RZ 3762 (# 99772) 01/15/2010
Computer Science 20 pages

Research Report

A Language Enabling Privacy-Preserving Access Control

Jan Camenisch, Sebastian Mödersheim, Gregory Neven, Franz-Stefan Preiss, and Dieter Sommer

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Email: {jca,smo,nev,frp,dso}@zurich.ibm.com

A Language Enabling Privacy-Preserving Access Control

Jan Camenisch, Sebastian Mödersheim, Gregory Neven,
Franz-Stefan Preiss, and Dieter Sommer
IBM Research Zurich, Rüschlikon, Switzerland
{jca,smo,nev,frp,dso}@zurich.ibm.com

January 2010

Abstract

We address the problem of privacy-preserving access control in distributed systems. Users commonly reveal more personal data than strictly necessary to be granted access to online resources, even though existing technologies, such as anonymous credential systems, offer functionalities that would allow for privacy-friendly authorization. An important reason for this lack of technology adoption is, as we believe, the absence of a suitable authorization language offering adequate expressivity to address the privacy-friendly functionalities. To overcome this problem, we propose an authorization language that allows for expressing access control requirements in a privacy-preserving way. Our language is independent from concrete technology, thus it allows for specifying requirements regardless of implementation details while it is also applicable for technologies designed without privacy considerations. We see our proposal as an important step towards making access control systems privacy-preserving.

1 Introduction

Current industry trends such as software as a service and cloud computing drive businesses to open up their software infrastructures to a wider online audience. Enterprises that used to populate their internal user directories by doing their own identity vetting are now moving away from their closed-world assumption and start relying more and more on external identity providers instead. On the other hand, as users' personal data move outside the enterprise boundaries, privacy protection becomes an even bigger concern than before. Clearly, companies have to strike a trade-off between the protection of their resources and the privacy of their users. A crucial tool in implementing this trade-off is an adequate language to express access control requirements in a way that does not force users to reveal more personal information than strictly necessary.

A plethora of technologies exist to authenticate users and bind attributes to them, including X.509 certificates, SAML, OpenID, trusted LDAP directories, and anonymous credentials [18, 11, 15]. The latter offer a number of appealing privacy features, such as proving predicates over attributes, disclosing attributes to third parties, accountable anonymity, and privacy-friendly consumption control. In this paper, we abstract all of these technologies into a generic model of digital "cards" that captures the advanced features they offer, but makes abstraction of the technical details. (The idea of using an abstraction is not new. In particular, the Identity Metasystem [28] advocates an abstraction called "identities", but our card model is slightly different. See Section 2 for details.)

We model a *card* as an authenticated list of attribute-value pairs issued by an issuer to a card owner. To obtain access to a server's resource, the card owner creates a *claim*, i.e., a statement about some of the attributes of one or more of her cards. She also generates (technology-dependent) *evidence* for the claim that is basis for the server (also called the *relying party*) to verify the authenticity of the claim, the freshness of the evidence, and fulfillment of the policy protecting the resource.

In this paper, we propose a card-based access control requirements language (CARL) that allows the server to express the requirements that a user's cards have to satisfy in order to gain access to a resource, without having to worry about the specifics of the underlying technology. Moreover, our language is privacy-preserving in the sense that it expresses the *minimal* claim that a user has to present. It does so in terms of which cards have to be involved in the claim, which attributes of those cards have to be revealed, and which conditions have to hold over the attributes (whether these were revealed or not). This approach allows the user to minimize the amount of data that she reveals to the server, which is important because cards often contain sensitive personal information. Moreover, some of the supported technologies (including SAML, OpenID, and in particular anonymous credentials) allow to derive claims that do not necessarily reveal all of the attributes of a card, but only a subset of them, or even just the fact that their values satisfy a certain boolean condition. Our policy language leverages these technologies to their full potential, without making compromises on compatibility with older technologies though, since the user can always choose to reveal more information than required.

We summarize the core features of our language below. While some of the individual features may also be supported in previous work, we see as important contribution of our language that it supports all of these features *simultaneously*. See the next section for a more detailed comparison with related work.

Privacy preservation. The CARL is privacy-preserving in the sense that it supports the principle of minimal information disclosure. Rather than assuming that all attributes in a card are revealed by default, it explicitly specifies which attributes have to be revealed, and clearly distinguishes between the requirement to reveal the value of an attribute (e.g., the date of birth) and the requirement that an attribute has to satisfy a certain condition (e.g., age greater than 18). Our policy language also supports *accountability*, so that anonymity can be revoked by a third party in case of abuse.

Technology independence. Our language is independent of the technology underlying the cards, so that different technologies or even a mix of technologies can be used without modifying the policy specifications. Also, its concepts are detailed enough to leverage the advanced features of available technologies (in particular those of anonymous credentials) such as consumption control and attribute disclosure to third parties.

Multi-card claims. Our policy language can express requirements involving multiple cards at the same time and has a way to refer to individual cards and the attributes they contain. It can thereby impose "cross-card" conditions, e.g., a car rental service can request to see an identity card, a driver's license, and a credit card, which are all registered to the same name. The *card type* determines the attributes a card contains, and the policy can specify a card to be of a certain type. For example, the car rental service may want to see the address as stated on the credit card, not on the driver's license. Being able to reference individual cards is also important when a user has multiple cards of the same type. For example, when a user has two credit cards, the policy should be unambiguous about whether it wants to see the credit card number and security code of the *same* card or of *different* cards. (We refer to this issue as the *card mixing problem*.)

Formal semantics. We provide a formal semantics for our language that mathematically defines an ideal system that determines what a particular realization must implement. This ideal system defines the proof goal for any realization with a concrete technology. We do not define a particular kind of realization in terms of a concrete set of actions (such as an exchange of particular messages), because this would be technology dependent.[1] Defining a formal semantics has helped us to surface subtle problems that might otherwise go unnoticed, such as the card mixing problem mentioned above.

In this paper, we focus only on the language that is used to describe card requirements. When designing a comprehensive access control language, various additional aspects have to be taken into account, such as how to specify the resources and requested actions to which the policy applies, how to express trust relationships and trust delegation, how to state data handling policies, how to combine multiple applicable policies, etc. In Section 6, we sketch how our language can be integrated into XACML [32] to profit from the mechanisms

[1] An example of such a realization is for instance described in Mödersheim and Sommer [29].

for defining the applicable resources and actions, and how it can be integrated into existing authorization languages like SecPal [6] and DKAL [24] to express delegation and other complex trust structures.

2 Related Work

Card-based access control can be seen as a generalization of a variety of access control models. Role-based access control [21, 34] can be seen as a special case of our card-based setting by encoding a user's roles as attributes in a card. Attribute-based access control [9, 32, 36] comes closer to our concept of card-based access control, but it does not see attributes as grouped together in cards.

The idea of technology independence for authentication mechanisms is not new. In particular, the Identity Metasystem [28] advocates a very similar abstraction, but focuses solely on use cases where only a single card (called "identity" in their framework) is used at each authentication session. The associated WS-* suite of policy languages and the CardSpace implementation support selective attribute disclosure and even a limited set of predicates over attributes, but again only for a single card per authentication. We think that multi-card authentication is an important use case, even though it necessarily adds to the complexity of the policy language; hence the need for our language. Advanced features such as disclosure to third parties, signing statements, and consumption control are also not supported in the Identity Metasystem.

Bonatti and Samarati [9] propose a language for specifying access control rules based on "credentials", the equivalent of our cards. The language focuses on card ownership and does not allow for more advanced requirements such as revealing of attributes or signing statements. The same is true for the language proposed in [3].

Winsborough et al. [37] present a credential-based access control language that allows to impose attribute properties on credentials, but does not support advanced features such as revealing of attributes, signing statements, or consumption control. The extension by Li et al. [26] supports revealing of attributes, but not the other features.

In the PolicyMaker [8] and KeyNote [7] trust management system, "credentials" are used to bind "assertions" to keys. In PolicyMaker assertions can be described in any safe language, while KeyNote fixes a language. Both could be used to encode attribute-value pairs in the assertion to implement our card concept, but selective revealing of attributes would not be possible, unless new credentials are re-issued at each login.

The languages mentioned above are not targeted to anonymous transactions and lack the ability for expressing semantics for obtaining accountability in anonymous transactions, which we do achieve through a combination of signing statements and disclosure to third parties. The first paper towards third-party disclosure is due to Backes et al. [5]. Gevers and De Decker [23] extend P3P to describe credentials and necessary access control requirements, including verifiable encryption (which we generalized to disclosure to third parties), but no precise syntax or semantics are specified.

The recent language by Ardagna et al. [2] features a card typing mechanism and advanced features such as consumption control and signing statements. It focuses only on anonymous credential systems though, hence cannot be used in combination with other technologies. It also lacks a formal semantics and, suffers from the card mixing problem.

Several logic-based approaches for authentication and distributed access control have been devised to achieve similar goals as this work: a technology-neutral, declarative specification of distributed access control [1, 10, 22, 27]. The fundamental idea is to describe access control requirements by formulae in authentication logic and to provide a calculus for proving such formulae. To gain access, the requester constructs a valid proof for the formula based on the credentials she owns and sends it, together with the relevant credentials, to the server. However, none of these approaches is fully privacy-preserving: they do not support selective disclosure of attributes (as opposed to transmitting entire credentials/cards), proving predicates over attributes (as opposed to disclosing these attributes), disclosing attributes to third parties, and signing statements with respect to the proved attribute properties.

In contrast, CARL specifies the minimal amount of information each involved party has to learn for access to be granted. The formal semantics is specified not as a calculus but as conditions on the cards/credentials that the requester holds and the precise amount of information the involved parties gain. It is then the

duty of an implementation (e.g. based on Identity Mixer) to show that users can only prove statements that hold on their credentials and that involved parties do not learn more information than specified. We briefly discuss how to combine CARL with existing logic-based approaches in Appendix 6.1; we leave an in-depth investigation of such combinations for future work.

3 Background

In the following we explain in more detail our abstract model of a "card". We also discuss some of the technologies that can be used to instantiate this model and the functionalities that they offer, because they have strongly influenced the design of our language. Finally, we sketch how our language fits into the bigger picture of a complete privacy-enhanced card-based access control solution.

3.1 Our Card Model

The language that we present is geared towards enabling user-centric and privacy-preserving access control based on certified cards. While it leverages the advanced features offered by anonymous credential systems, it is technology-agnostic in the sense that it makes abstraction of the particular technology that is used to certify the cards.

To better understand our model, we will at each step illustrate the concepts by describing how they are instantiated by X.509 v3 certificates [19]. These allow a certification authority (CA) to bind arbitrary community-specific attributes to a user's public key by creating a signature (under the CA's public key) of the user's attributes and her public key. In the next subsection, we sketch how a number of other technologies can also be seen to implement the same concepts.

A *card* is issued by a card *issuer* to a card *owner*. The issuer vouches for the correctness of the information on the card with respect to the intended owner. The information is meant to affirm qualification, typically in the form of *identity* or *authority*. However, the meaning of the information has no technical relevance and is subject to interpretation of the party relying on it. In an X.509 certificate, the CA acts as card issuer. While we are mainly interested in cards that can be presented to third parties, some cards may be intended for internal use only and be verifiable only by the issuer; think for example of a company-internal LDAP directory.

A card consists of a list of attribute-value pairs and of technology-specific auxiliary data called the *pre-evidence*. The pre-evidence can contain meta-data (cryptographic or other) that the owner needs when presenting the card to a relying party. In an X.509 certificate, e.g., the pre-evidence contains the CA's public key pk_{CA}, the user's public key pk_{U}, the user's secret key sk_{U}, and the CA's signature σ_{CA} on the list of attributes and pk_{U}. (For simplicity, we assume that the CA is a root authority; otherwise, the pre-evidence also contains the certificate chain of pk_{CA} to a root authority.) The issuing process may be carried out on-line, e.g., by visiting the issuer's website, as well as off-line, e.g., at the local town hall.

Cards are always of a certain *card type* that specifies the list of attributes that the card contains. For example, a national ID card may contain the first name, last name, date of birth, and address of the owner, while a movie ticket contains the time and date of the showing and a seat number. We consider a hierarchical ontology of card types so that types can inherit from other types; see Section 4.1 for details. A card serves as a means for proving qualification, i.e., it typically serves to prove identity, authority, or both. For example, a national ID card can be used to prove identity, a movie ticket to prove authority to attend a particular movie showing from a particular seat, and a driver's license to prove identity as well as the authorization to drive motor vehicles of a certain category.

For gaining access to a resource protected by a policy, the server has to be convinced that the policy is fulfilled. To do so, in our system model the card owner makes a *claim* about the cards she owns and about the attributes they contain. Claims are made independent from concrete technology, and are authenticated by accompanying *evidence*. The evidence is, however, specific to the technology underlying the cards.

In the ideal case of privacy-friendly technologies, this claim reflects exactly the policy. For other technologies, the claim is something stronger than required. With X.509, e.g., all cards' attributes are revealed,

no matter what the policy requires.

The evidence is derived from the card's pre-evidence, and used by the server to check (1) the integrity of the claim, (2) the freshness of the evidence, and (3) the rightful ownership of the card. Depending on the technology, the user may be able to derive the evidence herself, or she may need to interact with the card issuer. Likewise, the server may be able to independently verify the evidence, or may need the help of the issuer.

For X.509, for example, the evidence consists of pk_{CA}, pk_{U}, σ_{CA}, and a signature σ_{U} created by the owner on the claim statement and a random nonce chosen by the relying party. Here, σ_{CA} protects the integrity of the attributes, while σ_{U} simultaneously acts as a proof of freshness of the claim and a proof of ownership (through the knowledge of sk_{U}) of the card. The evidence is independently created by the owner; the relying party may have to contact the issuer however to check that pk_{U} has not been revoked.

3.2 Example Technologies

By the above card abstraction our policy language can specify access control restrictions without having to worry about the underlying technology. This means that a card of any supported technology can be used to satisfy a policy, and even that cards of different technologies can be combined in a single claim.

We already described how X.509 certificates fit our card model. Below, we sketch how anonymous credentials, trusted LDAP directories, OpenID, Kerberos, SAML, and even everyday email accounts can be seen to fit our model as well. This list is by no means exhaustive; in fact, we envision that most existing and even future technologies will fit our model.

Anonymous Credentials Much like an X.509 certificate, an anonymous credential [18, 11, 15] can be seen as a list of attribute-value pairs signed by the issuer with an underlying secret signing key for the user. Unlike X.509 certificates, however, anonymous credentials have the advantage that the owner can reveal subsets of the attributes, or merely prove that a condition over the attributes holds. Also, they provide additional privacy guarantees like unlinkability, meaning that, even when colluding with the issuer, a server cannot link multiple visits by the same user or link a visit to the issuing of the card.

Two main anonymous credential systems have been implemented today, namely Identity Mixer [15, 17] and UProve [20]. We will focus mainly on Identity Mixer in this paper because of its multitude of associated cryptographic tools such as verifiable encryption and consumption control (also known as "limited spending"), but we stress that UProve credentials can be used as a technology for our policy language as well.

LDAP The Lightweight Directory Access Protocol defines a standard interface to directory servers containing information about users. If the LDAP server is trusted, one can see each user directory entry as a card belonging to that user and issued by the LDAP server. The user authenticates to the LDAP server using a password or using a more advanced authentication mechanism.The evidence of a claim is simply the LDAP server's URL, the relying party can verify the attributes simply by looking them up in the directory. Any subset of attributes can be revealed, even though a cheating server can always look up other attributes as well.

OpenID In OpenID [33] each account is identified by a unique URL bound to the user by the OpenID provider. The recent OpenID Attribute Exchange extension [25] allows for the exchange of user-defined attributes. When logging on to a website, the user sends her unique URL to the relying party. The user is then redirected to her OpenID provider where she authenticates using a password. If the authentication is successful, the OpenID provider either sends a confirmation and the requested attributes directly to the relying party, or sends these through the user with integrity protection with a MAC (under a key that the provider and relying party agreed upon directly).

An OpenID account can be seen as a card issued by the OpenID provider. Any subset of attributes can be revealed in a claim, which is good for privacy, but on the negative side all transactions with the same card

are linkable through the unique URL. Also, since claim creation and verification both involve the OpenID provider, he learns which user authenticates to which server at which time. The user's password acts as a proof of ownership; the integrity of the attributes is protected by MACs.

Kerberos A Kerberos user account [30] could also be seen as a card issued by the key distribution center (KDC) containing the user's identity. Each claim derived from this card contains the user's identity; the Kerberos ticket acts as the evidence. The pre-evidence is the user's password and perhaps connection information to the KDC. Both claim creation by the user and verification by the relying party require interaction with the KDC.

SAML SAML [31] is an XML-based framework for communicating user authentication, entitlement, and attribute information. The user's attributes are stored by an Identity Provider (IP). When logging in to a relying party, the user authenticates to the IP and requests a signed claim (called *assertion* in SAML) containing the attribute values that the relying party requires. SAML is somewhat technology agnostic in the sense that it does not specify how the user has to authenticate to the identity provider, but the evidence of the claim is always an XML Signature by the IP. The user's account at the IP containing all the attributes can be seen as a card in our framework. The IP is the issuer of the card, the user's pre-evidence consists of her authentication secret with respect to the IP.

Email Even a simple email account can be seen as a card: the only attribute is the email address itself, the mail server acts as the issuer. The user's pre-evidence is her account password, the integrity and ownership of the email address are checked by the owner's ability to click on a link or enter a code sent to her by email.

3.3 Functionality

We now describe a number of special features of cards that our policy language supports and sketch how these features are or could be implemented in the different technologies.

Proof of ownership To bind a card to its legitimate owner, the card's pre-evidence may contain information that is used to authenticate the owner. This could be a picture of the user, a PIN code, a password, or a signing key. Proving card ownership in our notion means that the owner authentication is successfully performed with whatever mechanism is in place. Depending on the employed mechanism, successful authentication may also provide a liveness guarantee (to prevent replay attacks).

The actual implementation of the ownership proof is technology dependent. For X.509 certificates, ownership can be proven by signing a random nonce. In anonymous credentials, users construct a zero-knowledge proof of knowledge of an underlying master secret. OpenID, LDAP, Kerberos, and email accounts all work with a password-based approach.

Related to the question of ownership is the *transferability* of a card from one user to another. Some cards may be transferable (e.g., movie tickets) while others may not (e.g., driver's licenses and identification cards in general). We do not further elaborate this concept here, but rather assume that the underlying technology prevents abuse if necessary, for example by letting the underlying authentication secret be the same as that of a highly sensitive card, e.g., the signing key of a national ID card.

Selective attribute disclosure Some technologies allow attributes within a card to be revealed selectively, i.e., the relying party only learns the value of a subset of the attributes contained in the card.

Not all technologies support this feature. Verification of the issuer's signature on an X.509 certificate requires all attribute values to be known. The LDAP protocol specifies the attributes to fetch, but the user has no control over which attributes a cheating server looks up. Anonymous credentials, SAML, and (under certain settings) OpenID, on the other hand, have native support for this feature, although the mechanisms are quite different. For OpenID and SAML the provider simply only reveals those attributes that were

explicitly requested, while for anonymous credentials it is the cryptography that ensures that no information is leaked about non-disclosed attributes.

Proving conditions on attributes Anonymous credentials even allow one to prove conditions over attributes without revealing their actual values. (While in theory any condition can be proved, this mechanism is only truly practical for certain classes of conditions. We refer to [2] for details.)

For all other technologies the only way to prove that an attribute satisfies a condition is by revealing its value. Perhaps future versions of technologies with online verification such as OpenID or SAML will enable the identity provider to confirm that conditions over attributes hold, rather than having to reveal their values.

Attribute disclosure to third parties Usually attributes will be revealed to the relying party enforcing the policy, but the policy could also require certain attributes to be revealed to an external third party. For example, the server may require that the user reveals her full name to a trusted escrow agent, so that she can be de-anonymized in case of fraud, thereby adding accountability to otherwise anonymous transactions. As another example, an online shop could require the user to reveal her address to the shipping company directly, rather than disclosing it to the shop.

Of course, the relying party needs some sort of evidence that the user actually did reveal the necessary information to the third party. Identity Mixer elegantly supports this feature using verifiable encryption [13, 4, 16]. Here, the user hands to the relying party a ciphertext containing the relevant attribute(s), encrypted under the third party's public key, and adds a zero-knowledge proof that the correct attribute was encrypted. Moreover, a data handling policy can be bound to the ciphertext, e.g., to describe under what conditions it can be decrypted. The relying party is unable to change the data handling policy when forwarding the ciphertext to the third party.

This feature could be added to other technologies as well by letting either the user or the issuer send the necessary attributes to the third party directly, and letting the third party sign a receipt that can be shown to the relying party.

Signing of statements Our policy language also enables the server to require the user's explicit consent to some statement, e.g., the terms of service or the privacy policy of the site. The signature acts as evidence that this statement was agreed to by a user fulfilling the policy in question.

There are various ways in which users can express consent; our language does not impose a particular implementation. Using X.509 certificates the most straightforward way is to digitally sign the document. Anonymous credentials allow to sign statements while maintaining maximal privacy: anyone can verify that the signature was placed by a user satisfying the access control policy, but only a trusted opening authority can tell who exactly the user was. Using other technologies, the relying party could simply record the fact that the user confirmed the statement by clicking an OK-button.

Note that these signatures, together with attribute disclosure to third parties, are the key to make anonymity revocable and actions accountable. Consider, e.g., a policy requiring a user to reveal her real name n and an order confirmation number o to a trusted third party. In addition, the user has to sign a statement involving o. In case of a dispute, the server forwards the signature to the trusted party who can now resolve the issue with the person named n. The server itself, however, cannot find out this name. With conventional non-privacy-friendly technologies, the signature will also reveal the real name of the user to the server though.

Consumption control The server may want to impose limitations on the number of times that the same card can be used (or "consumed") to access a resource, e.g., to specify that each national ID card can only be used once to vote in an online opinion poll.

Bowers et al. [10] previously proposed a logic-based policy language for consumable cards (or *credentials* in their work). To each card, they associate a single global consumption limit and a *ratifier*, a central entity who keeps track of each usage of the card. This works well for settings such as electronic cash where the

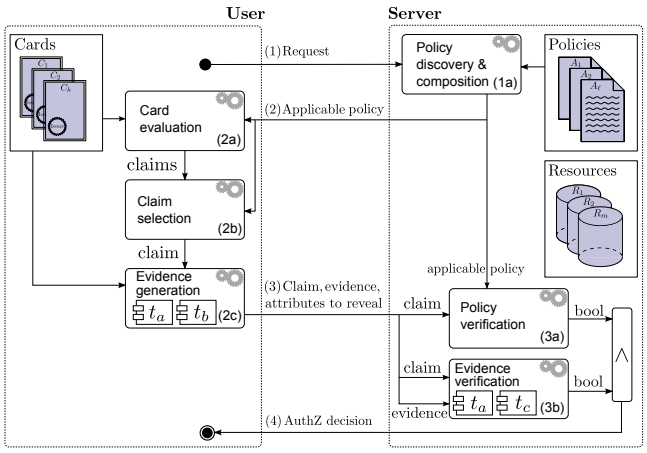

Figure 1: Decision rendering in our system model

issuer of the card determines how often the card is to be consumed *globally*, but falls short of covering use cases like the opinion poll example where the relying party wants to impose a limit how often the same card can be used *locally*.

We therefore extend their consumption functionality in our language such that the relying party itself can specify (1) the *consumption amount*, i.e., the number of units that is to be consumed from the card per access; (2) the *consumption limit*, i.e., the maximum number of units that can be consumed from the card before access is refused; and (3) the *ratification scope* being a URI defining the "scope" of the consumption, i.e., the limit becomes relative to this scope.

Part of the scope URI could be used to identify the central entity who keeps track of the number of consumptions, but it can additionally encode the scope within which consumption is to be tracked. For example, the access restrictions of the opinion poll service would be such that a different URI is used for each poll, specifying that each ID can be used once *for each poll*, rather than that the user can only take part in a single poll. The scope mechanism allows also to express more complex limitations. For example, to prevent two resources A and B from being accessed more than n times total per month, the scope URI in both policies is set to $append(\text{`examplescope:AorB:'}, currMonth(), \text{`/'}, currYear())$.

Some anonymous credential systems support consumption control (usually referred to as *limited spending*) natively in the underlying cryptography [12, 14]. They do so in a highly privacy-preserving way: the user's anonymity and unlinkability remain guaranteed until overspending occurs.

None of the other technologies have native support for consumable cards, but for all of them support could be added by letting the issuer play the role of ratifier, or by adding a unique serial number that is revealed whenever the card is consumed. (The latter approach makes all transactions of the same card linkable, though.)

3.4 System Model

Our system model involves three kinds of entities: users, servers (also called service providers), and issuers. Users hold certified cards $C_1, \ldots, C_k$ that they have obtained from issuers and want to access protected resources $R_1, \ldots, R_m$ (e.g., web pages, databases, web services) hosted by the servers. Servers restrict access to their resources by means of access control policies $A_1, \ldots, A_\ell$. Rather than simply specifying which user is allowed to access which resources by means of a classical access matrix, the policies contain requirements in terms of the cards that a user needs to own in order to be granted access.

Figure 1 depicts how authorization decisions are rendered in our model. Initially, a user contacts a server to request access to a resource she is interested in (1).

Having received the request, the server responds with the access control policy applicable for the re-

source (2). The applicable policy may be a composition (1a) of multiple policies the server holds, e.g., it may contain a number of alternative policies of which one must be satisfied to obtain access. It contains the *card requirements* expressing which conditions on which card attributes have to hold, which attributes have to be revealed to whom, and who the server trusts as issuers for these cards. For each attribute to be revealed, the server may also specify in a *data handling policy* what he aims to do with the obtained information, e.g., how long he will retain the data, or for which purposes he will use it.

Upon receiving the policy, the user's system evaluates which claims she can derive from her available cards that fulfill the given policy (2a). For example, a policy requiring a valid travel document may be fulfilled by means of the user's national identity card or her passport, while the validity may be shown by proving its expiration date to be in the future or by disclosing the exact date. The favored claim is then chosen interactively by the user, or automatically by a data-minimizing heuristic (2b). In addition, the user decides whether she agrees with the server's data handling policy, and whether she wants to proceed. In this case, evidence for the chosen claim is generated (2c) and sent, together with the claim and the attributes to reveal, to the server (3). As claims are expressed independent from a concrete technology, the user's systems must have respective means available (e.g., in the form of a plug-in for technology t_a) to generate evidence specific to the technology of the used cards.

Finally, the server verifies whether the policy is implied by the claim (3a) and if the evidence supports the validity of the claim (3b). If so, access to the resource is granted (4). Clearly, to verify the evidence the server must support the same technology that was used to generate the claim.

Finding possible claims in step (2a) could, e.g., be done by finding logical proofs with the policy formula as proof goal using the available cards as premises. A user could then select one proof and according to the proof-carrying paradigm [1] this proof would be sent to the server as sample solution for the claim verification in step (3a).

To implement the scenario sketched above, a number of languages need to be available. In particular, languages to express the access control policy on the server (which includes the card requirements and the data handling policies), to communicate the policy to the user, to express the claims as sent by the user, and to describe the cards that the user owns. In case the user's evaluation of the data handling policy shall be automated, also a language to express a user's privacy preferences is needed.

In this paper, we focus on the card requirements language, which is used as part of the access control policy on the server, as well as of the communicated policy to the user. The language contains placeholders for the data handling policies associated to revealed attributes, but we do not make this language explicit. We do not make any of the other languages explicit either.

4 Language

The language we propose is intended for expressing requirements on cards that have to be fulfilled in order to gain access to some resource. These requirements involve (1) the ownership of cards of the right type and issued by the right authority, (2) the disclosure of attributes certified in these cards to the verifier or to third parties, and (3) conditions on attributes expressed in a mathematical formula. In addition, (4) the signing of statements as well as (5) the consumption of cards can be required. To fulfill a policy, *all* requirements stated in it must be fulfilled.

A policy in our card-based access control requirements language (CARL) can only be fulfilled 'as a whole' (cf. Section 5.2). To combine multiple policies, e.g., offering a choice among a set of alternative claims to satisfy, they need to be embedded into other languages. In Appendix 6.2 we describe how this can be done for XACML.

Our language is typed, therefore we begin with depicting the basic properties of the typing. Then we illustrate how the particular requirements are expressed in our language.

4.1 Typing and Ontology

To help policy designers avoid specification and interpretation mistakes, we make use of a type system in a similar way as many programming languages do. We distinguish between *card types* and *data types*. We first motivate the use of card types and then describe the type system's properties.

For a server to make access control decisions, he needs to distinguish the different kinds of cards and attributes he processes, as their meanings may differ depending on their type. For example, consider a university issuing digital student IDs and diploma certificates having the same basic format (e.g., both contain the student's name and field of study). These cards do clearly have different purposes and must be distinguishable from one another. Relying only on the fact of who issued a card would be insufficient for determining its purpose and trustworthiness. Also, standard attributes such as 'name' have meaning only in conjunction with additional information such as a card type. To this end, we make use of card types (e.g., drivers license, residence permit) that specify the attributes contained in cards of this type.

We therefore assume as given an ontology $\mathfrak{T}$ that defines a set of data types, a set of card types, a partial order on those card types, as well as a set of functions and relations on the data types.

First, we assume that the ontology defines the data types $\beta_1, \ldots, \beta_n$. Examples of data types include *Int*, *String*, *Date*, *Boolean* and *URI*. Data types need not necessarily be disjoint, e.g., URIs are typically also strings. We denote by $[\![\beta]\!]$ the extension of a data type β (the set of constants of type β). A constant c can have several types, i.e., every type β such that $c \in [\![\beta]\!]$.

Further, we assume that the ontology specifies a set of card types $\tau_1, \ldots, \tau_m$. These types are similar to record types in programming languages: every card type τ is defined by a set of attributes with their types $A_\tau = \{a_1 :: \beta_1, \ldots, a_l :: \beta_l\}$, where we denote with $a :: \beta$ an attribute a that has type β. Like a record in a programming language, a card can be represented as a function from the attributes to values of the respective attribute type. For instance a card with attributes $\{a_1 :: \beta_1, a_2 :: \beta_2\}$ is a function f with domain $\{a_1, a_2\}$ such that $f(a_1) \in [\![\beta_1]\!]$ and $f(a_2) \in [\![\beta_2]\!]$. Thus, the extension of a card type τ (the set of all cards of type τ) is defined as the set of all such functions:

$$[\![\tau]\!] = \{f :: (A_\tau \to [\![\beta_1]\!] \cup \ldots \cup [\![\beta_n]\!]) \mid \forall (a :: \beta) \in A_\tau.\, f(a) \in [\![\beta]\!]\}$$

Additionally, like classes in object-oriented programming languages, a card type τ_1 may be *extended* by another card type τ_2, i.e., τ_2 *inherits* all attributes of τ_1 and includes further attributes. This induces a partial order $\leq_\mathfrak{T}$ on card types such that if $\tau_2 \leq_\mathfrak{T} \tau_1$ then $A_{\tau_1} \subseteq A_{\tau_2}$. Thus every card of type τ_2 may be used in place of τ_1. For example, *Passport* $\leq_\mathfrak{T}$ *PhotoID* models that *Passport* is lower in the type hierarchy than (i.e., a subtype of) *PhotoID* and can be used in place of it.

Finally, the ontology should define a set of functions and relations on the data types. We assume to have at least the equivalence relation $=_\beta$ on every data type β. We also assume the addition and a total order on both types, *Int* and *Date*. Functions may depend on the state of the access control system on which it is evaluated, e.g., the function *today*() can be defined to return the current date. For a function symbol f of the ontology, we denote by $f^\mathfrak{T}$ the function defined for this symbol by the ontology, similarly the relation symbol R is interpreted as the ontology-defined relation $R^\mathfrak{T}$.

In the following we only consider specifications that are type correct w.r.t. the type inference system given in Appendix B.

4.2 Expressing requirements

We now describe how the different kinds of card requirements are expressed in our policy language. We first give a basic intuition for our language by means of an example policy, and then use it as a running example to discuss the different elements of our language in detail. The full syntax of our language is found in Appendix A.

01: own p::*Passport* issued-by USAGOV
02: own r::*ResidencePermit* issued-by PITTSBGHTOWNHALL
03: own c::*CreditCard* issued-by VISA, AMEX
04: reveal $c.number$, $c.expDate$ under 'purpose=payment'
05: reveal $r.address$ to SHIPCO under 'purpose=shipping'
06: sign 'I agree with the general terms and conditions.'
07: where $p.dateOfBirth \leq dateMinusYears(today(), 21) \wedge$
08: $c.expDate > today()$

The policy states that access is granted to users who (1) are at least twenty-one years old, (2) reveal their valid credit card information for payment purposes, (3) reveal their address to the shipping company SHIPCO for shipping purposes and (4) agree to the general terms and conditions. Here, an American passport must certify the age, the payment data must be certified by a valid Visa or American Express card, and a residence permit from the city of Pittsburgh must certify the address. We assume the ontology defines the functions $dateMinusYears(ref, k)$, subtracting k years from date ref, and $today()$, returning the current date.

4.2.1 Proof of ownership

The most basic requirement expressible is the ownership of a card of a specific type by a specific issuer. For each required card the policy contains a line of the form (cf. lines 01–03 in the example policy):

$$\text{own } c :: \mathit{Type} \text{ issued-by } I_1, \ldots, I_n$$

Here, c is a *card variable* used to refer to this card within the policy. We already discussed card type ontologies in Section 4.1; we assume that each type *Type* is unambiguously defined by a uniform resource identifier (URI). In the same way, we assume that the allowed card issuers $I_1, \ldots, I_n$ are referred to by means of URIs. Depending on the underlying technology, these URIs may be mapped to public keys (for X.509 and anonymous credentials), server URLs (for LDAP and OpenID), or any other way of pointing to a card issuer. Specifying multiple issuers indicates that any of these issuers is accepted. Omitting the issued-by clause indicates that no restrictions are imposed on the issuer, meaning that even self-issued cards are allowed.

4.2.2 Attribute disclosure

To require disclosure of a card attribute's value, a line of the form (cf. lines 04–05 in the example policy) is stated:

$$\text{reveal } c.att \text{ to RECIPIENT under } dhp$$

The attribute to reveal is specified by the card variable c as defined in a previous own line, and the attribute name *att* as defined by the card type, separated by a dot.

By using the keyword to, the policy can optionally specify the recipient (identified by a URI) to whom the attribute is to be revealed. If no recipient is specified, then the intended recipient is assumed to be the server enforcing the access control. For proving the policy, the server has to be provided with evidence showing that the attribute values were indeed transmitted to the recipient. In the example policy, the credit card data has to be revealed to the server itself (line 04) and the address to the shipping company (line 05).

Optionally, the server may attach a data handling policy *dhp* describing how the server intends to treat the information after receiving it, e.g., the intended purpose of the data, retention periods, further recipients, etc. The policy could be specified in natural language or in a machine-interpretable language like P3P [35]; we do not impose any particular format though.

4.2.3 Conditions on Attributes

Using the keyword where, a policy may state a formula ϕ expressing conditions on the cards' attributes. For example, lines 07–08 require the date of birth to be at least 21 years in the past and the credit card to be valid.

In general, a formula allows for applying the standard operators for comparison (numeric and non-numeric), arithmetics and logics (cf. Appendix A). Functions and relations may be applied to expressions whereby we assume a built-in standard set of these (such as date and time arithmetics, string manipulation, etc.) defined in the ontology $\mathfrak{T}$ together with their meaning. Expressions may further be qualified attributes, constants or basic variables. Basic variables are different from card variables as they are of a data type, rather than a card type. Basic variables that appear in ϕ may act as substitute for (1) a card's issuer, (2) an attribute that has to be revealed, (3) the purpose and (4) recipient of attributes to reveal as well as (5) the statement to sign. Our running example could, e.g., as well have been written as own c::*CreditCard* issued-by i while extending ϕ with $i = \text{VISA} \vee i = \text{AMEX}$. In case a basic variable x appears in the list of attributes to reveal, the concrete value for x, under which ϕ is satisfied, must be revealed (cf. Section 5.3).

4.2.4 Signing of statements

By including a line of the form

sign statement

a policy requires to sign the statement in the following sense: it is ensured that the signature was produced by someone who fulfills (the rest of) the policy. For example, line 06 of the policy example requires signing of the statement 'I agree with the general terms and conditions'.

4.2.5 Consumption control

Limitations on the number of times that the same card can be used to obtain access can be imposed by lines of the form

consume *amount* maximally *limit* of c scope *scope*

where *amount* is the consumption amount, *limit* is the consumption limit, c is the card to be consumed, and *scope* is the ratification scope (cf. Section 3.3).

For example, the Pittsburgh Theater hands out discount cards entitling its holder to buy theater tickets at a reduced price. The below policy protects the theater's discounted ticketing service and states that students of the University of Pittsburgh are eligible for six discounts per year:

own *sid*::*StudentID* issued-by PITTSBGHUNIVERSITY
own *dc*::*DiscountCred* issued-by PITTSBGHTHEATER
consume 1 maximally 6 of *dc* scope *s*
where $s = append(\text{'urn:scope:pbgTheater:year:'}, currYear())$

To access the service, proofs of ownership for a student card and a discount card need to be provided to the server. For granting access, the server verifies that the sum of the amounts of all consumptions of the discount card (including the current one) in the given scope does not exceed the limit. Each time a student successfully purchases a discounted ticket, one unit is consumed. (More units could, e.g., be charged for extra-long shows.)

5 Semantics

We now provide a formal semantics for our language. The semantics abstractly defines the intended behavior of an access control system for a given policy and thereby defines the obligations that an actual realization must meet. The context of this definition is a state transition system with transitions for issuing cards, proving a policy, and other transactions such as revocation of cards or application-specific actions. As our CARL is concerned only with the conditions for fulfilling a policy and the effects of proving a policy, so is our semantics.

We define the formal semantics of CARL in three steps. First, we define the meaning of a policy formula based on the ontology and an interpretation of the variables. Second, for a given policy and a set of cards that a user owns, we define if the user fulfills the policy. Third, a user who fulfills the policy can prove this

fact to the server and we call this simply *proving the policy*; we define the effect that proving the policy has on the knowledge of the server and the trusted third parties.

Note that, in the last step, we define only the *effect* of proving a policy, rather than describing how such a proof could be performed. This is because such actions depend on the concrete technology that is employed, from which our language abstracts. Indeed, when mapping CARL onto a concrete technology by translating the policy specification into a sequence of actions within the respective card system, our semantics sets out two obligations for this translation: 1) actions can only be performed by a user who owns the necessary cards and 2) other parties learn from the action exactly the information specified by the semantics. As not all technologies offer the same level of privacy protection, we also provide a variant of our semantics that allows for more information to be disclosed than what is required by the policy. An example for such a mapping is provided by Mödersheim and Sommer for the Identity Mixer anonymous credential system [29].

5.1 Formula Semantics

We define the meaning of policy formula with respect to a given ontology $\mathfrak{T}$ and an interpretation $\mathcal{I}$ that maps all basic and card variables to values of the respective type. Recall that the ontology provides the interpretation $f^{\mathfrak{T}}$ for every function symbol f and $R^{\mathfrak{T}}$ for every relation symbol R, respectively. For a card C, an attribute a, and terms $t_1, \ldots, t_n$, we define $(C.a)^{\mathcal{I}} = (C^{\mathcal{I}})(a)$ and $f(t_1, \ldots, t_n)^{\mathcal{I}} = f^{\mathfrak{T}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$.

Our typing system ensures that all these values are well-defined for a given $\mathcal{I}$. The satisfaction relation is defined as:

$$\mathcal{I} \models \phi_1 \wedge \phi_2 \text{ iff } \mathcal{I} \models \phi_1 \text{ and } \mathcal{I} \models \phi_2\ , \quad \mathcal{I} \models \neg\phi \text{ iff } \mathcal{I} \not\models \phi\ ,$$
$$\mathcal{I} \models R(t_1, \ldots, t_n) \text{ iff } R^{\mathfrak{T}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$$

Other constructs of the CARL formulae are the usual abbreviations, e.g., $\phi_1 \vee \phi_2$ is short for $\neg(\neg\phi_1 \wedge \neg\phi_2)$. Further, we define $\phi_1 \models \phi_2$ iff for all $\mathcal{I}$ holds $\mathcal{I} \models \phi_1$ implies $\mathcal{I} \models \phi_2$. Finally, we define $\phi_1 \equiv \phi_2$ iff $\phi_1 \models \phi_2$ and $\phi_2 \models \phi_1$. These definitions are similar to other definitions in formal logic; following these standards improves the understanding of the concepts described by our language.

5.2 Fulfilling a Policy

The second step is concerned with the question whether a user owns cards sufficient to satisfy a given policy. We denote with $\mathfrak{P}_U$ the set of cards that the user U owns. We now consider the local state of the parties, the conditions on a party's state to fulfill a policy, and the transition of parties' states when proving policies. For now, we assume that a U's state contains at least the set of cards $\mathfrak{P}_U$ the party U owns.

We first look at a policy without consume lines, which we consider in Section 5.4. Let U be a user and S be a server and U be known to S under the identifier I. This identifier may for instance be a one-time pseudonym in privacy-friendly technologies or (linkable) username otherwise.

Let P be the policy of S that U needs to satisfy in the following form:

own $C_1 :: \tau_1, \ldots, C_n :: \tau_n$
reveal $t_{(1,S)}, \ldots, t_{(n_S,S)}$
reveal $t_{(1,S_1)}, \ldots, t_{(n_{S_1},S_1)}$ to S_1
...
reveal $t_{(1,S_m)}, \ldots, t_{(n_{S_m},S_m)}$ to S_m
where ϕ
sign s

We require that the variables of P and of the current state are disjoint (which can be achieved by a suitable renaming of the policy variables). Note that we here omitted the issued-by part of the own line because we treat the issuer of a card simply as an attribute (of type *URI*) and the issuer can thus be specified as part of the formula ϕ itself. Indeed, issued-by $I_1, \ldots, I_n$ for a card C is just an abbreviation for $C.issuer = I_1 \vee \ldots \vee C.issuer = I_n$ (as an additional conjunct of ϕ).

We say that U *can fulfill the policy* P *under card assignment* p *and interpretation* $\mathcal{I}$ iff

1. p is a total mapping from $\{C_1, \ldots, C_n\}$ to $\mathfrak{P}_U$,
2. for every $1 \leq i \leq n$ it holds that $p(C_i) :: \sigma_i$ for a type $\sigma_i \leq_{\mathfrak{T}} \tau_i$ and $C_i^{\mathcal{I}} = (A_{\tau_i} \lhd p(C_i))$ where $A \lhd f$ is the domain restriction of function f to set A, and
3. $\mathcal{I} \models \phi$.

We say that U *can fulfill the policy* P iff there exists such an interpretation $\mathcal{I}$ and mapping p.

5.3 Proving a Policy

When a user can fulfill a policy and chooses to prove the policy to another participant, the effect of this transition is an increase of knowledge of all participants to which information is revealed. We denote with $\mathfrak{K}_S(I)$ and $\mathfrak{K}'_S(I)$ formulae characterizing the knowledge of a party S about a user I before and after the transition, respectively. As before, I is an identifier under which the user is known to S.

For each part $O \in \{S, S_1^{\mathcal{I}}, \ldots, S_m^{\mathcal{I}}\}$ to whom information is revealed, we require

$$\mathfrak{K}'_O(I) = \mathfrak{K}_O(I) \wedge \phi \wedge (t_{(1,O)} = t^{\mathcal{I}}_{(1,O)}) \wedge \ldots \wedge (t_{(n_O,O)} = t^{\mathcal{I}}_{(n_O,O)}).$$

This models that each party O learns that the user acting under the identifier I owns cards with property ϕ together with the concrete values $t^{\mathcal{I}}_{(i,O)}$ of the terms $t_{(i,O)}$ that have been revealed to O. Additionally, the server S obtains a signature on the statement $s^{\mathcal{I}}$ with respect to the formula proved to S, i.e., $\phi \wedge (t_{(1,S)} = t^{\mathcal{I}}_{(1,S)}) \wedge \ldots \wedge (t_{(n_S,S)} = t^{\mathcal{I}}_{(n_S,S)})$.

To model unlinkable values, we use variables in a particular way here. For this, first recall that we require a renaming of the policy variables so that no variable occurs in the considered state. Therefore, when a user uses the same card several times in proving policies to the same or to different servers, this card is referred to by a fresh variable each time. This reflects that the servers, even when they cooperate, are unable to decide whether particular authentications have been performed with the same or with different cards. However, for what concerns all the information revealed by a single transaction for fulfilling a policy P to several different parties, the additional knowledge uses the same variable names for all parties. For two servers S and S' who cooperate, we define their shared knowledge about a user identified as I simply as: $\mathfrak{K}_{S,S'}(I) = \mathfrak{K}_S(I) \wedge \mathfrak{K}_{S'}(I)$. This reflects that they can relate only information about a transaction that was done under the same pseudonym I.

For proving a policy in a typical system realization, the user's system creates a claim that implies the policy to be fulfilled and sends it with accompanying evidence to the server (cf. Section 3.4).

Our definition of the knowledge of servers characterizes the ideal case of privacy-friendly technologies: one does not reveal more information than required by the policy. Conventional card systems cannot achieve this as cards are transmitted as a whole. Thus, for conventional technologies we relax the constraints of our semantics, allowing for the release of more information than necessary, formally, any formula ψ that implies the above $\mathfrak{K}'_O(I)$ formula.

5.4 Consumption Control

We now extend the semantics to specify the behavior of consumable cards. In our model, consumption of cards is tied to a ratification scope that controls the consumption (cf. Section 3.3). In a nutshell, one can spend a card if the value of all its consumptions (including the current one) in the ratification scope does not exceed the consumption limit.

To model the consumption, we introduce global *log files*, one log file for each consumption scope (these log files are only part of the ideal world that is modeled by our transition system; they usually will not have a counterpart in an real implementation). Each log file is a list of pairs $\langle C, k \rangle$ of a card C and a positive integer k, representing that k units of card C are spent in the domain represented by the log file. The *balance of card* C in a log file is the sum of all consumptions of C in the log file.

Consider a policy P as before with the following additional consumption line (where C is a card of some own line of P):

consume k maximally n of C scope *scope*

We define that P can be fulfilled for $\mathcal{I}$ and p if 1) the policy resulting from P by removing the consumption line according to the definition in Section 5.3 can be fulfilled and 2) the balance of card $p(C)$ in the log file of $scope^{\mathcal{I}}$ plus $k^{\mathcal{I}}$ is less than or equal to $n^{\mathcal{I}}$.

Similarly, we extend the definition of proving P, i.e., it will have the same effects as before, plus the effect that the log file of $scope^{\mathcal{I}}$ is augmented with the pair $\langle p(C), k^{\mathcal{I}} \rangle$.

We have covered here only a policy with one consumption line, the extension to several consumption lines is as expected, where we assume that the interpretation of the consumption scope $scope^{\mathcal{I}}$ (which usually is a constant chosen by the server) is never the same for different consumption lines.

6 Language Integration

The language we present addresses the specification of credential requirements in credential-based access control. In this section we sketch how our language can be integrated with other languages to utilize the strengths of those.

6.1 Abstract Authorization Languages

For expressing authorization, delegation, and trust, e.g., within a large organization, a complete enumeration of business units, employees, and their relations and access rights is usually not feasible and too inflexible. Therefore, authorization languages such as SecPal [6] and DKAL [24] have been developed. They are based on a simple but powerful mechanism: they specify Horn clauses on abstract facts, i.e., rules of the form "if facts $f_1, \ldots, f_n$ hold, then also fact f holds." The facts represent either direct statements, e.g., "X says Y", initially given relationships such as trust relationships, or derived facts that are consequences of other facts by the Horn-clauses.

All these facts thus represent a high-level view that is not related to concrete credentials, signatures and the like. Our CARL language can provide this relationship, i.e. specifying precisely what credentials and properties correspond to a particular abstract fact like "X says Y" or "Z is-an-adult". A connection between CARL and Horn-clause based authorization language can thus be made by specifications of the form $f \Leftarrow P$ for an abstract fact f and a CARL policy P. This means that one way to derive the fact f is to fulfill the policy P.

For example, one may specify with Horn clauses that X can access resource R if owner O of R has given permission to X. Permission can be given directly by O, or O could have delegated that to a deputy D and D granted access. CARL policies specify the concrete credentials and properties that correspond to the access permission from O or from D as well as a credential for the delegation from O to D. The combination tells us that one may either access R when (1) one has the credential that corresponds to the permission from O or (2) when there are credentials that show that O delegated this right to D (this may already be known by the respective policy enforcement point) and the credential that D issued the permission for access.

Combining the CARL with the abstract facts of a Horn-based language gives a powerful and high-level method to specify complex authorization, trust, and delegation rules on credentials. Observe that a delegation chain as in the example cannot be specified in CARL (because the length of the chain is unbounded). In fact, the focus of CARL is on the specification of the credentials, their conditions, and the revealing of information to different parties, exactly what the abstract Horn-based languages abstract from. The languages are thus complementary and it makes sense to distinguish between a high-level deduction engine and the connection to concrete credentials.

6.2 XACML

The access control language XACML [32] is the de facto standard for attribute-based access control policies. In order to reuse existing XACML infrastructure for privacy-friendly credential-based access control, we briefly describe how CARL can be integrated into XACML.

First, we define a number of new XML elements that can occur inside an XACML `<Rule>` to make XACML credential-aware. Each own, reveal, sign, and consume line in CARL is translated into a corresponding `<Own>`, `<Reveal>`, `<Sign>`, and `<Spend>` element in the XACML `<Rule>`. The schema of these new elements is such that they encode in a structured way all arguments on the corresponding lines in a CARL policy. Each `<Own>` element has an attribute `CredentialId` containing a URI by which this credential can be referred to within this `<Rule>`. Finally, the formula ϕ is encoded within the standard XACML `<Condition>` element using built-in data types and functions [32, Appendix A], but we extend the `<AttributeDesignator>` element with an extra attribute `CredentialId` to indicate from which credential the attribute should be taken.

Second, we make a number of architectural changes to make XACML privacy-friendly. Namely, standard XACML does not provide a mechanism for conveying an access control policy to the user since it does not assume that the policy is known by the user. This is against the idea of privacy-aware access control where a user provides only the credentials/attributes necessary for fulfilling a particular policy. A possible way of conveying a policy to the user would be to embed the policy in XACML's `<StatusMessage>` element that is optionally contained in an XACML access response.

In order to solve the architectural issue, a server could run a modified XACML policy decision point (PDP) that in case an access request is denied *and* the applicable XACML policy contains a CSL specification in its `<Condition>` element, it returns these CSL requirements embedded in the `<StatusMessage>` of the negative access response. In order for a user to learn the policy for a specific resource, she makes a request without providing any attributes whereupon the PDP will respond with the negative response that contains the CSL policy. Knowing that policy, the user can provide only the attributes necessary for this particular policy.

7 Conclusions & Future Work

We presented a language for specifying requirements on a user's cards to be used for obtaining access in any kind of open access control setting. We not only provide a formal language specification, but also formally define the semantics as to help future systems designers and implementers to avoid mistakes through ambiguous interpretation of the text. Our language aims to serve as a central piece in an open access control setting. It enables the use of a plurality of underlying card technologies that are available today or are becoming available, such as OpenID or anonymous credentials. Each card technology can be used for access control by providing a mapping from the verification process of the respective technology to a policy in CARL. Our language enables properties of privacy and anonymity through data minimization while retaining accountability. When being deployed together with technologies such as anonymous credential systems, a privacy-preserving user-centric model of access control can be realized as the user is put into full control over her data.

Our next step towards an open and privacy-enhancing system for access control is to link the feature set and formalism of CARL with concrete card systems. Mödersheim and Sommer [29] describe such a connection for the Identity Mixer anonymous credential system. A proof that this connection agrees with the CARL semantics, as well as providing similar connections for other technologies are part of our ongoing work.

8 Acknowledgments

The authors thank Mario Verdicchio and Günter Karjoth for fruitful discussions. This work was supported by the European Community's Seventh Framework Programme through the projects "PrimeLife" (grant agreement no. 216483) and "AVANTSSAR" (grant agreement no. 216471).

References

[1] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of Computer and communications security*, pages 52 – 62. ACM, 1999.

[2] C. A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio. Exploiting cryptography for privacy-enhanced access control. *Journal of Computer Security*, 18(1):123–160, 2010.

[3] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. *J. Comput. Secur.*, 16(4):369–397, 2008.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[5] M. Backes, J. Camenisch, and D. Sommer. Anonymous yet accountable access control. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 40–46. ACM New York, NY, USA, 2005.

[6] M. Becker, C. Fournet, and A. Gordon. Design and semantics of a decentralized authorization language. In *20th IEEE Computer Security Foundations Symposium, 2007 (CSF'07)*, pages 3–15, 2007.

[7] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, 1998.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1996.

[9] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

[10] K. D. Bowers, L. Bauer, D. Garg, F. Pfenning, and M. K. Reiter. Consumable credentials in linear-logic-based access-control systems. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2007*. The Internet Society, 2007.

[11] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[12] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 04*, pages 132–145. ACM Press, 2004.

[13] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 331–345. Springer-Verlag, 2000.

[14] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 201–210. ACM Press, 2006.

[15] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer-Verlag, 2001.

[16] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer-Verlag, 2003.

[17] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In V. Atluri, editor, *ACM CCS 02*, pages 21–30. ACM Press, 2002.

[18] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, Oct. 1985.

[19] 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`.

[20] Credentica. U-Prove SDK overview: A Credentica white paper, 2007. `http://www.credentica.com/files/U-ProveSDKWhitepaper.pdf`.

[21] D. Ferraiolo and R. Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, 1992.

[22] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. A linear logic of authorization and knowledge. In *ESORICS 2006*, volume 4189 of *Lecture Notes in Computer Science*, pages 297–312, 2006.

[23] S. Gevers and B. De Decker. Automating privacy friendly information disclosure. Technical Report CW441, K.U. Leuven, Dept. of Computer Science, April 2006.

[24] Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *IEEE 21st Computer Security Foundations Symposium, 2008. CSF'08*, pages 149–162, 2008.

[25] D. Hardt, J. Bufu, and J. Hoyt. OpenID attribute exchange 1.0, Dec. 2007. `http://openid.net/developers/specs/`.

[26] J. Li, N. Li, and W. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 46–57. ACM New York, NY, USA, 2005.

[27] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6(1):128–171, 2003.

[28] Microsoft. Microsoft's vision for an identity metasystem. Whitepaper, `http://msdn.microsoft.com/en-us/library/ms996422.aspx`, 2005.

[29] S. Mödersheim and D. Sommer. A formal model of identity mixer. Technical report, IBM Zurich Research Lab, 2009, 2009. Submitted, available as Research Report RZ 3749, `domino.research.ibm.com/library/cyberdig.nsf`.

[30] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, 1994.

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

[32] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. Available from: `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

[33] OpenID authentication 2.0, Dec. 2007. `http://openid.net/developers/specs/`.

[34] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[35] W3C. The platform for privacy preferences (P3P). `http://www.w3.org/TR/P3P11/`, November 2006.

[36] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In V. Atluri, M. Backes, D. A. Basin, and M. Waidner, editors, *ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, pages 45–55. ACM, 2004.

[37] W. Winsborough, K. Seamons, and V. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 1, 2000.

A Grammar

The following shows the grammar of our policy language:

Policy	=	CredDef$^+$ RevealDef* SpendDef* ['where' Formula] [SignDef] ;
CredDef	=	'own' CredVar '::' CredTypeIdent ['issued-by' IdentityTerm (',' IdentityTerm)*] ;
RevealDef	=	'reveal' ValueList ['to' IdentityTerm] ['under' Term] ;
SpendDef	=	'consume' Term 'maximally' Term 'of' CredVar 'scope' Term ;
SignDef	=	'sign' Term ;
CredVar	=	Identifier ;
CredTypeIdent	=	Identifier ;
IdentityTerm	=	Term ;
ValueList	=	Value ['::' TypeIdent] [',' ValueList] ;
TypeIdent	=	Identifier ;
Formula	=	Formula '$\wedge$' Formula
	|	Formula '$\vee$' Formula
	|	'$\neg$'Formula
	|	'(' Formula ')'
	|	Exp RelationOp Exp
	|	RelationName '(' [ExpList] ')' ;
Exp	=	Term
	|	Exp ArithmeticOp Exp
	|	FunctionName '(' [ExpList] ')' ;
RelationOp	=	'=' | '<' | '>'
	|	'$\leq$' | '$\geq$' | '$\neq$' ;
Term	=	Value | Constant ;
ArithmeticOp	=	'+' | '$-$' | '$\cdot$' | '$\div$' ;
Value	=	CredVar '.' AttrIdent
	|	BasicVar ;
ExpList	=	Exp [',' ExpList] ;
RelationName	=	Identifier ;
FunctionName	=	Identifier ;
Constant	=	Identifier ;
AttrIdent	=	Identifier ;
BasicVar	=	Identifier ;
Identifier	=	Alpha Alphanum* ;

Alpha and Alphanum are alphabetic and alphanumericcharacters. IdentityTerm must map to the URI of an identity. CredTypeIdent must map to a credential type. TypeIdent must map to a data type. An Identifier cannot be a keyword.

B Type Inference System

To limit specification mistakes in CARL policies, we have introduced typing and though types can be automatically inferred in many cases, it is recommended for the policy writer to explicitly specify the types of all variables, both basic and credential variables.

We describe the type system now by a type inference system similar to the ones used in the definition of programming languages. We use type judgments of the form $\Gamma \vdash t :: \tau$ to denote that under the type definitions in Γ, it can be derived that term t is of type τ. Further, we use type judgments of the form $\Gamma \vdash \phi$ to express that ϕ is type correct according to Γ. Initially, the set Γ contains the following. First, it includes the types for all constant, function, and relation symbols that are defined by the ontology. Note that we do allow several type definitions for the same symbol, e.g. '=' is a comparison relation for each type, e.g. $\mathbb{P}(\mathit{Int} \times \mathit{Int})$; similarly, e.g., a string constant may also have type URI. This models the overloading of symbols. Second, the initial Γ includes a unique type for every variable of the specification. Since for basic variables the type specification is optional, one must "guess" a type for each unspecified variable initially and verify that with that guess, the policy is well-typed. Note that variables for an issuer or the recipient of revealed data are always of type URI.

The axiom is that we can derive anything given:

$$\frac{}{\Gamma \cup \{t :: \tau\} \vdash t :: \tau}$$

We can now derive types of terms and formulae:

$$\frac{\Gamma \vdash t_1 :: \alpha_1 \quad \ldots \quad \Gamma \vdash t_n :: \alpha_n \quad \Gamma \vdash f :: \alpha_1 \times \ldots \times \alpha_n \to \beta}{\Gamma \vdash f(t_1, \ldots, t_n) :: \beta}$$

$$\frac{\Gamma \vdash C :: \tau}{\Gamma \vdash C.a :: \beta} \; (a :: \beta) \in A_\tau$$

$$\frac{\Gamma \vdash t_1 :: \tau_1 \quad \ldots \quad \Gamma \vdash t_n :: \tau_n \quad \Gamma \vdash R :: \mathbb{P}(\tau_1 \times \ldots \times \tau_n)}{\Gamma \vdash R(t_1, \ldots, t_n)}$$

$$\frac{\Gamma \vdash \phi_1 \quad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \wedge \phi_2} \qquad \frac{\Gamma \vdash \phi}{\Gamma \vdash \neg\phi}$$

Given a formula ϕ and an initial set of type definitions Γ, we say that ϕ is *well typed* iff $\Gamma \vdash \phi$ can be derived. If there is more than one derivation, we say that ϕ is *ambiguous*. While it is common that subterms of a formula may have different type derivations, there should be only one derivation for the entire formula. Otherwise, there is probably a typing problem. This could, e.g., lead to using a wrong instance of an overloaded function symbol. In such a case, the policy specifier should thus be advised of the ambiguity.