Credential-Based Access Control Extensions to XACML

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

Abstract

Access control and authentication systems are currently undergoing a paradigm shift towards openness and user-centricity where service providers communicate to the users what information they need to provide to gain access to a given resource. This paradigm shift is a crucial step towards allowing users to manage their identities and privacy. To ensure the service provider of the validity of the presented information, the latter is typically attested to by a trusted issuer or identity provider. There are multiple means to transmit such attestation to the service provider including X.509 certificates, anonymous credentials, and OpenIDs. In this position paper, we advocate to abstract all attestation means into the concept of a 'credential' and propose to extend XACML so that it allows service providers to specify the set of credentials that a user is required to present to get access to a given resource. Our extensions not only allow one to express conditions on the credentials that the user has to present, but also which attributes have to be disclosed, and to whom, and which statements the user has to consent to before being granted access.

1 Motivation

Certified credentials form the cornerstones of trust in our modern society. Citizens identify themselves at the voting booth with national identity cards, motorists demonstrate their right to drive cars with driver licenses, customers pay for their groceries with credit cards, airline passengers board planes with their passports and boarding passes, and sport enthusiasts make their way into the gym using their membership cards. Often such credentials are used in contexts beyond what was originally intended: for example, identity cards are also used to prove eligibility for certain social benefits, or to demonstrate to be of legal age when entering a bar.

Each of these credentials contains attributes that describe the owner of the credential (e.g., name and date of birth), the rights granted to the owner (e.g., vehicle class, flight and seat number), or the credential itself (e.g., expiration date). The information in the credentials is trusted because it is certified by an issuer (e.g., the government, a bank) who on its turn is trusted.

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, and create a demand for mapping the credential-based society into the digital world. For example, collaborating enterprizes are setting up joint virtual infrastructures in the cloud that must be accessible to users from all partners. Also, enterprizes 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. Private companies as well as governments are already stepping in as external identity providers, by issuing identity credentials in the form of X.509 certificates, OpenIDs, or digital identity cards.

Although identity credentials are the ones most commonly used, we also consider credentials that do not contain information about the owner. For example, concert or movie tickets are credentials that describe only show and seating information, rather than the owner's identity.

The goal of the proposed extensions to XACML is to enable privacy-friendly authentication and access control on the base of all such credentials. The system we envision enables servers to express requirements on the users' credential attributes, and users to control — on a fine grained level —

which attributes from their credentials are disclosed and which are not. Clearly, this requires that users are told up-front what the applicable policy for successfully authenticating to a server or gaining access to a resource is. As most credentials contain sensitive identity information, such *privacy-enhanced credential-based access control* is essential to enable users to manage their identities in a privacy-friendly manner.

Eventually, such systems will replace the typical username-password accounts that are currently created by filling web forms with self-stated information. This evolution will be beneficial to both users and service providers: service providers will profit from the reliability of the provided information, while users will enjoy increased privacy as well as the convenience of not having to remember dozens of username-password combinations and not having to fill in the same information in web forms over and over again.

A crucial condition to leverage privacy-enhanced credential-based access control systems to their full potential is the availability of a suitable policy language in which a service provider can express the requirements which credentials a user needs to own. The claims languages used in the existing credential-based identity frameworks CardSpace [12] and Higgins [11] are limited to expressing the list of attributes that need to be revealed and the issuers that are trusted to certify these attributes. Moreover, policies in CardSpace only ask for single credentials rather than information out of multiple ones, and do not allow for expressing conditions over them. This falls short of many of the goals that we see as key for true privacy-friendly credential-based access control. In particular, what is missing is a language addressing the following needs:

- Credentials must be the basic unit for reasoning about access control. Users do not have stand-alone attributes, but rather own credentials that contain sets of attributes. Credentials must be of a certain *credential type* that determines the attributes contained in the credential. The policy specifies the credential type that must be used to satisfy the policy. For example, when a government issues birth certificates as well as ID cards, both of which contain the owner's last name, then the service provider can specify in his policy that he wants the last name as stated on an ID card, not on a birth certificate.
- When users have multiple credentials of the same type with the same issuer, they should not be able to mix attributes from different credentials, i.e., the language must be able to express whether attributes have to come from the same or from different credentials. For example, if a user who owns two credit cards is asked to reveal her credit card data, then she should not be able to reveal the number of one card and the expiration date of the other.
- The language must be independent of the underlying technology, in the sense that it makes no assumptions on the cryptography and network topology used to issue and prove statements about credentials. At the same time, it should provide generic concepts that enable the use of advanced features of available technologies, in particular those of anonymous credentials. Also, a technology-neutral language allows to deploy a combination of different technologies without modifying the policy specification.
- The language must be privacy-friendly and follow the principle of minimal information disclosure. Meaning, the policy should specify which attributes of a credential have to be revealed, rather than assuming that the user reveals all attributes contained in the credential by default, or even that she reveals all attributes in all of her credentials.
- The language must clearly distinguish between attributes that need to be revealed (e.g., name, date of birth) and the conditions that attributes have to satisfy (e.g., age greater than 18). For some technologies the only way to prove that a certain condition holds may be by revealing the relevant attributes, but this is not true for all technologies.

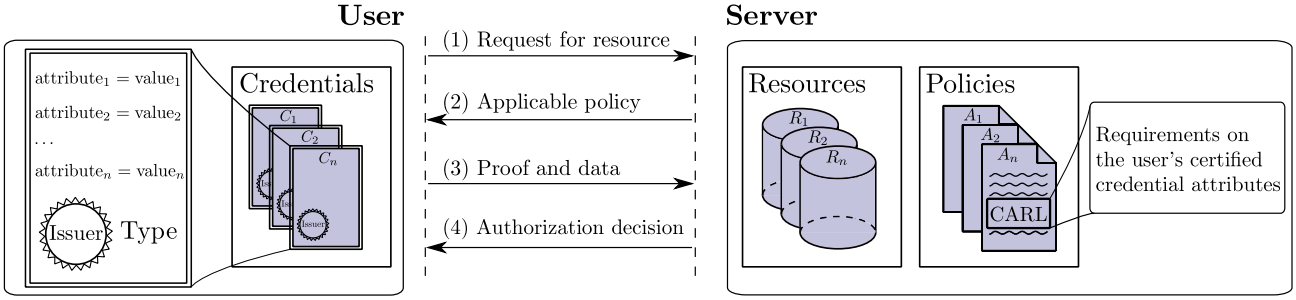

Figure 1: A Privacy-Enhanced Credential-Based Access Control Setting

2 Credential-based Access Control

2.1 Scenario

We consider a *privacy-enhanced credential-based access control* setting involving three kinds of entities: users, servers (also called service providers), and issuers. As shown in Figure 1, users hold trusted credentials $C_1, \ldots, C_n$ that they obtained from issuers and want to access protected resources $R_1, \ldots, R_m$ (e.g., web pages, databases, web services, etc.) hosted by the servers. Servers restrict access to their resources by means of access control policies $A_1, \ldots, A_n$. 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 credentials that a user needs to own in order to be granted access.

Figure 1 depicts the typical message flow between a user and a server. Initially, a user contacts a server to request access to a resource (1). The server responds with the access control policy applicable for the resource (2), containing the *credential requirements* that express which conditions on which credential attributes have to hold, which attributes have to be revealed, and who the server trusts as issuers for these credentials. Upon receiving the policy, the user evaluates whether she is able to fulfill the given requirements and whether she's willing to reveal the required information. If so, she produces a token that proves her fulfillment of the requirements and sends it, together with the attributes to reveal to the server (3). The exact format and content of this proof token depend on the underlying credential technology (see further). If the server successfully verifies the validity of the proof token, access is granted (4).

2.2 Credentials

By a *credential* we mean an authentic statement made by its issuer whereby the statement is independent from a concrete mechanism for ensuring authenticity. The statement made by the issuer is meant to affirm qualification, typically in the form of *identity* or *authority*. A credential serves as means for proving qualification, i.e., it typically serves as proof of identity, proof of authority, or both at the same time. For example, national identity cards are proofs of identity, movie tickets prove authorization to watch a particular movie from a particular seat, and driver's licenses authorize to drive motor vehicles of a certain category yet prove identity at the same time.

2.3 Credential Technologies

We propose extensions that allow the policy author to express his policy in a technology-independent way and allow a user to freely choose the technology to fulfill the policy — to the extent that the same technology is supported by the server. This includes the possibility to use credentials of different technologies to fulfill one policy. A multitude of different credential technologies already implement the generic concepts of credential-based access control. We highlight some candidate technologies here.

X.509. While the original purpose of X.509 certificates was merely to bind entities to their public signing and/or encryption keys, the latest version of X.509 v3 certificates [8] also allows additional community-specific attributes to be included in the certificate. When used as a credential mechanism,

the issuer essentially acts as a certification authority by signing certificates containing the bearer's list of attribute values as well as his public key. The bearer can prove ownership of the credential by proving knowledge of the underlying private key.

Anonymous Credentials. Much like an X.509 certificate, an anonymous credential [7, 2, 4] can be seen as a signed list of attribute-value pairs issued by an issuer. Unlike X.509 certificates, however, they have the advantage that the owner can selectively reveal any subset of the attributes, or merely prove that they satisfy some condition, without revealing any more information. Also, they provide additional privacy guarantees such as unlinkability between different visits by the same user. Two main anonymous credential systems have been implemented today, namely identity mixer (Idemix) [4, 6] and UProve [9].

OpenID. OpenID accounts [16] can be seen as credentials issued by the OpenID provider. The recent OpenID Attribute Exchange extension [10] allows user-defined attributes to be exchanged.

LDAP. One can see all of a user's attributes in an LDAP [17] directory tree as being contained in a single credential issued by the LDAP host, or one could exploit the hierarchical structure to group attributes together in multiple credentials. Attributes are verified simply by looking them up in the issuer's directory.

Kerberos. Even Kerberos tickets [14] could also be seen as digital credentials, albeit with a limited set of attributes being the user's identity, the server that the ticket gives access to, and some validity information.

2.4 Credential Functionality

We now describe a number of credential features that we leverage for credential-based access control, and sketch how these features could be supported in the different technologies. The extensions that we propose are able to express all the concepts listed below.

Proof of ownership. To bind a credential to its legitimate owner, authentication information may be tied to the credential in the form of a picture of the user, the hash value of a PIN, or the public key of a cryptographic identification scheme. Proving credential ownership in our notion means that authentication is successfully performed with respect to the authentication information whenever such information is present.

For X.509 certificates, users can prove ownership of the credential by signing a random nonce under the public key that is certified by the certificate. In anonymous credentials, users execute a zero-knowledge proof of knowledge of an underlying master secret. In OpenID the user authenticates to the OpenID provider by means of a password, for LDAP the user sends the password to the relying party.

Selective attribute disclosure. Some technologies allow attributes within a credential to be revealed selectively, meaning that the server only learns the value of a subset of the attributes contained in the credential. Not all technologies support this feature. For example, verification of the issuer's signature on X.509 certificates requires all attribute values to be known. Anonymous credentials and OpenID, on the other hand, have native support for this feature.

Proving conditions on attributes. Anonymous credentials allow to prove conditions over attributes without revealing their actual values by means of zero-knowledge proofs. OpenID could be extended supports this too, as the provider can simply confirm to the server that the condition holds.

Attribute disclosure to third parties. Usually attributes are revealed to the server that enforces 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. 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. The Idemix anonymous credential system elegantly supports this feature using verifiable encryption [3, 1, 5].

Signing of statements. The server may 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 transferable evidence that this statement was agreed to by a user fulfilling the policy in question.

Limited spending. The server may want to impose limitations on the number of times that the same credential can be used (or "spent") to access a resource, e.g., to specify that each user can only vote once in an online poll. The policy language should be able to express the *amount*, i.e., the number of units that have to be spent to obtain access, and the *spending limit*, i.e., the maximum number of units that can be spent until access is refused, and the *scope* with respect to which the units are to be spent. Some anonymous credential systems support this type of limited spending natively in the underlying cryptography. For other technologies the server can simply keep track how many times each credential was used to authenticate.

3 Proposed Extensions to XACML

We first present an example policy specified in a human-readable, non-XML-based language called Credential Access Requirements Language (CARL) and then suggest how the same concepts can be incorporated into XACML. The full syntax of CARL is given in Appendix A. The policy states that access is granted to users who (1) can prove that they are at least twenty-one years old, (2) reveal their valid credit card information for payment purposes, (3) reveal their address to ShippingCo for shipping purposes and (4) agree to the general terms and conditions. Here an American passport is needed to certify the age, a valid Visa or American Express card is needed for the payment data, and a residence permit from the city of Chicago must certify the address.

```
01: own p::Passport issued-by USAgov
02: own r::ResidencePermit issued-by ChicagoTownhall
03: own c::CreditCard issued-by Visa, Amex
04: reveal c.number, c.expirationDate under 'purpose=payment'
05: reveal r.address to ShippingCo under 'purpose=shipping'
06: sign 'I agree with the general terms and conditions.'
07: where p.dateOfBirth ≤ dateMinusYears(today(), 21) ∧
08:       c.expirationDate > today()
```

To make XACML credential-aware, we define a number of new XML elements that can occur inside an XACML `<Rule>`. Each own, reveal, sign, and spend 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 [15, 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.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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`.

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

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

[11] Higgins open source identity framework. `http://www.eclipse.org/higgins/`.

[12] Microsoft. Windows cardspace. `http://www.microsoft.com/net/windowscardspace.aspx`.

[13] 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`.

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

[15] 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`.

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

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

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     = 'spend' Term 'maximally' Term 'of' CredVar 'scope' Term ;
SignDef      = 'sign' Term ;

CredVar      = Identifier ;
CredTypeIdent = Identifier ;
IdentityTerm = Term ;
ValueList    = Value ['::' TypeIdent] [',' ValueList] ;
TypeIdent    = Identifier ;
Formula      = Formula '∧' Formula
             | Formula '∨' Formula
             | '¬'Formula
             | '(' Formula ')'
             | Exp RelationOp Exp
             | RelationName '(' [ExpList] ')' ;
Exp          = Term
             | Exp ArithmeticOp Exp
             | FunctionName '(' [ExpList] ')' ;

RelationOp   = '=' | '<' | '>'
             | '≤' | '≥' | '≠' ;
Term         = Value | Constant ;
ArithmeticOp = '+' | '−' | '·' | '÷' ;
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 alphanumeric characters. 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.