, we hide all the cryptographic complexity (e.g., which groups need to be used or which exponentiation should be computed) while offering access to primitives that proved helpful when designing various privacy friendly systems.

When it comes to established standards we note that the proof specification together with the corresponding (cryptographic) proof values can be seen as the privacy-enhanced equivalent of an X.509 attribute certificate or SAML token: The proof specification defines the attributes that are stated and the proof values correspond to the digital signature on the certificate/token. We could also integrate with X.509 and SAML by using their formats for the specification of the attribute statement and then derive the proof protocol specification from that. The proof specification and the proof values would in this case be the digital signature. This approach would, however, require some changes in the X.509 and SAML specifications. We leave this as future work.

Acknowledgements

We enjoyed numerous discussions about Identity Mixer and its implementation with far too many people to mention all of them here. Thanks to all of you! We are especially grateful to our collaborators at IBM who contributed in numerous ways: Endre Bangerter, Abhilasha Bhargav-Spantzel, Carl Binding, Anthony Bussani, Thomas Gross, Anna Lysyanskaya, Susan Hohenberger, Els van Herreweghen, Thomas S. Heydt-Benjamin, Phil Janson, Markulf Kohlweiss, Sebastian Mödersheim, Gregory Neven, Franz-Stefan Preiss, abhi shelat, Victor Shoup, Dieter Sommer, Claudio Soriente, Michael Waidner, Andreas Wespi, Greg Zaverucha, and Roger Zimmermann.

References

1. E. Bangerter, J. Camenisch, and A. Lysyanskaya. A cryptographic framework for the controlled release of certified data. *SPW '04*, LNCS. 2004.
2. P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous credentials on a standard Java Card. *Proc. 16th ACM CCS*, p.600–610. Nov. 2009.
3. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. *Proc. 11th ACM CCS*, p.168–177. 2004.
4. S. Brands. Electronic cash systems based on the representation problem in groups of prime order. *CRYPTO '93*, p.26.1–26.15, 1993.
5. S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy.* PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.
6. S. Brands, L. Demuynck, and B. D. Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. *ACISP*, v.4586 of *LNCS*, p.400–415. 2007.
7. S. Brands and C. Paquin. U-Prove cryptographic specification v1.0, Mar. 2010.
8. J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. *ASIACRYPT '08*, p.234–252, 2008.
9. J. Camenisch and T. Groß. Efficient attributes for anonymous credentials. *Proc. 15th ACM CCS*, p.345–356. Nov. 2008.
10. J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. *Proc. 13th ACM CCS*, p.201–210. 2006.

11. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-cash. *Eurocrypt 2005*, v.3494 of *LNCS*, p.302–321. 2005.
12. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). *SCN '06*, v.4116 of *LNCS*, p.141–155, 2006.
13. J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. *Public Key Cryptography*, p.481–500, 2009.
14. J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. *EUROCRYPT '01*, v.2045 of *LNCS*, p.93–118. 2001.
15. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. *SCN '02*, v.2576 of *LNCS*, p.268–289. 2003.
16. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. *CRYPTO '04*, v.3152 of *LNCS*, p.56–72. 2004.
17. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. `http://eprint.iacr.org/2002/161`, 2002.
18. J. Camenisch and E. Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. *Proc. 9th ACM CCS*. 2002.
19. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Comm. of the ACM*, 24(2):84–88, Feb. 1981.
20. I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. *ASIACRYPT '02*, v.2501 of *LNCS*. 2002.
21. T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki. Revocable group signature schemes with constant costs for signing and verifying. *Public Key Cryptography*, v.5443 of *LNCS*, p.463–480. 2009.
22. IBM Research–Zurich, Security Team. Specification of the identity mixer cryptographic library. IBM Research Report RZ 3730, IBM Research Division, Apr. 2010.