

RZ 3890  
Computer Science

(#ZUR1510-053)  
20 pages

10/21/2015

# Research Report

## A Prolog Program for Matching Attribute-Based Credentials to Access Control Policies


Jan Camenisch<sup>1</sup>, Sebastian Mödersheim<sup>2</sup>, Gregory Neven<sup>1</sup>, Franz-Stefan Preiss<sup>1</sup>,  
and Alfredo Rial<sup>1</sup>

<sup>1</sup>IBM Research – Zurich, 8803 Rüschlikon, Switzerland

<sup>2</sup>Technical University of Denmark

LIMITED DISTRIBUTION NOTICE

Some reports are available at <http://domino.watson.ibm.com/library/Cyberdig.nsf/home>.

 **Research**  
Africa • Almaden • Austin • Australia • Brazil • China • Haifa • India • Ireland • Tokyo • Watson • Zurich

# A Prolog Program for Matching Attribute-Based Credentials to Access Control Policies

Jan Camenisch<sup>1</sup>, Sebastian Mödersheim<sup>2</sup>, Gregory Neven<sup>1</sup>,  
Franz-Stefan Preiss<sup>1</sup>, and Alfredo Rial<sup>1</sup>

<sup>1</sup> IBM Research – Zurich, Switzerland

<sup>2</sup> Technical University of Denmark

**Abstract.** In an attribute-based credential system, users employ credentials issued by credentials issuers to compute presentation tokens that prove to service providers that the user’s credentials fulfill the access control policies to access services. The number of user credentials and the number of ways a policy can be satisfied can be large. Therefore, a user has to choose which subset of her credentials she wishes to employ to compute a presentation token. This choice has both efficiency and privacy implications. We present a Prolog program that lists all the credentials subsets that can be used to fulfill a given policy. In our program, credentials are represented by facts and policies by rules. By querying a rule, the Prolog engine lists all the combinations of facts that satisfy the rule. Therefore, we remark the simplicity of our approach, which simply requires representing credentials and policies in Prolog and avoids the need of implementing credential-policy matching or exhaustive search algorithms. Furthermore, our program is also useful on the verifier side. By using facts to represent the credential information disclosed by a user’s presentation tokens, when the user wishes to access a new service, the service provider can verify whether the credential information already disclosed fulfills the policy for that service. Our Prolog program implements a variety of features of an attribute-based credential system: pseudonyms, key binding, different restrictions for attributes values, issuer-driven and verifier-driven revocation, and inspection. Our program can easily be extended to implement more features.

## 1 Introduction

In an attribute-based credential system [6, 8, 7, 1, 2, 5, 4, 9], users receive credentials from credentials issuers. A credential is a container of user attributes that are certified by the credential issuer. Users employ their credentials to be granted access to services that are protected by access control policies. An access control policy describes the credentials that a user must possess in order to be granted access to a service. An access control policy may describe the type of non-revoked credentials that a user must possess, the identity of the issuers of those credentials, the type of attributes that the credentials must contain and restrictions on the values of those attributes.

A user computes a presentation token in order to prove to the service provider that she possesses credentials that fulfill an access control policy. A presentation token consists of a description of the credentials information that the user reveals to the service provider in order to prove that her credentials fulfill the policy, and a cryptographic proof that guarantees that the user indeed possesses credentials with such information. This cryptographic proof certifies that the user credentials fulfill the policy, but does not disclose any other information on the user's credentials.

In order to compute a presentation token, a user must first check whether her credentials fulfill the access control policy. Some access control policies could be fulfilled by different subsets of the users credentials. For example, consider a policy that restricts access to the books offered by a library. The policy requires users to be members of the library, which they can prove if they possess a credential issued by the library, or if they are students and nationals of the country where the library is located, which they can prove if they possess an identity card and a student card that store the corresponding credentials. If a user possesses those three types of credentials, the user can choose which ones to use in order to compute the presentation token. This choice may have both efficiency and privacy implications: on the one hand, proving possession of the credential issued by library could be more efficient than proving possession of two credentials on an identity card and on a student card; on the other hand, if the library has few members, proving possession of the credential issued by the library hides the user identity only in a small set of users.

In the example above, it is easy to compute the different subsets of credentials that a user can employ to satisfy the policy. However, in general, the number of credentials a user possesses and the number of ways a policy can be satisfied can both be large. Additionally, presentation tokens can be associated to a pseudonym. Presentation tokens are in general unlinkable, i.e., the verifier does not know whether two tokens were computed by the same or by different users unless the policy that the tokens fulfill allows the verifier to link them. However, a policy may require tokens to be linked through a pseudonym, or may allow users to choose whether to link her presentation tokens. Therefore, when computing a presentation token, a user is confronted with multiple combinations of credential subsets and pseudonyms.

We provide a Prolog program that, given a policy, allows the user to compute all the combinations of credentials and pseudonyms that can be employed to compute a presentation token that fulfills the policy. Prolog is a logic programming language and it is declarative, i.e., the program logic is expressed in terms of relations represented as facts and rules. Our Prolog program employs facts to represent the user pseudonyms and the user credentials information, such as the credential issuer, type and attributes, and employs rules to represent policies. By querying whether a rule is fulfilled by the existing facts, the Prolog engine computes and lists all the subsets of facts that fulfill the rule. Therefore, simply by representing users credentials as facts and policies as rules in Prolog, we obtain a program that outputs the desired credentials subsets. We remark

thus the simplicity of our approach in comparison to using other programming paradigms, which would require the implementation of both a credential-policy matching algorithm to know whether a subset of credentials fulfill the policy, and an exhaustive search algorithm to list all the subsets of credentials that fulfill the policy.

Our Prolog approach is also useful for the verifier. The verifier’s program represents the user pseudonyms and the user credential information disclosed by the user’s presentation tokens as facts, while the policies are represented as rules. When the user wishes to access a new service, the verifier can check whether the user pseudonyms and credential information disclosed before already fulfill the access control policy associated to the new service. To do this, like in the user program, the verifier runs a query to check whether the rule that represents the policy is fulfilled by the existing facts. Thanks to this program, the verifier can spare the user from computing a new presentation token when the facts known by the verifier already fulfill the policy.

*Outline of the paper.* In Section 2, we describe an attribute-based credential system. We depict the parties that are involved in the system and the functionalities it provides. In Section 3, we give a short introduction to Prolog. In Section 4, we describe our Prolog program. We show how credentials and presentation tokens are represented as facts and how policies are represented as rules, and we show some examples of queries and results. Finally, we conclude and hint future work in Section 5.

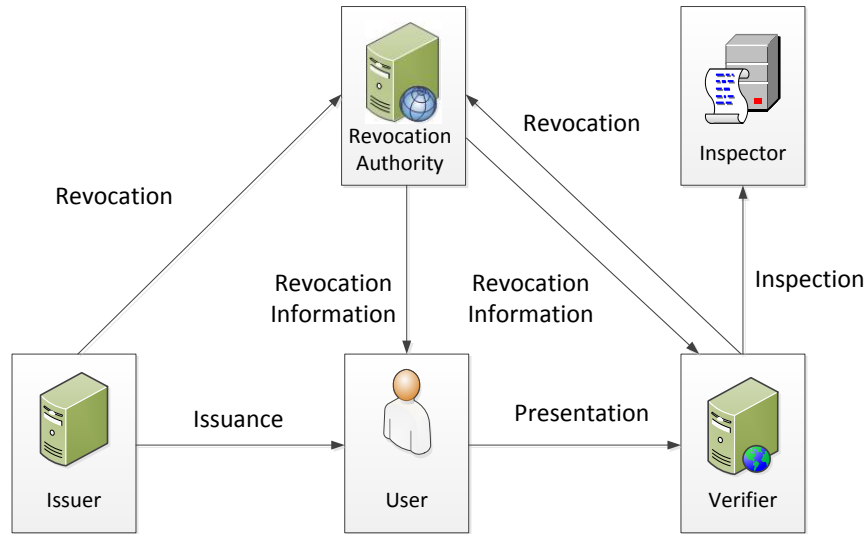
## 2 Concepts and Features of Attribute-Based Credentials

In this section, we follow the description of an attribute-based credentials (ABC) system in [3]. We depict an ABC system in Figure 1. The system consists of users, issuers, verifiers, revocation authorities and inspectors.

**User.** A user receives attribute-based credentials from one or more issuers. A user also receives from the revocation authority revocation information that shows whether her credentials are valid or revoked. When a user needs to prove to a verifier that her credentials fulfill the access control policy associated to a service, the user computes a presentation token on input her credentials, the issuer’s public key, and the revocation information, and sends the presentation token to the verifier.

**Issuer.** An issuer issues attribute-based credentials to the users. The issuer also publishes a public key and public parameters needed to compute and verify presentation tokens that employ the issued credentials. The issuer can also revoke credentials and send information on the revoked credentials to the revocation authority. The public parameters of the issuer specify the revocation authority responsible for the revocation of the credentials issued by that issuer.

**Verifier.** A verifier creates access control policies associated to services that need to be protected. An access control policy requires the user to possess



**Fig. 1.** Attribute-Based Credentials System

a number of credentials issued by trusted issuers. The attribute values in a credential may also be restricted. When a user wishes to access a service, the user sends a presentation token and the verifier checks whether the presentation token fulfills the associated access control policy. To verify the presentation token, the verifier employs the revocation information received from the revocation authority. Additionally, a verifier may also revoke credentials and send information on the revoked credentials to the revocation authority.

**Revocation Authority.** A revocation authority is responsible for revoking credentials and releasing revocation information to users and verifiers. A credential can be revoked by its issuer. In that case, the credential is no longer valid. The issuer parameters specify the identity of the revocation authority. Additionally, a credential can be revoked by a verifier. In that case, the credential remains valid for other verifiers. The verifier specifies in his access control policies the identities of the revocation authorities responsible for the revocation of the credential.

**Inspector.** An inspector is a trusted authority that receives presentation tokens from the verifiers and has the power to extract some information from those presentation tokens. Access control policies specify the conditions that must be fulfilled so that the inspector performs such an extraction. For this extraction to be possible, the access control policy associated to the presentation token must specify the identity of the inspector and the information that the inspector must be able to extract.

We describe now the elements and the functionalities of an ABC system.

**Credentials.** A credential is a certified container of attributes issued by an issuer to a user. Each of the attributes has an attribute type, which determines the meaning of the attribute, and an attribute value. The issuer must attest the validity of those attributes before issuing the credentials to the user. To verify the correctness of a credential, the issuer's public key must be employed. The issuer's public key must also be employed to verify presentation tokens that were computed on input one or more credentials issued by that issuer. Additionally, the issuer's public parameters also include the identity of the revocation authority responsible for revoking the issued credentials.

**Access Control Policies.** An access control policy is specified by a verifier in order to restrict access to a service. An access control policy is a description of the credentials that a user must possess in order to be granted access to a resource. For each credential, the access control policy specifies the issuer of the credential, the credential type, the attribute types that the credential must contain along with restrictions on the attribute values, and possibly the identity of an inspector and the information that an inspector must be able to extract about that credential from the presentation token. The revocation authority responsible for revoking each of the credentials is specified in the access control policy in the case of verifier-driven revocation and is given by the credential issuer in the case of issuer-driven revocation. Additionally, an access control policy can require credentials to be bound to the same secret key, and can also require a presentation token to be associated to a given pseudonym.

**Presentation Tokens.** A presentation token is computed by a user in order to access a service. In order to compute a presentation token, the user must possess the credentials specified in the access control policy associated to the service. A presentation token consists of a description of the information revealed from the user's credentials to fulfill the access control policy and a cryptographic proof that the user possesses those credentials. To compute and verify the cryptographic proof, the issuer's public key and the revocation information from the revocation authority is needed. The public key of the inspector is also needed for those credentials subject to inspection. A presentation token only reveals to the verifier that the user's credentials fulfill a given access control policy, but does not reveal to the verifier any information about the user's credentials that cannot be derived from the access control policy.

**Key binding.** Key binding is a countermeasure against users who share their credentials. The main idea is as follows. A user possesses a secret key. When a credential is issued, if the credential specification requires key binding, the issuer binds the credential to the user secret key. This binding is performed without the issuer learning the user's secret key. The issuer can require the user to prove that the secret key employed in the binding process equals the secret key bound to a previously issued credential. This way, the issuer certifies that the user's credentials are bound to the same user secret key.

An access control policy can require that two or more credentials be bound to the same key. Therefore, to compute a presentation token that fulfills such an access control policy, a user cannot employ credentials bound to different keys or credentials bound to a secret key that the user does not know.

**Pseudonymity.** Presentation tokens are unlinkable between each other, i.e., a verifier of two or more presentation tokens does not know whether they were computed by the same user or not, unless this information can be derived from the corresponding access control policies. Nevertheless, when key binding exists, the verifier can impose controlled linkability of presentation tokens by requiring them to be associated to a pseudonym. There are two types of pseudonyms: scope-exclusive or not. A scope-exclusive pseudonym is unique per verifier and per user secret key. Therefore, for a given verifier, all the presentation tokens computed by the user on input credentials that are bound to the same user secret key are linkable thanks to the scope-exclusive pseudonym. Presentation tokens for other verifiers remain unlinkable. Pseudonyms that are not scope-exclusive allow presentation tokens computed with the same user secret key and for the same verifier to remain unlinkable, although the user can choose to remove unlinkability by reusing the same pseudonym.

**Revocation.** Revocation is the process by means of which an issued credential changes its status from valid to not valid. A credential can be revoked for multiple reasons. We distinguish two settings: issuer-driven revocation and verifier-driven revocation. In issuer-driven revocation, the issuer of a credential decides whether a credential should be revoked. This type of revocation is global in scope. The issuer specifies the revocation authority responsible for the revocation of the issued credentials. In verifier-driven revocation, a verifier decides whether a credential should be revoked. This type of revocation is local in scope, i.e., the credential is still accepted by other verifiers. The verifier also specifies the identity of the revocation authority. The revocation authority publishes revocation parameters and updates regularly the revocation information that specifies whether a credential is valid or revoked. Typically, time is divided into epochs and the revocation authority publishes updated revocation information at the beginning of each epoch. In order to compute and verify presentation tokens, users and verifiers employ the revocation parameters and the last update of the revocation information released by the revocation authority. Presentation tokens only reveal that each of the credentials is valid or revoked, but not other information about the credentials beyond what can be derived from the access control policies.

**Inspection.** An access control policy may specify that, under some conditions, some information should be extracted from a presentation token. For this purpose, the access control policy specifies an inspector identifier or public key, the credentials and the attribute types from which some attribute values must be extracted, and a description of the conditions under which that extraction can be performed. In order to compute and verify presentation tokens, the user and the verifier need the inspector's public key, but the inspector is not involved. After verifying a presentation token, a verifier can

forward it to the inspector to perform the inspection. The inspector is a trusted entity for both users and verifiers. Users trust that the extraction will only be performed when the conditions depicted in the access control policy are fulfilled, while verifiers trust that the extraction will always be performed when those conditions are fulfilled. Inspection is useful to prevent user misbehavior by allowing inspection on the grounds of abuse.

### 3 Introduction to Prolog

Prolog is a logic programming language. We will describe briefly the syntax and semantics and the execution of Prolog.

#### 3.1 Data Types

Prolog's single data type is the *term*, which can be an atom, a number, a variable or a compound term.

**Atom.** An atom is a name, i.e., a sequence of characters that Prolog parses as a single unit. Atoms that contain spaces, special characters, or that start with a capital letter must start and end with a single quote. Examples of atoms are

`age`, `'user age'`, `'Age'`

**Number.** A number can be an integer or a float.

**Variable.** A variable is a sequence of characters that starts with a capital letter or with an underscore. A variable acts as a placeholder for arbitrary terms. An underscore `_` represents any variable. Unlike other variables, an underscore does not represent the same value everywhere it occurs within a predicate definition.

**Compound Term.** A compound term consists of an atom followed by a sequence of terms, which is contained in parenthesis and separated by commas. An example of a compound term is

`hasAttributeValue(idcard,age,35).`

Special cases of compound terms are strings and lists. A string is a sequence of characters surrounded by quotes. A list is an ordered collection of terms, which is denoted by `[]` in the case of an empty list and otherwise by square brackets with the terms separated by commas, e.g. `[red,blue,green]`.

#### 3.2 Rules and Facts

Prolog programs describe relations defined by means of clauses. There are two types of clauses: rules and facts. Rules are of the form

`Head :- Body.`



A rule means that **Head** is true if **body** is true. The body consists of calls to predicates, which are called the rule's goals. The built-in predicate `;` denotes conjunction of goals and `!` denotes disjunction. An example of a rule is

```
isGreaterThan(A,B) :- integer(A), integer(B), A > B.
```

A cut `!` inside a rule prevents Prolog from backtracking on the choices it has made. For instance, the evaluation of the rule

```
a(X, X) :- b(X), !, c(X)
```

will output `false` if the first value found for `X` that makes `b(X)` true leads to `c(X)` being false. Rules that have an empty body are called facts. For example, the fact

```
isGreaterThan(age,18).
```

is equivalent to

```
isGreaterThan(age,18) :- true.
```

The built-in predicate `true` is always true.

### 3.3 Execution

Execution of a Prolog program is initiated by posting a query, which consists of a single goal. The Prolog engine tries to refute the negated query. If the negated query can be refuted, it follows that the query, with the appropriate variable bindings in place, is a logical consequence of the program. In that case, all generated variable bindings are reported to the user, and the query is said to have succeeded. Consider the following program as an example.

```
isGreaterThan(A,B) :- integer(A), integer(B), A > B.
hasAttributeValue(idcard,age,35).
hasAttributeValue(passport1,age,16).
hasAttributeValue(passport2,age,19).
satisfiesPolicy(Id) :-
    hasAttributeValue(Id, age, Age),
    isGreaterThan(Age,18).
```

The result of the following query is true.

```
?- satisfiesPolicy(idcard).
true.
```

In a nutshell, the result is obtained as follows. In the program, the only clause-head matching the query is `satisfiesPolicy(Id)`. Therefore, to prove the query Prolog applies the variable binding `Id = idcard` and evaluates the body of the clause. The body consists of a conjunction of two goals. First, Prolog evaluates the leftmost goal, which is `hasAttributeValue(Id, age, Age)` with the binding `Id = idcard`. This goal can only be proven by using the fact `hasAttributeValue(idcard,age,35)`, and thus a variable binding `Age = 35` is created. Then Prolog proceeds to evaluate the second goal, which is `isGreaterThan(Age,18)` with the binding `Age = 35`. There is only one clause-head in the program that matches this goal, which is `isGreaterThan(A,B)`. Prolog creates the bindings `A = 35` and `B = 18` and evaluates the body of the clause. Since that body is also evaluated as true, both goals are proven and the result of the query is true. In this example, there was only one clause that could prove each of the goals. If there was more than one, Prolog would create a choice point and evaluate the first alternative, and would return to the choice point to evaluate the next alternatives when needed.

Following the same evaluation procedure, the result of the following query is false.

```
?- satisfiesPolicy(passport1).  
false.
```

The execution of the following query enumerates all the valid answers, i.e., all the valid bindings for the variable `Id`.

```
?- satisfiesPolicy(Id).  
Id = idcard ;  
Id = passport2.
```

## 4 Program for Credential-Policy Matching

We describe our Prolog program for matching attribute-based credentials with access control policies. In Section 4.1, we depict the representation of the user's credentials. In Section 4.2, we depict the representation of presentation tokens. In Section 4.3, we depict the representation of access control policies.

### 4.1 Representation of a Credential

We show how the information contained in the user's credentials is represented in our Prolog program. First, we describe how the information on the user's secret key and pseudonyms is represented. This information can be associated to one or more of the user's credentials.

**User Secret Key.** The fact that the user possesses a secret key `usk1` is represented by

```
isUserSecret(usk1).
```

**Pseudonyms.** The fact that a pseudonym `nym1` is bound to the secret key `usk1` and was used with a verifier `'verifier1'` is represented by

```
isEstablishedPseudonym(nym1,usk1,'verifier1').
```

For scope-exclusive pseudonyms, the fact that the scope-exclusive pseudonym `senym1` is bound to the secret key `usk1` and was used with `'verifier1'` is represented by

```
isEstablishedScopeExclusivePseudonym(senym1,usk1,'verifier1').
```

The file `user.pl` contains an example of a user credential store. It comprises a user secret key `usk1`, two non-scope-exclusive pseudonyms `nym1` and `nym2` with scopes `'verifier1'` and `'verifier2'`, and a scope-exclusive pseudonym `senym1` for `'verifier1'`.

```
isUserSecret(usk1).
isEstablishedPseudonym(nym1,usk1,'verifier1').
isEstablishedPseudonym(nym2,usk1,'verifier2').
isEstablishedScopeExclusivePseudonym(senym1,usk1,'verifier1').
```

A user credential is described by a credential issuer, possibly a user secret key to which the credential is bound, one or more attributes with their respective attribute values, and some revocation information. A user credential is represented by a set of facts as follows.

**Key Binding.** The fact that the user possesses a credential `idcard` that is bound to the key `usk1` is represented by

```
hasKeyBinding(idcard,usk1).
```

**Credential Issuer.** The fact that the credential `idcard` was issued by the issuer `townhall` is represented by

```
hasIssuer(idcard,townhall).
```

**Attributes.** The fact that the credential `idcard` possesses an attribute `age` with an attribute value `35` is represented by

```
hasAttributeValue(idcard,age,35).
```

**Revocation.** We discuss first issuer-driven revocation. The fact that the credentials issued by the issuer `townhall` are revoked by the revocation authority `townhall_ra` is represented by

```
hasIssuerDrivenRA(townhall,townhall_ra).
```

The fact that, for the revocation authority `townhall_ra`, the current epoch number is 3, is represented by the fact

```
currentRevocationEpoch(townhall_ra,3).
```

The fact that the credential `idcard` is not revoked at epoch 3 is represented by

```
isNotIssRevokedAt(idcard,3).
```

In verifier-driven revocation, the revocation authority revokes certain attribute values. For example, the fact that the attribute values ‘Jane’ and ‘Doe’ are revoked by the revocation authority `hooligans_ra` at the epoch 2 is represented by

```
isVerRevokedAt(['Jane','Doe'], hooligans_ra, 2).
```

The file `user.pl` contains the representation of three credentials for a user Jane Doe: an identity card with age 35 and birth date 28/01/1978, a driving licence for vehicle category C and a passport.

```
/* ID card */
hasIssuer(idcard,townhall).
hasKeyBinding(idcard,usk1).
hasAttributeValue(idcard,firstname,'Jane').
hasAttributeValue(idcard,lastname,'Doe').
hasAttributeValue(idcard,age,35).
hasAttributeValue(idcard,dob,19780128).
hasIssuerDrivenRA(townhall,townhall_ra).
currentRevocationEpoch(townhall_ra,3).
isNotIssRevokedAt(idcard,3).
```

```
/* Driving licence */
hasIssuer(drivinglicense,deptofmotorvehicles).
hasKeyBinding(drivinglicense,usk1).
hasAttributeValue(drivinglicense,first,'Jane').
hasAttributeValue(drivinglicense,last,'Doe').
hasAttributeValue(drivinglicense,vehicle,'C').
hasIssuerDrivenRA(deptofmotorvehicles,deptofmotorvehicles_ra).
currentRevocationEpoch(deptofmotorvehicles_ra,1234).
isNotIssRevokedAt(drivinglicense,1234).
```

```
/* Passport */
hasIssuer(passport,government).
hasKeyBinding(passport,usk1).
hasAttributeValue(passport,firstname,'Jane').
hasAttributeValue(passport,lastname,'Doe').
hasAttributeValue(passport,nationality,'USA').
hasIssuerDrivenRA(government,government_ra).
currentRevocationEpoch(government_ra,698).
isNotIssRevokedAt(passport,698).
```

## 4.2 Representation of a Presentation Token

We describe how the information contained in a presentation token is represented in our Prolog program. The verifier creates a user identifier ‘userid1’ to relate all the pseudonyms and credentials for which the verifier knows that they were shown by the same user. The fact that a pseudonym ‘nym0x00123’ is related to the user ‘userid1’ is represented by

```
userPossesses('userid1', 'nym0x00123').
```

The verifier creates an identifier, e.g. id00123, for each of the credentials shown in a presentation token. The fact that a user ‘userid1’ has shown a presentation token that contains a credential id00123 is represented by

```
userPossesses('userid1', id00123).
```

A presentation token can contain a pseudonym. A pseudonym is derived from a user secret key. The verifier creates a user secret key identifier, e.g. usk00123, to relate all the pseudonyms that are bound to the same user secret key. The fact that a pseudonym ‘nym0x00123’ is related to a user secret key usk00123 and was shown to the verifier ‘verifier1’ is represented by

```
isPseudonym('nym0x00123', usk00123, 'verifier1').
```

A presentation token can involve one or more credentials. For each of the credentials shown in the token, the verifier stores information on the key bound to the credential, the credential issuer and type, and the attributes types and values, as well as revocation information and inspection information.

**Key Binding.** The fact that the credential id00123 is bound to the same user secret key as the pseudonym ‘nym0x00123’ is represented by

```
sameKeyBindingAs(id00123, 'nym0x00123').
```

**Credential issuer and type.** The fact that the credential id00123 is of type idCard and is issued by the issuer townhall is represented by

```
isCredential(id00123,idCard,townhall).
```

**Attributes.** The fact that the credential `id00123` contains an attribute of type `firstname` with attribute value `'Jane'` is represented by

```
hasAttributeValue(id00123, firstname, 'Jane').
```

If the value of the attribute is not revealed, the verifier creates an identifier for the value, e.g. `last00123`. For example, the fact that the credential `id00123` contains an attribute of type `lastname` with an unknown attribute value `last00123` is represented by

```
hasAttributeValue(id00123, lastname, last00123).
```

If the attribute value is not revealed but, instead, a predicate about the value is proven, the predicate is also included in the credential information. For example, the fact that the credential `id00123` contains an attribute of type `age` with an unknown attribute value `age00123` such that `age00123 > 18` is represented by

```
hasAttributeValue(id00123, age, age00123).
isGreaterThan(age00123,18).
```

**Revocation.** For revocation, we employ the same rules and facts described in Section 4.1 for the representation of credentials.

**Inspection.** For the attribute of type `lastname` and unknown value `last00123` described above, the fact that the inspector `inspector2` is able to retrieve the attribute value on the grounds of `'court order'` is represented by

```
isInspectable('ctxt0x0f3d110', inspector2, last00123, 'court order').
```

The file `verifier.pl` contains a representation of a presentation token that shows a credential of type `idCard` and a credential of type `driversLicense`.

```
/* Binding to User */
userPossesses('userid1', 'nym0x00123').
userPossesses('userid1', id00123).
userPossesses('userid1', dl00123).

/* Pseudonym */
isPseudonym('nym0x00123', usk00123, 'verifier1').

/* ID card */
isCredential(id00123,idCard,townhall).
sameKeyBindingAs(id00123, 'nym0x00123').
```

```

isNotIssRevokedAt(id00123,20).
hasAttributeValue(id00123, firstname, 'Jane').
hasAttributeValue(id00123, lastname, last00123).
isInspectable('ctxt0x0f3d110', inspector2, last00123, 'court order').
hasAttributeValue(id00123, age, age00123).
isGreaterThan(age00123,18).

/* Driving Licence */
isCredential(dl00123,driversLicense,deptofmotorvehicles).
sameKeyBindingAs(dl00123, 'nym0x00123').
isNotIssRevokedAt(dl00123,1234).
hasAttributeValue(dl00123,vehicle,'C').

```

### 4.3 Representation of a Policy

We show how a policy is represented in our Prolog program. A policy is represented by a rule in which the body consists of one or more goals. Policies are employed both on the user side and on the verifier side. The user runs a Prolog query that consists of the head of the rule in order to find out whether the user's credentials satisfy the policy and, if that is the case, all the combinations of credentials that the user can employ to compute a presentation token that satisfies the policy are shown. The verifier runs a query that consists of the head of the rule in order to check if the credential information disclosed in the presentation tokens that the user has shown fulfills the policy, and, in that case, the different combinations of credential information that satisfy the policy are shown. We now describe the different elements in a policy and the rules employed to evaluate whether a policy is satisfied on the user side and on the verifier side.

*Pseudonyms.* To check whether there is a pseudonym `Nym` bound to a secret key `Usk` that was used with the verifier `'verifier1'`, a policy employs the goal

```
isPseudonym(Nym, Usk, 'verifier1')
```

To evaluate this goal, the following rules are declared both on the user program and on the verifier program.

```

isPseudonym(.,.,.) :- false.
isPseudonym(Nym, Usk, Scope) :-
    isEstablishedPseudonym(Nym, Usk, Scope).
isPseudonym(Nym, Usk, Scope) :-
    isScopeExclusivePseudonym(Nym, Usk, Scope).
isScopeExclusivePseudonym(SENym, Usk, Scope) :-
    isEstablishedScopeExclusivePseudonym(SENym, Usk, Scope).

```

On the user side, if a pseudonym does not exist, it can be created. Therefore, the following rules are also included.

```
isPseudonym(Nym, Usk, Scope) :-
    ground(Scope),
    isUserSecret(Usk),
    Nym = nymDer(Usk, Scope).

isScopeExclusivePseudonym(SENym, Usk, Scope) :-
    ground(Scope),
    isUserSecret(Usk),
    SENym = seNymDer(Usk, Scope).
```

*Key Binding.* To check whether there is a credential `Id` bound to a secret key `Usk`, a policy employs a goal

```
hasKeyBinding(Id, Usk)
```

To check whether there is a credential `Id` bound to the same key as a pseudonym `Nym`, a policy employs the goal

```
boundToSameKey(Id, Nym)
```

This goal can also be employed on input two credentials or two pseudonyms. To evaluate this goal, the following rules are included in the programs.

```
boundToSameKey(X, Y) :-
    (sameKeyBindingAs(X, Y); sameKeyBindingAs(Y, X)),
    (isPseudonym(X, -, -); isCredential(X, -, -)),
    (isPseudonym(Y, -, -); isCredential(Y, -, -)).

boundToSameKey(X, Z) :-
    sameKeyBindingAs(X, Y), boundToSameKey(Y, Z).
```

*Credential issuer and type.* To check whether there is a credential `Id` issued by the issuer `townhall`, a policy employs the goal

```
hasIssuer(Id, townhall)
```

More generally, to check whether there is a valid credential `Id` of type `idCard` issued by the issuer `townhall`, a policy employs the goal

```
isValidCredential(Id, idCard, townhall)
```



This goal can also include an epoch number, e.g.

```
isValidCredential(Id, idCard, townhall, 19)
```

To evaluate this goal, the following rules are included in the programs.

```
isValidCredential(C,T,I) :-  
    isCredential(C,T,I), isNotIssRevoked(C).  
isValidCredential(C,T,I,Epoch) :-  
    isCredential(C,T,I), isNotIssRevokedAt(C,Epoch).
```

As can be seen, this goal also checks that the credential is not revoked. We describe the rules for revocation below.

*Attributes.* To check whether there is a credential `Id` that contains an attribute of type `firstname` with a value `First`, a policy employs the goal

```
hasAttributeValue(Id, firstname, First)
```

A policy can restrict the possible values of an attribute by including one or more goals about the attribute value. For example, a policy can include the goals

```
hasAttributeValue(Id, dob, Dob),  
isLessThan(Dob, 19950320)
```

to restrict the attribute value `Dob`. To evaluate this goal, the programs include the following rules.

```
isLessThan(A,B) :-  
    integer(A), integer(B), A < B.  
isLessThan(A,B) :-  
    integer(B), isLessThan(A,C), integer(C), C < B.
```

*Revocation.* To check whether there is a credential `Id` that is not revoked by the issuer, a policy employs a goal

```
isNotIssRevoked(Id)
```

To evaluate this goal, the following rules are included in the programs.

```

isNotIssRevoked(Cred) :-
    isCredential(Cred,_,Iss),
    hasIssuerDrivenRA(Iss,RA),
    currentRevocationEpoch(RA,CurrEpoch),
    isNotIssRevokedAt(Cred,EvidenceEpoch),
    integer(CurrEpoch),
    integer(EvidenceEpoch),
    CurrEpoch =< EvidenceEpoch, !.

isNotIssRevokedAt(Cred,Epoch) :-
    integer(Epoch),
    isNotIssRevokedAt(Cred,EvidenceEpoch),
    integer(EvidenceEpoch),
    Epoch =< EvidenceEpoch, !.

```

In verifier-driven revocation, the verifier revokes some attribute values for attributes of certain types. For example, a policy contains the following goals in order to check if there is a credential `Id` with attributes of types `firstname` and `lastname` such that the attribute values `First` and `Last` are not revoked by the revocation authority `hooligans.ra` designated by the verifier.

```

hasAttributeValue(Id, firstname, First),
hasAttributeValue(Id, lastname, Last),
isNotVerRevoked([First,Last], hooligans.ra).

```

To evaluate those goals, the following rules are included in the programs.

```

isVerRevokedAt([],_,_) :- false.
isNotVerRevokedAt(AttList, RA, Epoch) :-
    integer(Epoch),
    not((isVerRevokedAt(AttList,RA,RevEpoch), RevEpoch =< Epoch)), !.

isNotVerRevoked(AttList, RA) :-
    currentRevocationEpoch(RA,Epoch),
    isNotVerRevokedAt(AttList, RA, Epoch).

```

*Inspection.* To check whether there is a credential `Id` of type `lastname` with an attribute value `Last`, such that `Last` can be inspected by the inspector `inspector1` on the grounds of ‘`court order`’, a policy contains the following goals.

```
hasAttributeValue(Id, lastname, Last),
isInspectable(Ctxt, inspector1, Last, 'court order')
```

To evaluate those goals, the following rules are included in the programs.

```
isInspectable(Ctxt, Pk, AttList, Grounds) :-
    ground(Pk),
    ground(Grounds),
    Ctxt = vfEncrypt(Pk, AttList, Grounds).
```

The file `user.pl` contains an example of a policy. The policy requires the user to produce a presentation token with a pseudonym `Nym` and two credentials `Id` and `Dl`. The goals of the policy require the pseudonym `Nym` and the credentials `Id` and `Dl` to be bound to the same user secret key `Usk`. The credential `Id` must be issued by the issuer `townhall` and must not be revoked by the issuer. It must also possess attributes of types `firstname`, `lastname` and `dob`. The inspectors `inspector1` and `inspector2` must be able to inspect the value of the attribute of type `lastname`. The value of the attribute of type `dob` must be less than the value `TodayMinus18Years`. The credential `Dl` must be issued by the issuer `deptofmotorvehicles`, must not be revoked by the issuer and must possess an attribute of type `vehicle` whose value must be `'C'`.

```
satisfiesPolicy1(Nym, Id, Dl, Ctxt, First) :-
    isPseudonym(Nym, Usk, 'verifier1'),
    hasKeyBinding(Id, Usk),
    hasIssuer(Id, townhall),
    isNotIssRevoked(Id),
    hasAttributeValue(Id, firstname, First),
    hasAttributeValue(Id, lastname, Last),
    (
        isInspectable(Ctxt, inspector1, Last, 'court order');
        isInspectable(Ctxt, inspector2, Last, 'court order')
    ),
    hasAttributeValue(Id, dob, Dob),
    isLessThan(Dob, TodayMinus18Years),
    hasKeyBinding(Dl, Usk),
    hasIssuer(Dl, deptofmotorvehicles),
    isNotIssRevoked(Dl),
    hasAttributeValue(Dl, vehicle, 'C').
```

A query `?- satisfiesPolicy1(Nym, Id, Dl, Ctxt, First)` lists all the valid variable bindings for the variables `(Nym, Id, Dl, Ctxt, First)`. As an

example, we show the result of running this query, which reports of the variable bindings found for the pseudonym and credential store declared in the file `user.pl`, which is described in Section 4.1.

```
?- satisfiesPolicy1(Nym, Id, Dl, Ctxt, First).
  Nym = nym1,
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector1, 'Doe', 'court order'),
  First = 'Jane' ;

  Nym = nym1,
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector2, 'Doe', 'court order'),
  First = 'Jane' ;

  Nym = nymDer(usk1, verifier1),
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector1, 'Doe', 'court order'),
  First = 'Jane' ;

  Nym = nymDer(usk1, verifier1),
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector2, 'Doe', 'court order'),
  First = 'Jane' ;

  Nym = senym1,
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector1, 'Doe', 'court order'),
  First = 'Jane' ;

  Nym = senym1,
  Id = idcard,
  Dl = drivinglicense,
  Ctxt = vfEncrypt(inspector2, 'Doe', 'court order'),
  First = 'Jane' ;
```

## 5 Conclusion and Future Work

We have proposed a Prolog program for credential-policy matching that lists all the credential and pseudonym subsets that a user can employ to satisfy a policy. On the verifier side, our program allows the verifier to check whether the credential information already disclosed by the user fulfills the policy to access a new service. Our approach simply requires to represent the credential information as facts and the policies as rules. Therefore, our program can easily be extended to incorporate new features of an attribute-based credential system. For instance, new credential and attribute types can simply be added by representing them as facts, or new restrictions on attribute values can be added by declaring rules that evaluate those restrictions. As future work, we need to integrate our Prolog program into a full-fledged implementation of an attribute-based credential system. For this purpose, a tool that takes in the credential and policy representations employed by the chosen implementation and outputs Prolog representations is required.

## References

1. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology-CRYPTO 2009*, pages 108–125. Springer, 2009.
2. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *Theory of Cryptography*, pages 356–374. Springer, 2008.
3. Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R Enderlein, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Janus Dam Nielsen, Christian Paquin, Franz-Stefan Preiss, et al. H2. 2-abc4trust architecture for developers. *ABC4Trust heartbeat H*, 2:2, 2013.
4. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 345–356. ACM, 2008.
5. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography-PKC 2009*, pages 481–500. Springer, 2009.
6. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in CryptologyEUROCRYPT 2001*, pages 93–118. Springer, 2001.
7. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in CryptologyCRYPTO 2002*, pages 61–76. Springer, 2002.
8. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology-CRYPTO 2004*, pages 56–72. Springer, 2004.
9. Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. Analysis of revocation strategies for anonymous idemix credentials. In *Communications and Multimedia Security*, pages 3–17. Springer, 2011.