An extended abstract of this paper appeared in the Proceedings of ACM DIM, Chicago, IL, USA, November 2011.

A Comprehensive Framework Enabling Data-Minimizing Authentication

Patrik Bichsel, Jan Camenisch and Franz-Stefan Preiss
IBM Research – Zurich, Switzerland
{pbi, jca, frp}@zurich.ibm.com

ABSTRACT

Classical authentication mechanisms have various drawbacks such as the weak security properties they achieve, users' privacy, service providers' data quality, and the necessary protection of the collected data. Credential-based authentication is a first step towards overcoming these drawbacks. When used with *anonymous credentials*, the personal data disclosed can be reduced to the minimum w.r.t. a business purpose while improving the assurance of the communicated data. However, this privacy-preserving combination of technologies is not used today. One reason for this lack of adoption is that a comprehensive framework for privacy-enhancing credential-based authentication is not available. In this paper we review the different components of such an authentication framework and show that one remaining missing piece is a translation between high-level authentication policies and the cryptographic token specification level. We close this gap by (1) proposing an adequate claim language specifying which certified data a user wants to reveal to satisfy a policy and by (2) providing translation algorithms for generating the anonymous credentials (cryptographic tokens) providing the data to be revealed. For the latter we consider the Identity Mixer and the U-Prove technologies, where we provide detailed translation instructions for the former.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls, Authentication, Cryptographic Controls*;
K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Languages, Security

Keywords

Access Control, Policy Languages, Privacy, Anonymous Credentials, Digital Credentials.

DIM'11, October 21, 2011, Chicago, Illinois, USA.

1. INTRODUCTION

Authentication has become an all-embracing requirement in electronic communication. Virtually any online service, from music streaming platforms to online bookstores, requires its users to log in or at least provides added value for registered users. The prevailing method to authenticate is by username and password, a simple and cheap solution most users are familiar with. At account establishment time, users often have to provide an extensive set of personal information. This erodes the users' privacy and opens the door for criminals misusing the data, e.g., for identity theft.

Authentication based on anonymous credentials (proposed by Chaum [11] and implemented by Brands [5] or Camenisch and Lysyanskaya [8]) can overcome these security issues by providing strong authentication, minimizing the personal data required for a transaction, and ensuring correctness of data revealed, all at the same time. Unfortunately, this technology is not deployed today as it is hard to understand and complex to use.

In this paper we aim at removing the barriers for using privacy-friendly authentication. To this end, we review the different pieces of a framework for credential-based authentication. This framework consists of (1) a policy language that allows service providers to express which data about a user they require and which authorities they trust in vouching for such data, (2) mechanisms to generate a specification about how a user wants to satisfy the policy, (3) means to generate evidence supporting this specification, (4) mechanisms for the service provider to verify whether the evidence corresponds to the original policy, and (5) means to verify the evidence itself.

As a concrete instance of the first component, i.e., a policy language, we use the credential-based authentication requirements language (CARL) as proposed in [9]. It abstracts the cryptographic aspects of anonymous credentials into well-known authentication concepts. We selected this language as it provides the most comprehensive support for privacy-preserving features. The third and fifth components are provided by several credential technologies, e.g., Identity Mixer (idemix), U-Prove, or X.509 that provide token generation and verification. However, so far no efforts have been made to provide the components (2) and (4) and thereby to close the gap between high-level authentication languages and low-level technology-dependent specification languages responsible for the token generation and verification algorithms.

We solve this problem by (a) proposing a high-level claim language, (b) showing how this language can be translated into two specific specification languages of anonymous credentials, namely, the idemix proof specification and the U-Prove token specification, and (c) showing how the service provider can verify the different pieces it received against the policy it had sent. For translating the claim specification into the idemix proof specification, we show

how the different cryptographic building blocks need to be orchestrated the generate tokens realizing the different claim elements. Finally, we discuss how to extend idemix and U-Prove so that all concepts in the claim language (except disjunction) can be realized. We believe that connecting the high-level languages to the specific technologies is a major step towards enabling data-minimizing credential-based authentication and will foster deployment of the technology.

Related Work.

While currently no other authentication solution provides the comprehensive set of privacy features offered by our framework, the Security Assertion Markup Language (SAML) and WS-Trust as standards for exchanging certified information must be mentioned.

SAML enables a party to send certified attribute information to a recipient. Such attribute information is accumulated within so-called *assertions*, which are similar to what we call credentials. Depending on the underlying certification technology, attributes may be disclosed selectively, however, there is neither support for attribute predicates nor for concepts such as attribute disclosure to third parties. Therefore, without extensions, SAML is not suitable for being used as claim language in data-minimizing authentication scenarios. Ardagna et al. [1] give a brief intuition on how SAML may be extended with those missing features. This extended version of SAML is an alternative to our proposed claim language, however, our language makes deriving claims from CARL policies much easier as it is based on CARL. Further, our language provides a clear grammar that can directly be used for implementing the language. WS-Trust defines protocols to issue, renew and cancel WS-Security tokens. However, WS-Trust is agnostic w.r.t. the type of token. Users obtain tokens from so called security token services (STS) and present those to web services. Web services publish their security policy by means of the WS-Policy standard. WS-Policy, however, merely standardizes an empty container that needs to be filled by concrete policy languages such as CARL. Thus, while we may use WS-Trust or WS-Policy in a data-minimizing authentication framework, they do not close the existing gap between the different levels of languages we currently have.

2. PRELIMINARIES

Anonymous credentials are an important ingredient of privacy-friendly authentication. Therefore we provide a brief overview about the concepts and technologies that implement such systems, borrowing notation from Camenisch et al. [9].

We consider a *credential* to be a set of attributes together with the values of a specific entity, which we call the *owner* of the credential. Each credential has a type that defines the set of attributes the credential contains. As an example, a credential of type 'passport' would contain the attributes *name*, *address*, and *date of birth*. Further, each credential has an entity, the so-called *issuer*, that vouches for the attribute values. As the certified values identify the owner, we also denote the issuer as *identity provider* (IdP). The reputation of an issuer as well as the issuance process (e.g., with physical presence or not) influence the trustworthiness of a credential.

Credentials can be issued using various technologies such as anonymous credential systems [5, 8], X.509 [12], OpenID [15], SAML [14], or LDAP [19]. Identity providers can vouch for users directly or by means of certification. That is, the issuer either communicates the credential directly to the relying party (i.e., service provider) or it provides the user with a certified credential she can then show to the relying party. We call these two approaches *online* and *certified* credentials, respectively, and discuss them in the remainder of this section.

2.1 On-Line Credentials

In the case of on-line credentials, the issuer retains the user's attribute values and when a user wants to use a credential, the relying party and the issuer interact directly. We call such credentials 'online' as the issuer needs to be online for each transaction of a user.

Let us illustrate how on-line credentials work on the example of OpenID. An OpenID provider, which may be seen as identity provider, stores the user's attribute values, e.g., in an database. If the user wants to release any of her attribute values, she relates the relying party to her issuer, i.e., her OpenID provider. The latter provides the attributes to the relying party by using a secure channel to transfer the information. Based on the trust of the relying party in the OpenID provider as well as the security provided by the communication channel, it derives the assurance about the communicated attribute values. Note that this information flow does not require certified information to be transferred.

2.2 Certified Credentials

Credential technologies such as X.509 or anonymous credentials use a different approach. They add a certification value to the credential, i.e., some form of a digital *signature*. This value allows a user to prove that the issuer vouches for her credential without involving the issuer into the communication with the relying party. From a privacy perspective, this is an important advantage over online credentials as the issuer does not get involved into any transaction of a user. As mentioned before, our main interest lies in credential technologies that support even more privacy-preserving features compared to standard certification technology such as X.509.

Anonymous credential system implementations, more specifically, idemix [18] or U-Prove [16] offer such additional features. In essence, they allow a user to obtain a signature from an issuer on a number of attributes similar to standard certification technology. The difference in the issuing process being that the issuer does not get to know the credential the user obtains nor does it learn the attributes that it certifies. This is possible as anonymous credentials use a *blind issuance* process.

After a user has obtained a credential she can release the certified attributes to a relying party. In contrast to other certified credentials where all attributes need to be shown for verifying the signature, anonymous credentials enable a user to only release a subset of the attributes where the signature of the issuer may still be verified by the relying party. This feature is called *selective attribute disclosure*. Another advantage of anonymous credentials lies in the fact that properties about attributes can be proven without revealing the attributes themselves. For example, using an anonymous credential containing a user's date of birth, she can prove the certified statement that she is older than 21 (provided this is indeed the case) without revealing the exact date itself.

3. DATA-MINIMIZING AUTHENTICATION

In this section we discuss the different components of credential-based authentication systems [9] and classify them into already existing and missing components. More concretely, we discuss the existing ones also in this section and provide the missing components in the remainder of this paper.

Figure 1 depicts the components of a data-minimizing authentication system and the sequence of an authentication transaction. Users own certified credentials (in [9] called "cards") that were previously issued to them from identity providers. The figure depicts how a user wants to use a service (e.g., a teenage chat room) hosted by some server. For using their service, the server requires the user to authenticate w.r.t. service-specific authentication policy. An important aspect of data-minimizing authentication is that the policy

is formulated in terms of properties of the user's credentials. For example, a policy could specify that only users who are teenagers according to a national ID card may use the service.

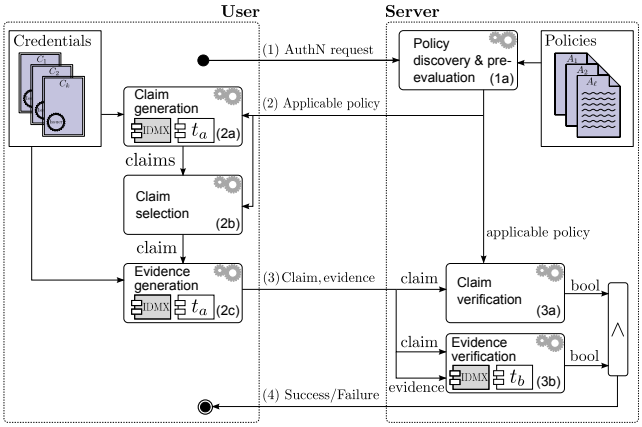

Figure 1: Data-Minimizing Authentication

Upon receiving an authentication request (1) for a service, a server determines and pre-evaluates the applicable policy (1a) and sends it to the user (2). During this pre-evaluation, references to static content such as the current date are resolved to generate the policy sent. Having received the policy, the user's system determines which *claims*, i.e., statements about a subset of attributes of one or more of the available credentials, can be made that fulfill the given policy (2a). For example, a policy requiring the user to be a teenager according to an ID card may be fulfilled by means of a user's national ID card or her student ID. Thereby, the statement of being a teenager can be made by disclosing the exact date of birth or by a (cryptographic) proof that the birth date lies within the required range. Indeed, the claims that a user can make depend on the capabilities of the underlying credential technology. The favored claims are then selected (2b) interactively by the user [3] or automatically by a heuristics capable of finding the most privacy-preserving one. Once the claims are defined, the specific credential technologies have to be instructed to generate the necessary credentials (or tokens) that satisfy these claims. To this end, a technology-specific *proof specification* (e.g., an idemix proof specification or a U-Prove token specification) must be generated. Based on this specification, technology-specific *evidence* is generated (2c). The claim is then sent together with the accompanying evidence to the server (3) who verifies that the claim implies the policy (3a) and checks whether the claim's evidence is valid (3b). Depending on the credential technology, the evidence may be generated and verified with or without the credential issuer being involved. After successful verification, the user is authenticated (4) as someone fulfilling the authentication requirements dictated in the policy. The strength of anonymous credential systems lies in the fact that the server does not learn more than what it strictly requested. For example, the only information the server learns about the user is the fact the she or he is indeed a teenager according to an ID card issued by a trusted identity provider. Thus, the user has minimized the information revealed about herself w.r.t. the given authentication policy. Ideally, the policy also reflects the minimal information necessary for conducting the scenario at hand.

For implementing such authentication scenario, at least three types of languages are required. First, a policy language to express the server's authentication requirements. Second, a claim language to make statements about (attributes of) the user's credentials, and third, a technology-specific language that defines how the evidence is generated. Camenisch et al. [9] provide a suitable policy language with their *credential-based authentication requirements language* (CARL, cf. Section 3.1). The currently existing idemix proof specification [2] language is a technology-specific language for generating evidence (cf. Section 3.2). The language specifying claims made by the user, however, is missing. In this paper, we provide this missing piece by defining such language on the basis of CARL. We also show how to extend the currently existing idemix proof specification language to make it as expressive as our claim language.

Given this new claim language, in Section 4 we further describe how technology-specific claims can be generated and verified for a given policy. In Section 5 we explain how evidence can be generated and verified for a claim formulated in the specified claim language with a focus on anonymous credential technologies.

3.1 CARL Policy Language

We briefly introduce the CARL policy language [9] as it is the basis for the claim language that we define in Section 4.2. The language allows for expressing authentication requirements in terms of credentials in a technology-agnostic way. One kind of requirement states a predicate over the credential's attributes. The predicate is a Boolean expression that allows for applying logic, comparison and arithmetic operators to attributes, constants or further expressions. Consider the following example policy for a car rental service that illustrates the language features relevant for us:

```
01: own dl::DriversLicense issued-by DeptMotorVehicles
02: own mc::MemberShipCard issued-by CarRentalCo
03: own cc::CreditCard issued-by Amex, Visa
04: own li::LiabilityInsurance issued-by InsuranceCo
05: reveal cc.number
06: reveal li.pNo to EscrowAgent under 'in case of damage'
07: where dl.issueDate ≤ dateSubtrDuration(today(), P3Y) ∧
08:       li.guaranteedAmoutUSD ≥ 30.000 ∧
09:       (mc.status == 'gold' ∨ mc.status == 'silver') ∧
10:       dl.name == li.name
11: sign 'I agree with the general terms and conditions.'
```

According to this policy, authentic users (a) have their driver's license for at least three years, (b) have *gold* or *silver* membership status with the car rental company, (c) reveal their American Express or Visa credit card number, (d) reveal the policy number of the driver's liability insurance – with a coverage of at least thirty thousand dollars – to a trusted escrow agent who may disclose this number only in case of damage to the car, and (e) consent to the general terms and conditions. Users who fulfill this policy with idemix evidence reveal – aside from their credit card number – merely the facts that (1) the credentials they own do fulfill the stated requirements and (2) they indeed disclosed the proper values to the escrow agent. In particular, the server neither learns the exact values of the attributes occurring in the policy's where clause nor the number of the insurance policy. However, despite the users' preserved privacy, they are accountable in case of damage due to the information the escrow agent learned. Note that identities such as Visa represent aliases (e.g., to cryptographic keys) that are resolved by the credential technology used to fulfill the policy. We refer to Sections 4 and 5 of [9] for more detailed information on syntax and semantics of CARL, respectively.

3.2 Idemix Proof Specification

The idemix anonymous credential system consists of a number of cryptographic building blocks including signature scheme, en-

cryption, and commitment schemes. Combined they can achieve the different features of the idemix authentication system. The components as well as their combination is driven by specification languages that abstract from cryptographic details [2]. Thus, to generate an idemix credential, i.e., the evidence, that fulfills a given claim, we have to translate claims into the idemix proof specification language. We summarize the main features that idemix provides and that can be realized using the idemix proof specification. One major advantage of anonymous credentials over other credential technology is their ability to *disclose attributes selectively*. The language supports this feature by specifying for each attribute of each credential whether it should be revealed or not. A further advantage is that they allow a user to prove that attributes encoded in different credentials fulfill a specified predicate by so called *cross-credential proofs*.

Note that there is a gap between what has been proposed as features for idemix in the scientific community (e.g., limited spending [7]), what is specified and implemented [18], and what can be expressed with the language proposed in [2]. We will highlight what *predicates* can be expressed by the language as this is the most limited set. Those predicates are (1) equality among attributes, (2) inequality between attributes and constants, and (3) set membership proofs. First, equality among attributes can cryptographically be proven by using the same values for both attributes within a zero-knowledge proof. The proof specification achieves this feature by using one so-called *identifier* for several attributes. Second, inequalities allow the user to specify that an attribute is smaller or larger than some constant. In fact, the language supports the operators $<, \leq, \geq$ and $>$ and provides a distinct construct for the specification of the attribute, the constant, and the operator. Third, the language specifies a construct to define attributes that should be used in a set membership proof. Set membership proofs are only available for specially encoded attributes, where the idemix implementation uses prime encoded attributes as proposed by Camenisch and Gross [6].

In addition to the given predicates the library implements concepts such as *disclosure of attributes to a third party*, or *signature on messages*. The former is realized using verifiable encryption as proposed in [10] and the latter amounts to signing the message using the Fiat-Shamir heuristic [13]. As those features are not credential-specific, they are addressed by dedicated statements in the proof specification language. Furthermore, the library implements that idemix proofs can be tied to pseudonyms. However, this concept is currently not reflected in our claim language.

Finally, the proof specification language offers the creation of commitments as well as representations. Both are cryptographic objects that can be employed to implement high level functionality. In this paper we show how those constructs can be used to implement arbitrary arithmetic statements about certified attributes. For example, in Figure 2 we provide the idemix proof specification corresponding to the claim described in Section 4.2. Note that we only indicate the attributes required in the claim.

```
<ProofSpecification [...]>
 <Declaration>
  <AttributeId name="id1" proofMode="unrevealed"
                                        type="string"/>
  <AttributeId name="id2" proofMode="unrevealed"
                                        type="date"  />
  <AttributeId name="id3" proofMode="revealed"
                                        type="string"/>
  <AttributeId name="id4" proofMode="revealed"
                                        type="int"   />
  <AttributeId name="id5" proofMode="unrevealed"
                                        type="int"   />
  <AttributeId name="id6" proofMode="unrevealed"
                                        type="int"   />
 </Declaration>

 <Specification>
  <Credentials>
   <Credential ipk="http://www.DeptMotorV.com/ipk.xml"
     credStruct="http://www.un.org/license/driver.xml"
     name="kdsfjk230fsefj32">
    <Attribute name="name">id1</Attribute>
    <Attribute name="issueDate">id2</Attribute>
    [...]
   </Credential>
   <Credential ipk="http://www.CarRentalCo.com/ipk.xml"
     credStruct="http://www.CarRentalCo.com/memCard.xml"
     name="oiwd26ia3m232ewo">
    <Attribute name="status">id3</Attribute>
    [...]
   </Credential>
   <Credential ipk="http://www.visa.com/ipk.xml"
     credStruct="http://www.imf.org/creditcard.xml"
     name="kdf92fjiu01fj028">
    <Attribute name="number">id4</Attribute>
    [...]
   </Credential>
   <Credential ipk="http://www.InsuranceCo.com/ipk.xml"
     credStruct="http://www.InsucanceCo.com/policy.xml"
     name="028dkd93rdlra039">
    <Attribute name="name">id1</Attribute>
    <Attribute name="pNo">id5</Attribute>
    <Attribute name="guaranteedAmountUSD">id6</Attribute>
    [...]
   </Credential>
  </Credentials>

  <Inequalities>
   <Inequality pk="http://www.DeptMotorV.com/ipk.xml"
     operator="leq" secondArgument="76168">id2
   </Inequality>
   <Inequality pk="http://www.InsuranceCo.com/ipk.xml"
     operator="geq" secondArgument="30000">id6
   </Inequality>
  </Inequalities>

  <VerifiableEncryptions>
   <VerifiableEncryption name="jd2e0asfdkkj3rqq1"
     pk="http://www.EscrowAgent.com/vepk.xml"
     label="in case of damage">id5</VerifiableEncryption>
  </VerifiableEncryptions>

  <Messages>
   <Message name="d0fsdfkii2fucxzkl">I agree with the
              general terms and conditions.</Message>
  </Messages>
 </Specification>
</ProofSpecification>
```

Figure 2: Idemix proof specification that realizes the example claim specified in Section 4.2.

3.3 Privacy Benefits

The choice of anonymous credential systems as credential technology lies in their privacy benefits over competing technologies. We provide an overview of the privacy features and distinguish again between on-line and certified credentials. Depending on the definition of privacy, on-line credentials may have many advantages. That is, if we only care about a user's privacy w.r.t. to the relying parties, they are a feature-rich and privacy-friendly variant. Their main drawback is that the issuer of the credential (i.e., the IdP) is involved in each transaction, i.e., it provides unlinkability when using a credential multiple times *only* w.r.t. the relying parties. In addition, features such as proving predicates that involve credentials issued by different IdPs can only be achieved using special protocols between those IdPs. Table 1 shows that, except for the generation of the evidence independently from the IdP, all privacy features are provided by on-line credentials. Note that the

restrictions on unlinkability and cross-credential proofs are due to the reasons mentioned before.

Certified credentials such as X.509, idemix, or U-Prove credentials provide a significant advantage over on-line credentials. They allow a user to proof possession of the credential without involving the IdP, which provides privacy w.r.t. to the issuer that an on-line credential can never provide. However, when it comes to the privacy w.r.t. to the relying party, certified credentials cause some difficulty. On one hand, protection of the user's privacy demands that the latter can change the credential to make it fit for a given purpose. On the other hand the issuer needs to make sure that only specific changes to the credential can be made, i.e., the semantics of the credential must remain unchanged.

Feature	On-line	X.509	U-Prove	Idemix
Message Signatures	✓	✓	✓	✓
Proof of Ownership	✓	✓	✓	✓
Evidence w/o IdP		✓	✓	✓
Selective Disclosure	✓		✓	✓
Predicate Proofs	✓		$\checkmark_{(+)}$	✓
Disclosure to Third-Parties	✓		$\checkmark_{(+)}$	✓
Limited Spending	✓		$\checkmark_{(+)}$	$\checkmark_{(+)}$
Cross-Credential Proofs	(✓)		$\checkmark_{(+)}$	✓
Unlinkable Multi-Use	(✓)			✓

Table 1: Certification technology feature comparison. ✓: Supported. (✓): Limited support. $\checkmark_{(+)}$: Possible and described in the literature, but currently not implemented.

Table 1 shows that X.509 credentials only support basic signatures and ownership proofs. As mentioned before they do allow to generate this proof (or evidence) without involving the IdP. The reason for providing such limited set of functionality lies in the fact that a user cannot adapt the cryptographic signature.

Implementations of anonymous credentials, such as idemix or U-Prove, provide more flexibility in terms of privacy protection. Both allow a user to selectively reveal attributes that have been certified in a credential (cf. *Selective Disclosure* in Table 1). U-Prove strives for simplicity, thus, it currently does not provide further privacy protecting features even though the underlying signature scheme would support further features. The idemix library[1] is currently the most advanced implementation of a privacy-preserving authentication system. In addition to all previously mentioned features it supports proofs of predicates over attributes, e.g., proving equality among attributes, and other features described in Section 3.2. Using verifiable encryption, it allows for conditionally disclosing attributes to a (trusted) third party. Limiting the possibilities of spending a credentials such as only allowing a credential to be used only k times within a certain time interval as proposed in [7] is currently not implemented but it could be added to the current implementation. The distinction between idemix and U-Prove boils down to the fact that an individual idemix credential can be used multiple times without the resulting evidence becoming linkable. This is referred to as *Unlinkable Multi-Use* in Table 1.

4. CLAIM HANDLING

We compare in Section 4.1 different ways on how an authentication policy can be fulfilled . The decision on which way is chosen is mainly driven by the capabilities of the underlying credential technology. Therefore, those capabilities have to be known and considered at the time of claim generation. No matter which way the policy is fulfilled, however, an adequate claim language is needed to express the statements made about the credential's attributes.

In general, claims may be accompanied with evidence from different credential technologies. However, requirements across different credentials, so called *cross-credential requirements* cannot be proven using different credential technologies. Although the claim language itself is independent from technology, the expressed statement must be in accordance with the capabilities of the underlying credential technology. In this section we describe how to generate claims for the anonymous credential technologies idemix and U-Prove. Naturally, of interest are only claims that logically imply the policy, therefore we define claim semantics in Section 4.2.

4.1 Methods To Fulfill A Policy

An authentication policy can be fulfilled in several ways. Intuitively, in case a policy requires the user to show that she owns a driver's license, we can see that the user can comply by providing a proof of such statement or by simply revealing the driver's license information as we do today. On a more conceptual level we can distinguish three methods for complying with a given authentication policy. First, using non-certified credentials a user can request a claim that closely matches the given policy. Second, using a standard certified credentials as introduced in Section 2 allows a user to generate a claim without involvement of the IdP. However, this technology lacks the ability to adapt claims to a given policy. Third, privacy-preserving certified credentials such as anonymous credentials enable a user to generate a claim specific to a given policy. The privacy implications on each of those options are discussed in more detail in Section 3.3. For all three methods we need a claim language to describe what the content or structure of generated evidence is. Therefore, we need a claim language and define is now.

4.2 Claim Language

Analog to servers who express their authentication requirements in a policy language, users make authentication statements in a claim language. A claim precisely describes the statements that a user proves by means of technology-dependent evidence. In particular, claims serve as technology-independent input to technology-specific evidence generation modules. Although such claim language is a crucial building block for data-minimizing authentication systems, no adequate claim language has been proposed yet.

The claim language we propose allows a user to state which credentials she owns and what properties those credentials have. Such properties are expressed in terms of a logical predicate over the credential's attributes. Additionally, the language allows users to consent to a certain message or to disclose attributes to a (trusted) third party. The language we propose is intended as counterpart to the CARL policy language (cf. Section 3.1). In fact, most language constructs can be reused, however, three concepts need special attention.

First, for credential ownership (i.e., 'own' lines) CARL policies allow for specifying a list of issuers with disjunctive semantics, i.e., ownership can be proven for any of those issuers. As it must be unambiguous for the underlying credential technology which cryptographic key to use for generating the claim's evidence, the claim language just states one issuer. Second, disclosure requests for attributes that are to reveal to the server (i.e., 'reveal' lines without 'to') are only meaningful in policies. A claim must rather fulfill those requests by disclosing the corresponding attribute values. In our language, such attribute disclosure is addressed by stating equality between the respective attribute and its value. Third, CARL supports basic variables that may act as substitute for a num-

[1] http://www.zurich.ibm.com/~pbi/identityMixer_gettingStarted/

ber of syntax elements. While being useful in policies, such variables provide no benefit to a claim language. Therefore, the only kind of variable we consider are attribute variables which reference credential attributes.

Figure 3 shows the (left factored) grammar of our claim language. Apart from above mentioned restriction, credential ownership is expressed with 'i own' lines in the same way as 'own' lines in CARL. We prefix the main keywords with 'i' to stress the claim's active statement character. The attribute predicate is expressed after the 'where' keyword in terms of a Boolean expression. Beside the standard operators of *logics* ($\wedge$, $\vee$ and $\neg$), also *equality*, *inequality* ($\neq$, $>$, etc.) and *arithmetic* operators may be applied to expressions. Expressions may further be (1) attributes qualified with the ID of a previously declared credential (e.g., $dl.issueDate$), (2) constants of data type String, Boolean, Date (e.g., 1984/01/01), Float and Duration (e.g., P3Y represents a period of three years), as well as (3) function calls with expressions as arguments. A type system equivalent to the one of CARL [9, Appendix C] ensures that the predicates are type correct w.r.t. the data types defined in the credentials' types and the function definitions. The message to sign is given after the 'i sign' keyword. It must be a constant expression that evaluates to data type string. To disclose a list of terms to a third party, the 'i reveal' keyword is used. Although CARL also provides syntax to address limited credential spending, we do not consider this concept here. Consider the following example claim:

```
01: i own dl::DriversLicense issued-by DEPTMOTORVEHICLES
02: i own mc::MemberShipCard issued-by CARRENTALCO
03: i own cc::CreditCard issued-by VISA
04: i own li::LiabilityInsurance issued-by INSURANCECO
05: where dl.issueDate ≤ dateSubtrDuration(today(), P3Y) ∧
06:       li.guaranteedAmoutUSD ≥ 30.000 ∧
07:       mc.status == 'silver' ∧ dl.name == li.name ∧
08:       cc.number == '1234 5678 9012 3456'
09: i reveal li.pNo to ESCROWAGENT under 'in case of damage'
10: i sign 'I agree with the general terms and conditions.'
```

This claim is one possible counterpart to the policy given in Section 3.1. Its intent is to fulfill the choices given in the policy with a visa credit card and a membership card of silver status. Additionally, a concrete credit card number is revealed. The functions and their interpretations are specified in an ontology that is commonly agreed upon by the user and the server.

Note that the example claim reveals that the membership status is silver, rather than just saying that it is silver or gold. In general, the latter would be preferable, however, current implementations do not yet allow to prove disjunctive statements (cf. Section 5). Further note that the user may be involved in the selection of the most desirable claim in case (1) multiple credentials are possible to use according to policy (e.g., multiple credit cards), or (2) multiple claims result from one assertion, e.g., because a credential technology does not support disjunctive statements.

4.3 Claim Generation

When a user u receives the policy applicable to her authentication request (cf. Step 2 in Figure 1), is has to be determined which claims can be made w.r.t. her credential portfolio $\mathfrak{P}_u$. To do so, the possible options of assigning credentials from $\mathfrak{P}_u$ to all of the credential variables occurring in the policy are calculated. We call one such option a *credential assignment*. An important aspect of the claim generation component is to determine *all* possible credential assignments of a user for a given policy. This is to enable the user to select the most privacy-preserving assignment. In case

```
Claim    = PrfOfOs+ ['where' Exp] Discl* ['i sign' Exp] ;
PrfOfOs  = 'i own' CredVar '::' URI 'issued-by' URI ;
Discl    = 'i reveal' Term (', ' Term)* 'to' URI ;
CredVar  = ID ;
AttrVar  = CredVar '.' ID ;
Term     = AttrVar
         | String | Float | Date | Bool | Duration ;
Exp      = DisjExp ;
DisjExp  = ConjExp ('∨' ConjExp)* ;
ConjExp  = EquExp ('∧' EquExp)* ;
EquExp   = InEquExp ('==' InEquExp)* ;
InEquExp = AddExp (('≠' | '<' | '>' | '≤' | '≥') AddExp)*;
AddExp   = MultExp (('+' | '−') MultExp)* ;
MultExp  = NegExp (('·' | '÷') NegExp)* ;
NegExp   = ['¬'] SigExp ;
SigExp   = ['+' | '−'] PrimExp ;
PrimExp  = '(' Exp ')'
         | Term
         | ID '(' [Exp (', ' Exp)*] ')' ;
ID       = Alpha Alphanum* ;
```

Alpha and Alphanum are alphabetic and alphanumeric characters. The URI after the 'issued-by' keyword must map to a identifier that the underlying credential technology can resolve. The URI ater the '::' keyword must map to a credential type. IDs must be different from the used keywords.

Figure 3: Claim Language Grammar

no credential assignments are found, the user's credentials are not sufficient to fulfill the policy.

Every credential assignment found by above mentioned algorithm is then transformed into a claim. The transformation, however, is dependent on the technologies of the assignment's credentials. This is due to the varying capabilities the different credential technologies have (cf. Section 3.3). In case all credentials in the assignment are of the same technology, the assignment is transformed to a technology-specific claim according to the idemix and U-Prove restrictions given in the following subsections, respectively. In general it is possible to also support the case where the assignment's credentials are of different technologies. However, in this work this is out of scope.

4.3.1 Idemix Claim Restrictions

To generate the claims for which evidence shall be generated with idemix, one has to account for its capabilities. The current implementation of idemix supports most of the possible claim statements. With the extension to the proof specification we discuss in this paper, we are able to support all statements except those that contain disjunctive expressions (cf. Tables 1 and 2). Thus, when generating a claim for which evidence with idemix shall be generated, one has to proceed as follows. The policy's predicate is first transformed to disjunctive normal form (DNF) and separate claims are generated for all monomials (also called conjunctive clauses, or conjunctions of literals) of the DNF that (a) hold with respect to the given credential assignment, and (b) resemble the given policy (apart from the predicate being only the monomial, not the full predicate). It can assumed that at least one of those monomials holds, otherwise the assignment finder component would not have produced this assignment. Clearly, using a monomial of the predi-

cate's DNF as predicate in the claim is less privacy-preserving than stating the full predicate as it is disclosed which disjunct is proven. For creating a claim that resembles a policy, put simply, a 'copy' of the policy is used as claim and the constraints on the claim language as defined in Section 4.2 are enforced. According to those constraints, every credential declaration must have exactly one issuer. This issuer is unambiguously defined as the issuer of the credential that is assigned to the corresponding credential variable via the credential assignment. Further, all attributes, for which a disclosure request exists in the policy, are disclosed by adding an equality expression between the corresponding attribute and its value as additional conjunct to the monomial.

4.3.2 U-Prove Claim Restrictions

The latest specification of U-Prove [16] allows for selective disclosure of attributes, signing messages, and proof of ownership, but does not support features such as predicate proofs and disclosure to third parties (cf. Tables 1 and 2). A policy that includes predicates over attributes or disclosure to third parties, can be fulfilled if all these attributes are fully disclosed to the relying party. To this end, one needs to process the policy's predicate and transform it into a claim predicate in which all attributes occurring in the predicate are selectively disclosed. Claims including cross-credential statements and limited spending cannot be fulfilled with the current specification of U-Prove.

4.4 Claim Verification

A server receiving a claim with accompanying evidence from a user verifies whether this claim indeed implies the initially provided policy (cf. Step 3a in Figure 1). A claim implies a policy if all of the following five conditions hold.

(1) For each disclosure request in the policy there is a corresponding attribute – with the same credential variable and the same attribute variable – disclosed in the claim. (2) The predicate of the policy implies the predicate of the claim's statement. To account for technologies that fulfill the policy's predicate by disclosing all attributes occurring in it (e.g., U-Prove), all the attributes of the policy's predicate that are disclosed in the claim are substituted with the revealed values. Then, if the resulting predicate is constant, it is verified whether it evaluates to true. If so, the predicate is fulfilled. Otherwise the claim does not imply the policy. (3) The credential declarations of the claim's statement imply those of the policy. A claim's credential declaration implies a policy's declaration if (a) their credential variables are equal, and (b) their credential types are equal (for hierarchical credential types, this might be extended to checking whether the claim's credential type is a subtype of the policy's credential type), and (c) the issuer of the claim's declaration is contained in the list of issuers of the policy's declaration. (4) In case the policy requires the signature of a message m, the claim must also contain an 'i sign' statement for m. (5) The set of terms disclosed to third party S_1 in the claim must be a superset of the set of terms that is required to be disclosed to S_1 in the policy.

5. EVIDENCE HANDLING

In this section we show how idemix and U-Prove evidence is generated and verified for a given claim. In particular, this section elaborates on the components (2c) and (3b) of Figure 1. Note that in the evidence verification we only handle claims that have previously been generated (cf. Section 4.3) and we assume that claims adhere to the restrictions of the respective technology.

For transforming the claim to semantically equivalent evidence, we break our claim language syntax down to a set of building blocks. We therefore only need to show how evidence is generated for those building blocks.

5.1 Claim Building Blocks

In Table 2 we show the building blocks of our claim language and detail which are supported by the current implementations of idemix and U-Prove. For idemix, we distinguish between existing support and support introduced through extensions we propose in Section 6. Of particular interest are the building blocks for attribute predicates. We show how every attribute predicate can be rewritten in terms of building blocks. In particular, any attribute predicate with arbitrary logical nesting and negations can be transformed to disjunctive normal form (DNF), i.e., $\bigvee_i \bigwedge_j \ell_{ij}$, where ℓ_{ij} is an atomic expression (AtomicExp in Table 2) with no further logical structure. In DNF negations occur only immediately before atomic expressions. Such negations are eliminated by inverting the respective operators (e.g, $\neg(a < b)$ maps to $a \geq b$). Further, expressions with no further logical structure that do not match any of the building blocks 6 – 15 can be rewritten to do so. For example, the expression $(a.b + c.d) \leq e.f$ can be rewritten to $(a.b + c.d) - e.f \leq 0$ which is building block 14. Atomic expressions that are constant cannot be proven using any credential technology but they can be trivially evaluated.

Note that we distinguish three cases for equality, not equal to, and inequality although those could further be reduced. For example, blocks 6 and 7 are instances of 8. The reason being that typically simpler cases can be implemented more efficiently and are already present, while the most general case may not be implemented. For instance, block 6 and 7 are currently implemented in idemix, but block 8 is not.

5.2 Idemix Evidence

To generate evidence for a claim with the idemix technology, we assume it only contains expressions that are supported in the implementation extended with our proposals as described in Section 6. In particular, for idemix this means that a claim must not contain disjunctive expressions, while all other constructs are supported. As the current idemix implementation does not support proving of disjunctive expressions, the corresponding idemix claim generator (cf. Section 4.3.1) only produces claims that have monomials (i.e., conjunctions of Boolean literals) as their statement's formula (or the formula is null). Thus, we assume that the statement of the given claim is a monomial and determine the individual Boolean literals of the statement's formula. In case the formula is null, the list of Boolean literals is empty. Every literal of the monomial is an instance of one of the expression patterns described in the following paragraphs, which describe how a semantically equivalent proof specification can be created.

5.2.1 Generating Idemix Evidence

This section we describe how to generate idemix-specific evidence. We show an overview of the supported features, including the ones that require extension to the idemix proof specification, in Table 2.

Proof of Ownership (incl. Selective Disclosure).

A proof of ownership is the basic feature of certified credentials. The idemix proof specification provides this functionality for each credential included in the proof specification using the `Credential` tag. This tag also allows for referring to the credential in the rest of a proof specification.

Each attribute of any credential statement must defined a corresponding identifier using the `AttributeId` tag. The attribute identifiers have an explicit possibility to define whether a linked attribute should be revealed or remain unrevealed, which we can use to match the definition in the claim. In addition, they are used to

#	Type		Claim Syntax	Examples	Idemix	U-Prove
1	Proof of Ownership		i own $c::\tau$ issued-by I	i own $c::CreditCard$ issued-by VISA	✓	✓
2	Message Signatures		i sign m	i sign 'terms and conditions'	✓	✓
3	Disclosure To Third Parties		i reveal $c.a$ to TTP under ℓ	i reveal $li.pNo$ to ECRAGT under 'damage'	✓	
	Attribute Predicate		where ϕ	*See rows 4 - 15*		
4	Logics		AtomicExp $\wedge$ AtomicExp	$(a.b == c.d) \wedge (e.f < 1984/01/01)$	✓	
5			AtomicExp $\vee$ AtomicExp	$(a.b == c.d) \vee (e.f < 1984/01/01)$		
6	Equality (incl. Selective Disclosure)		$a.b == c.d$		✓	
7			$a.b ==$ ConstExp	$a.b ==$ 'Chicago'; $a.b == 1984/01/01$	✓	✓
8			NonConstExp $==$ ConstExp	$(a.b + 2 \cdot c.d) == 7$; $log(a.b) == 7$	✓+	
9	Not Equal To		$a.b \neq c.d$		✓+	
10			$a.b \neq$ ConstExp	$a.b \neq 7$; $a.b \neq$ 'male'; $a.b \neq 1984/01/01$	✓+	
11			NonConstExp $\neq$ ConstExp	$(a.b + 2 \cdot c.d) \neq 7$	✓+	
12	InEquality		$a.b$ Op $c.d$	$a.b < c.d$; $a.b \geq c.d$	✓	
13			$a.b$ Op ConstExp	$a.b > 8$, $a.b > 1984/01/01$	✓	
14			NonConstExp Op ConstExp	$(a.b - 2 \cdot c.d) > 25$	✓+	
15	FunctionCall		f(NonConstExp, ...)	$charAt(2, a.b)$	Func.-Dep.	Func.-Dep.

Table 2: Claim Building Blocks. ✓: Supported in current implementation. ✓+: Supported with the extensions described in Section 6. AtomicExp: Any of the building blocks 6 – 15. ConstExp: Expression not containing attribute variables. NonConstExp: Expression containing at least one attribute variable.

define equalities among attributes, which is possible even if they are encoded into different credentials. Thus, we have to use the same attribute identifier for all attributes that the claim states to be equal, where attribute equality can only be proven if the types of the attributes match.

Message Signatures.

A policy may request a user to sign a message, which will be reflected in a claim according to the example in Section 4.2. The idemix proof specification provides a dedicated `Message` element, which allows for a direct translation from the claim.

Attribute Disclosure to Third Parties.

Conditional attribute disclosure to third parties is also included into the proof specification as a distinct primitive. Consequently, it can be almost employed as directly as message signatures. The essential difference is that a verifiable encryption requires the user to specify the pubic key of the trusted entity, which she has to (authentically) retrieve before being able to create the encryption. Furthermore, she needs to add the condition under which the decryption is released to the `VerifiableEncryption` element.

Note that to attain a transaction binding, i.e., the binding among all attributes that are verifiably encrypted to one another, the verifiable encryption needs to contain all the transaction relevant attributes. This is of importance to make sure that none of the parties in the accountability transaction can change the context, i.e., misuse the verifiable encryption. While the claim language allows for disclosing a list of terms, i.e., attribute references or constants, the proof specification does not currently do so. Thus, for every attribute reference we create a separate verifiable encryption. Disclosure of constants is currently not supported by the proof specification.

Attribute Predicates.

Predicates over attributes range from simple expressions that can be directly achieved using a dedicated element in the proof specification to almost arbitrarily complex statements, which need extensions to the proof specification language to be expressable. We explain the transformations for the cases mentioned in Table 2 in Section 5.2.2. Note that attribute predicate proofs over revealed attributes are not supported. Therefore, all the predicate's attribute variables whose values are revealed have to be replaced with their disclosed values.

5.2.2 Generating Attribute Predicate Evidence

In the following, we introduce the mappings of a claim's attribute predicate to an idemix proof specification. We refer to Section 6 for further details.

Equality.

Details of equality predicates in a claim are given on Line 6-8 in Table 2. As already mentioned, the proof specification expresses equality among attributes (Line 6) by using the same attribute identifier. Proving equality between an attribute and a given constant (Line 7) amounts to revealing the attribute value. Note that the verifier needs to check whether the value stated in the claim actually corresponds to the revealed value. Finally, equality statements involving arithmetic expressions have to be mapped to commitments and representations. We show a concrete example of the equality $a.b \cdot c.d == e.f$ in Section 6.2.

Not Equal To.

Statements that express that one value is unequal to another one, are generically hard to proof as technology like idemix is built for proving equality of elements. Still, we can prove a statement $a.b - c.d \neq 0$ rather than $a.b \neq c.d$. While maintaining semantic equivalence we manage to change the statement such that the verifier can check it. This results as we handle the attributes as exponents and the verifier can check that the resulting exponent not being equal to zero.

Using this rational we can transform the statements of Line 9 and 10 into statements that are not equal to zero, i.e., $a.b - c.d \neq 0, a.b - ConstExp \neq 0$. Consequently, using commitments and representations in the generalized form as we explain in Section 6.2, we can express "Not Equal To" statements.

InEquality.

The inequality operators $<, \leq, \geq$, and $>$ are implemented using Boudot [4] interval proofs. This concept profits from a dedicated

element called `Inequality` in the idemix proof specification. A limitation w.r.t. the claim language is that only integers and dates (which are encoded into integers) are supported.

In addition, the implementation only allows for unrevealed attributes to be compared with (1) constants, or (2) revealed attributes. For instance, it does not allow to express a formula of the form indicated in Line 12 in Table 2, i.e., $a.b < c.d$. Such expression where both attributes are unrevealed, needs to be transformed into $a.b - c.d < 0$. We use commitments to each of the attributes to build a representation that contains the subtraction of the attributes, i.e., $a.b - c.d$. Using this value we can proceed and prove an inequality statement as if the value $a.b - c.d$ were a regular attribute value directly certified by a credential.

Non-constant Expressions.

In the Lines 8, 11, and 14 of Table 2, the non-constant expressions (NonConstExp) may either be an *arithmetic expression* or a *function call*. We address the former by generating commitments as well as representations such that the desired value, e.g., $a.b - 2c.d$, is available as if it were a regular attribute in a credential. The example in Section 6.2 provides an intuition on how the commitments and representations have to be generated.

Support for function calls heavily depends on the concrete function. Subsequently, each function would require a dedicated mapping and possibly even special algorithms for the functionality to be supported. We envision special functions only to become available once a convincing use case for a particular function is available.

5.2.3 Verifying Idemix Evidence

Once the idemix evidence has been generated and transmitted to the server, the latter needs to translate the user-provided claim into a proof specification the same way the user did. As a result it will get an idemix proof specification that, together with the evidence itself, serves as input to the idemix library (cf. Figure 1). The first step of the verification consists of the idemix library verifying the cryptographic properties of the evidence. The second step consists of the verification of the disclosed attributes, which have to be matched with the constants used in the claim. A particularity of the idemix implementation lies in the fact that strings currently are encoded by employing a hash function. Thus, the disclosed attributes can, in case of strings, not be mapped to their original value, thus, they have to be transmitted from the user to the verifier. Still the verifier needs to assert that the transmitted values match the values revealed in the evidence.

5.3 U-Prove Evidence

To fulfill a claim with U-Prove, our claim language needs to be translated into a U-Prove token as specified in the U-Prove WS-Trust profile [17]. This profile defines which attributes are revealed (in WS-Trust attributes are called claims). Thus, to generate U-Prove evidence in our system, we would need to translate our claim into a set of U-Prove WS-Token specifications, one for each credential (U-Prove token) that shall be used. These specifications then define the attributes that are revealed. Finally, the different U-Prove tokens generated according to these specifications are assembled to build the final evidence.

6. IDEMIX PROOF-SPEC EXTENSIONS

Table 2 shows basic expression patterns that are theoretically supported by the idemix library but cannot be expressed using the current proof specification language. As we only need to slightly change the languages proposed in [2] in order to provide a substantial improvement to the overall system, we describe those changes here. We provide an intuition on how to extend the idemix proof specification language such that the concepts marked with a $\checkmark_+$in Table 2 are supported.

6.1 Generalized Issuance Process

The design of the proof specification considers a limited issuance scenario, namely, it does not consider that a credential structure is defined by an entity different from an issuer. As we presume that multi-national organizations will specify the format of widely used credentials, we need to specify credential structures independently from issuer-related values. We attain this independence by removing the issuer public key from the credential structure and adding it to each credential in a proof specification. The rational being that a credential structure is independent from an issuer and only a credential, i.e., the instantiation of a credential structure, is linked to the issuer.

6.2 Generalized Representations

Considering the definition of representations in [2] we require a set of extensions. More specifically we require that a representation may (1) refer to other elements (e.g., commitments or representations) as its bases, (2) use constant exponents, and (3) re-use an already defined representation object. We use the first and second property to recursively build elements from arithmetic formulas involving certified attributes as referred to in Section 5.2.2. The last property is needed to establish an equality relation among different representations, which can be used to establish the equality of formulas.

For instance, assume that we need to prove that one attribute is the product of two other attributes, i.e., $a.b \cdot c.d = e.f$ (cf. Line 8 of Table 2). To realize this, we need to generate a commitment to each of these attributes and then prove that the commitment to the third attribute is equal to the commitments to the second attribute, raised to power of the value of the first attribute, times the group element used to randomize commitments raised to power of some integer (the value of which is not of relevance). Similar to this example we can implement more elaborate arithmetic expression by translating them into commitments and representations.

6.3 Relation between U-Prove and Idemix

The signature schemes that underlie U-Prove and idemix are similar. That is, they are both schemes that allow an issuer to (blindly) sign messages where the messages are algebraically encoded as exponents of a representation of an element of an algebraic group. The selective disclosure of attributes is in both cases realized by revealing some exponents (messages) and using a zero-knowledge proof of knowledge of the undisclosed attributes. A zero-knowledge proof can, as the name suggests, convince a verifier of the fact that the prover holds some values without communicating any other information. The difference between the two schemes is that they are based on different cryptographic assumptions and that, due to its cryptographic properties, a U-Prove signature can only be used once to in a proof (otherwise, the proof is no longer zero-knowledge and transactions become linkable).

The advanced features that idemix provides are all realized with cryptography that uses discrete-logarithms mechanisms. Therefore, they can in principle also be employed for U-Prove if the U-Prove specification [16] were modified accordingly. As a result the specification will presumably become rather complex.

Alternatively, one could also embed U-Prove into the idemix framework [18]. As the idemix implementation treats different cryptographic building blocks such as signature, commitment, and verifiable encryption schemes as different modules and orchestrates

them guided by issuing and proof specifications (cf. Section 3.2), the Brands signature scheme [5] could be integrated as an alternative to the CL signature scheme [8].

7. IMPLEMENTATION

We have implemented the data-minimizing authentication framework shown in Figure 1. In particular, we implemented the CARL policy and the claim languages, the pre-evaluation aspect of component (1a) as well as the components (2a), (2c), (3a) and (3b). Although the implementation is open for being used with any credential technology, the components (2c) and (3b) have been instantiated with the evidence generation and verification mechanisms that employ the idemix cryptographic library. Note that message signatures and disclosure to third parties are currently not implemented. All components of the framework have been released as Open Source Software under the Eclipse Public License and are available for download at http://www.primelife.eu, where also the idemix cryptographic library is available.

We are currently working on the implementation of the claim selection (2b) that presents the possible claims to the user so that she can make a selection accordingly. Once this is finished, applications can be built that employ our framework for privacy-friendly authentication. Building such an application could for instance be realized by integrating our framework with an XACML access control engine. Ardagna et al.[1] give an intuition on how this could be done.

8. CONCLUSION

We presented an important reason that hinders privacy-friendly authentication to be used in practice today, namely, the lack of a framework that utilizes privacy-friendly credential technologies, such as anonymous credentials, for authentication purposes. In this paper we describe all necessary components that allow for an implementation of such framework. We propose a simple claim language that provides adequate expressivity to address the core functionality of anonymous credential systems. Further, we describe how those functionalities are mapped to the concrete evidence specification languages of idemix and U-Prove.

We implemented the proposed framework and connected it to the existing idemix implementation. We show how the latter should be amended to attain the full expressivity of our claim language. Using our implementation has the following advantages, namely, (1) users benefit from significantly increased more privacy, (2) service providers gain in data quality due to the certified data being used, and (3) service providers substantially reduce the risks associated with holding large sets of sensitive information.

We envision to continue this trail of thought and provide a mapping from the claim language to SAML. By using SAML as WS-Trust security token, our data-minimizing authentication scenario may be implemented by means of current standards, which would also benefit its adoption.

9. REFERENCES

[1] Claudio A. Ardagna, Sabrina De Capitani di Vimercati, Gregory Neven, Stefano Paraboschi, Franz-Stefan Preiss, Pierangela Samarati, and Mario Verdicchio. Enabling Privacy-Preserving Credential-Based Access Control with XACML and SAML. In *Proc. of the Third IEEE TSP*, 2010.

[2] Patrik Bichsel and Jan Camenisch. Mixing identities with ease. In Evelyne De Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *IDMAN '10*, pages 1–17. Springer, November 2010.

[3] Patrik Bichsel, Jan Camenisch, Franz-Stefan Preiss, and Dieter Sommer. Dynamically-changing interface for interactive selection of information cards satisfying policy requirements. Technical Report RZ 3756, IBM Research Zurich, 2009. domino.research.ibm.com/library/cyberdig.nsf.

[4] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

[5] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

[6] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In *Proc. 15th ACM CCS*, pages 345–356. ACM Press, November 2008.

[7] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proc. 13th ACM CCS*, pages 201–210. ACM Press, 2006.

[8] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT '01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[9] Jan Camenisch, Sebastian Moedersheim, Gregory Neven, Franz-Stefan Preiss, and Dieter Sommer. A Card Requirements Language Enabling Privacy-Preserving Access Control. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, 2010.

[10] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. http://eprint.iacr.org/2002/161, 2002.

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

[12] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. http://www.ietf.org/rfc/rfc5280.txt.

[13] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[14] OASIS. Assertions and protocols for the OASIS security assertion markup language (SAML) v2.0, 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[15] OpenID authentication 2.0, December 2007. http://openid.net/developers/specs/.

[16] Christian Paquin. U-Prove cryptographic specification V1.1. Technical report, Microsoft Corporation, February 2011.

[17] Christian Paquin. U-Prove WS-Trust Profile V1.0. Technical report, Microsoft Corporation, February 2011.

[18] Security Team, IBM Research Zurich. Specification of the identity mixer cryptographic library. IBM Research Report RZ 3730, IBM Research Division, April 2010. http://domino.research.ibm.com/library/cyberdig.nsf.

[19] K. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. RFC 4510 (Proposed Standard), June 2006. http://www.ietf.org/rfc/rfc4510.txt.