

Relying Party Credentials Framework

Amir Herzberg¹ and Yosi Mass²

¹ NewGenPay Inc., <http://www.ngPay.com>; part of work done while with IBM HRL

² IBM Haifa Research Lab, <http://www.hrl.il.ibm.com>, yosimass@il.ibm.com

Abstract. We present architecture for e-business applications that receive requests from a party over the Net, to allow the applications to make decisions relying on the credentials of the requesting party. Relying party applications will be provided with uniform interface to the credentials of the requesting party. This will allow use of simple, widely available credentials as well as more advanced credentials such as public key certificates, attribute certificates and 'Negative' credentials such as certificate revocation lists (CRL). The core of the architecture is a Credential Manager who will provide all credential management functions, including collection of credentials, providing uniform interface to credentials, and extracting semantics relevant to the relying party's applications.

1 Introduction

Credentials are critical for secure business and commerce between entities. A credential is a statement by an *issuer* on some *properties* of the *subject* of the credential. The subject normally presents credentials to a *relying party*. The relying party needs to make a (business) decision based on the credentials, typically whether to allow a request or access. Current e-business systems do not have a separate module for managing credentials. Many systems use only very simple forms of credentials, such as user-id/password identification, and subsequent lookup in local membership database. However, it is well recognized that advanced credentials such as public key and attribute certificates are essential for e-business, and indeed these are used by some systems. We propose an architecture and framework for credentials management, that may help to extend the use of credentials for e-business, and in particular, support advanced credentials such as public key and attribute certificates.

We notice that in the recent years there have been a large number of works on trust management by relying parties, most notably PolicyMaker [2], KeyNote [1]. These works propose improved certificate formats, and policy based tools for the relying party to make decisions based on the certificates. The policies and tools are very broad in their scope, and allow pretty complex decisions as a pretty complex function of the available certificates. We suggest a much more piece-meal approach, where the mechanisms in this work will only collect credentials (including certificates) and map them to interface and semantics known to the relying applications. The (small) step we provide is often needed: the collection is needed whenever some credentials are not available immediately with the request (e.g. SSL passes just one certificate chain), and the mapping is needed whenever different issuers may use different styles (e.g. there are multiple potential locations for e-mail address, even for PKIX [11] certificates). Almost all of the 'interesting' logic of trust management should properly be done after

the more 'mechanic' steps of collection and mapping of credentials are complete. Therefore, our work can be used to remove substantial complexity from the trust management application and policy. This modular approach of simplifying trust management by looking at specific sub-problems continues the work of [8,17] where trust management is simplified into mapping from credentials to roles.

In order to simplify and focus on a specific module, our work does not address the actual authentication and identification of the requesting party. Notice that in any implementation, the relying party also needs to verify the identity of the requesting party (and that this party is the owner of the credentials), e.g. by user-id, password, e-mail verification, checking a digital signature on the request or using authentication protocol such as SSL. This verification would be done by a separate module and is outside the scope of this paper.

There are many forms of credentials. It is instructive to first consider some physical credentials, e.g.:

- Passport, visa, id-card, driver license, other documents
- Charge/bank cards, employee card, membership card, ...
- Professional and other licenses and certificates

Our focus is on digital credentials, which may be passed over the Net. Specifically:

- Identity public key certificates, signed by a Certificate Authority (CA). Links the subject name (or other identifier) with a specific public key, and possibly some other properties (e.g. in extension fields). For X.509 certificates, subject name is specified in the distinguished name field or in the alt-name extension.
- Non-identity public key certificates. These are certificates, which do not include an identifier, but only a public key (or hash of it) and properties of the owner of the private key corresponding to the public key. One reason for not including an explicit identifier is to preserve anonymity of the subject (e.g. for group signatures). Or, the issuer may not know the identity or prefer not to include it, e.g. to avoid liability.
- Attribute certificates are a signed message linking an identifier with some properties. An attribute certificate normally would not contain the public key of the subject (or hash of it), therefore another method should be used to validate the identifier corresponds with the subject making a specific request. Typically, an attribute certificate contains the identity of a CA and the serial number of a public key certificate issued by that CA to the subject.
- Digitally signed or otherwise authenticated documents, e.g. PICS rating [13] or an entry from a database (e.g. Duns and Bradstreet record). An especially simple and common case is the use of the entry from a local (membership) database as a credential of an entity, after this entity was (locally) authenticated using user-id and password.

The existence of this large number of potential credential forms and sources, results in difficulties in managing and making decisions based on credentials. As a result, the relying party has to use manual mechanisms and processes, or – if an automated application is attempted – limit itself to few types of credentials and to very simple policies and suffer substantial complexity.

We present an architecture and framework for the internal design of an e-business party which relies on credentials. Our goal is to simplify the task of creating automated credential relying applications. Such a framework may complement the large amount of existing works on credentials, which mostly focused on the issuer side (rather than relying party side), and on public key and attribute certificates (which we consider specific kinds of credentials – albeit with special importance).

The main services we hope the framework can provide, to multiple relying party applications, are:

- Mapping multiple formats of credentials into a simple common format and interface
- Simplified interface to complex credentials
- Extraction, from credentials, of the semantics relevant and understood by the relying party applications
- Credential management, including acquisition, storage, updates and revocation checking

This paper will present only high-level architecture of the framework and its components. We expect substantial follow up work, by us as well as by others in the security community, in order to transform this high level architecture into practical, widely accepted and standardized framework.

1.1 Identities in credentials

The discussion above glossed over (at least one) basic problem: what are the identities in a credential (or certificate), and what are the functions of the identities in management of credential relying applications. The complexity of this problem is reflected by the controversy regarding it, in particular with respect to public key certificates (see `related works` section below). It may help to consider first the situation with physical credentials such as passports, identity cards, and other forms of `paper and plastic` credentials and certificates. Such credentials are typically used to grant some permission to a physical person holding the credential. In many of these credentials, a picture, signature, or another means of direct authentication identifies the person. We call such means of direct authentication a *direct subject identifier*. In other physical credentials, authentication is done indirectly. For example, the credential may contain the name of the holder, which may need to present another credential proving his name (with direct authentication e.g. a picture). The name in the two credentials is serving different purposes: as *identifying property* being authenticated (in an identity card, e.g. with a picture); or as an *indirect subject identifier* allowing linkage from one credential (e.g. without picture) to another (with a picture).

Subject identifiers should have well defined interpretations. Namely, we assume that each issuer uses a known, well-defined set of subject identifiers; and the subject field of a credential will contain only subject identifiers from this set. Typical subject identifiers would be a name, an identity number (e.g. SSN), a URL, an e-mail address, a user-id in a given server, a certificate number in a given CA, a picture, or a (hash of) public key. Only the last two – picture or (hash of) public key - are direct subject identifiers, i.e. allow direct authentication.

Clearly, it is much simpler to use one credential with a direct subject identifier directly linked to properties, rather than use two credentials: one for providing the

properties with an indirect subject identifier, the other with direct subject identifier, specifying the same subject identifier as an identifying property. Why, then, are credentials often using indirect identifiers, sometimes in addition to direct identifiers? Here are some reasons:

- **Issuing costs** involves the cost of identification, concern about liability (if others may rely on a false identification), and cost of the identifier itself. For a physical credential such as a credit card, the cost of the identifier is that of embedding a picture or smartcard. For a digital credential, the identifier cost may be of performing the public key signature operation – merely few dozens milliseconds on today’s workstations; the identification and liability concerns are more relevant.
- **Relying costs** involve the cost of verifying identity using a direct subject identifier. These costs are often negligible, e.g. for manual verification (using picture, signature etc.) or for identifying using public key certificate (a few milliseconds of CPU). However for some identifiers, e.g. smartcard or fingerprint, the cost of the verification (hardware) may be substantial.
- **Counterfeiting** may be possible for some identifiers, such as a picture. Counterfeiting may be done either by replacing or modifying the identifier, or by simply fooling the relying party who fails to distinguish between the identified subject and a similar other entity. Counterfeiting a digital certificate is difficult or infeasible, if a secure cryptographic signature algorithm is used with sufficient key length.
- **Reliability of identification** is a concern when the relying party may fail to identify the subject, for example using an outdated picture. This reason seems relevant only to physical identifiers.
- **Role-based credential** is a credential that is given to any entity (person) which has a certain role assigned to it (possibly by another issuer). The use of role-based physical credentials is rare (or at least we do not have a good example). However, they may be useful for digital credentials, e.g. to provide some privileges to all members of a role or group.
- **Remote credential properties** – in some cases, the relying application may be separate from a server that keeps updated record of the properties of the subject. Since the properties may change, it is not possible or desirable to put them in the credential. Therefore, the credential will contain an identifying property, which will be used to link to another credential where it will be an indirect identifier. For example, a passport contains a picture, but also a number; authorities will normally identify a person using the picture, and then use the number to look up an online database containing e.g. suspects listed by passport numbers.

1.2 Credential properties and types

In simple scenarios, credential issuers are closely coordinated with credential relying applications, and the applications can use the credential directly. This may be achieved by standardizing the credentials. This is done in the PKIX standards for public key certificates [11], which define exact certificate format with exact encoding (both based on [18]) and specific ways for encoding identifiers and properties in the certificates. We comment that the usage of PKIX is still quite complex, in particular since there are

multiple ways to specify properties – as fields (e.g. validity and issuer and subject names), attributes of subject name, extensions, privileges within extensions, and more. Furthermore, even [11] itself sometimes permit the same property to be specified in multiple locations. For example, the e-mail address of the subject may be specified as attribute `EmailAddress` of the subject distinguished name, although the standard specifies that in new implementations the e-mail address must be specified as `rfc822Name` in the subject alternative name field. As a result, even when the credential issuers and credential relying applications use the same exact standard for credentials, there may be substantial complexity in processing the properties in the credential due to potential ambiguity and alternative mappings, as well as to having some properties as fields, some as attributes, some as extensions, and so on. Our framework will simplify this, by providing the mapping function as an external service to the relying applications.

In many realistic scenarios, a credential relying application may need to be able to handle credentials from multiple issuers – possibly even for the same subject. The different issuers may use slightly or dramatically different credential formats. Consider even a very basic case of two issuers using PKIX X.509 certificates, but with different private extensions, usage of options, or semantic meanings as defined in the Certificates Policy Statement (CPS) [10]. It is quite possible that the two certificates actually carry the same semantic properties, however they are encoded slightly differently. We say that the two certificates are of different *type*¹. A credential type identifies a particular set of properties as well as their precise semantic meanings. The credential framework provides a general mechanism for mapping between compatible credential types. Even if used simply to implement the CPS mappings defined by the PIKS and X.509 standards, this will already remove complexity from the relying applications.

In order to identify which mapping should be used, it is easier if the credential type is known. We consider credentials with an explicitly known type, and credentials where the type is not known. When the type is not known, the framework will attempt to identify the type; afterwards, it will use mappings among identified credential types. The framework will also provide the type identifier to the relying application in a standard way, which will make it easier for the application to use multiple credential types.

The use of credentials from multiple potential issuers, for the same subject or for different subjects, may be further complicated if the credentials may use different formats. As mentioned above, credentials may be, in addition to public key certificates, also attribute certificates, revocations, or other credentials such as a [13] rating or a record from a database (e.g. the Dun and Bradstreet record returned by Eccelerate [3]). Furthermore, there are multiple formats for public key certificates, ranging from different X.509 extensions, to completely different certificate formats such as PGP, SPKI, PolicyMaker and KeyNote. The framework maps different credential formats to one common format, specifically the Credential Markup Language (CML).

¹ The term ‘profile’ is also sometimes used for this purpose, however we prefer the term ‘type’ as ‘profile’ is also used for other certificate – related purposes.

1.3 Related Works

The most well-known and deployed approach to public key infrastructure is the X.509 standard [18], recently revised and extended [19]. The X.509 approach focus on identity based public key certificates. It assumes a universal convention for selecting *distinguished names (DN)*, which are unique identifiers based on the subject name. The distinguished names consist of several components or attributes, one of them being the common name – which would typically be the first, middle and last name of the subject (typically, X.509 would assume that subjects are persons). The other components of the distinguished name should provide the uniqueness, as common names are clearly not unique. Notice that this requires careful selection of the other identifiers in the distinguished name, and in fact in many implementations some of the common name entries had to be artificially modified to ensure that the distinguished name will be unique, resulting in common names like John Smith1 (actual example from IBM).

A bigger problem with the traditional X.509 approach results from the implicit requirement that a certificate issuer is responsible for correct identification, with potential liability for damages from wrong identification [5]. This became a concern, and indeed many companies refrained from issuing certificates (e.g. to employees). Attribute certificates [19,6,12] provides a mechanism to provide a credential by referring to a public key certificate, thereby allowing a company to at least issue a credential (attribute certificate), using public key certificate issued by some other CA. It is also possible to use X.509 certificate format without putting a real name, as in [23].

More recent works, in particular [15,4], suggested that names should only be unique with respect to a given issuer, and do not necessarily have to have global meaning (and therefore liability). In fact, in this approach the name field in a certificate becomes just a convenience and an option, and the subject is really identified by possessing the private key corresponding to the public key in the certificate. Namely, these works capture the separation between the use of the name as an identifier and its use as just a simple property (used e.g. for addressing the user, but not assumed to be unique) – a notion that we adopt and extend.

Another problem with traditional X.509 approach lies with the implicit assumption that is a hierarchy of certificate (and attribute) authorities, and relying parties know and trust the root CA of this hierarchy. A very different approach is taken by PGP [22], where certificates define a `web of trust` and there is no central CA. We share the view advocated by [22,4,5,9,15,1,2,14,7,17,8,12], namely a relying party may not completely trust the issuers of the credentials. Instead, these works advocate a model where the relying application may need multiple credentials to make its decisions, and has a non-trivial policy for the necessary credentials. Our work is a follow-up to our work in [17,8], which developed a tool to determine if a subject has the right set of credentials (from properly trusted issuers) according to a given policy.

2 Relying Party Credential Management Architecture

We propose that the relying party use architecture as illustrated in Figure 1 below. The core module is the Credential Manager. The manager receives requests for resolving credentials, with an identifier and often with an initial credential – an attribute certificate or a public key certificate. The initial credential is typically

received from with some request (e.g. connection). The Credential Manager is not concerned with validating that the requestor has the right to the credential – that should be validated thru independent means, such as SSL authentication (for a public key certificate).

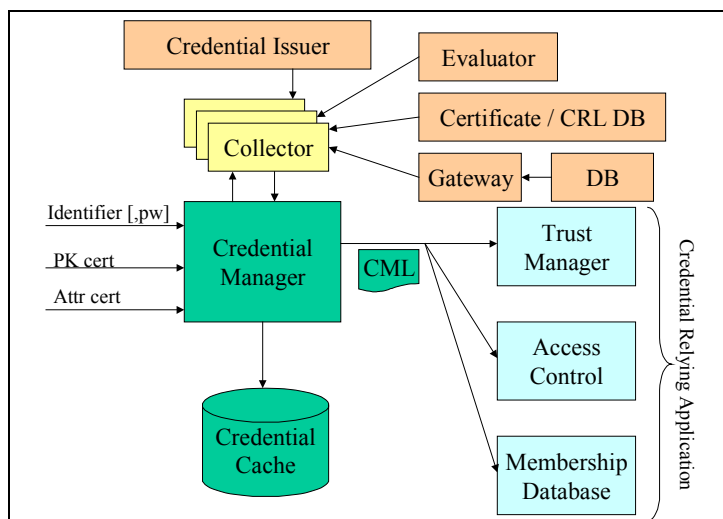


Fig. 1. Relying Party Credential Management Framework

The Credential Manager performs several functions:

- **Translates credentials into the common interface, e.g. Credentials Markup Language.** This allows credential relying applications to be oblivious to the specific format and even method of a credential. For example, the logic in an e-marketplace application which deals with a membership request may only care if the subject is an employee of a member company, but not if this information was received by a public key certificate, by an attribute certificate, by e-mail to the company or by a direct query. We note that the Credentials Markup Language is simply a convenient common interface for the credentials (notice it is not a certificate – in fact it is not even signed).
- **Collects additional relevant credentials.** In many cases, the subject may not present all of his credentials together with the request. In particular, in the typical case when the subject is authenticated using SSL or TLS client authentication, then only one certificate (chain) is sent from the client (subject) to the server. Furthermore, there may be credentials that the subject may not even be aware of (e.g. review by a referee trusted by the relying party), or 'negative' credentials that the subject may not willingly disclose. The credential manager will automatically collect such missing credentials as described below.
- **Checks for revocations of credentials.**
- **Caches credentials to speed up processing.** To improve efficiency, the credential manager may store received credentials, checking for updates.

- **Collects credentials for issuers.** It is possible that a credential is issued by an entity that is not known sufficiently, and the relying party may need to collect credentials also for that entity. The credential manager may be asked to automatically collect credentials for any such unknown issuer. Alternatively, the credential manager may return the credential and leave it to the relying application to decide whether to request the credential manager to collect credentials for the unknown issuer.

In a typical deployment, a request is received at the (web) server from a client. If SSL client authentication is used, the server will receive a certificate from the client², otherwise the server may receive some other identifier for the client (e.g. user name and/or e-mail address³), and potentially a password. The Credential Manager is called with the received credential (certificate / identifier / password). Notice that the Credential Manager may also be called directly by the credential relying application.

2.1 Credential Collector Agents

To collect credentials, the Credential Manager will contact one of potentially several *credential collector agents*, or simply *collectors*. The Credential Manager will select which collector(s) to use based on its policy, the calling application (and parameters passed by it), the available credential(s) for the subject, and the installed collectors.

Collectors may use different mechanisms appropriate to collect different kinds of credentials, from different sources. Some of these are:

- Collectors are likely to request public key and attribute certificates from a repository identified by the subject and given in the request to the Credential Manager, and / or from predefined central repositories.
- Collectors may try to collect credentials from central repositories, which keep credentials for many entities. This approach is needed whenever the subject may be unaware of the credential, and even more if the credential is `negative` - such as a certificate revocation list or an unfavorable product review. As described in [14], this approach is also useful to find a chain of certificates from these trusted by the relying party to the subject.
- Collectors may try to collect credentials using general-purpose search engines, for credentials that will have well defined and accepted format. This is particularly useful for negative credentials.
- Collectors may use predefined gateways to provide credentials. A typical role for a gateway may be to provide interface to a database, using a public key certificate that includes an identifier of an entry in the database. In this case, the client is authenticated using a public key certificate (e.g. using SSL/TLS). Then, the certificate, or just the identifier, is sent (by the collector) to the gateway. The

² Many web servers will reject such a request if they do not have the public key of the issuer of that certificate. To allow users to receive certificates also from unknown issuers, we recommend that a fixed (public, private) key pair be published so that the initial certificate may be signed using this key. Servers will be installed with this public key, so they do not reject certificates signed by it. This workaround has overhead – the initial certificate is never trusted – but is important for allowing use of some web servers.

³ Notice an e-mail address may be weakly-authenticated by the server sending it a challenge and receiving a response, before calling the credentials manager.

- gateway performs an appropriate query on a database, using the identifier, and returns the record. This mechanism is used by *eccelerate.com* to provide subject records from Dun & Bradstreet's database [3].
- Collectors may contact a server for the subject using secure mechanisms, providing the subject identifier and optionally a password, and receive back the credentials of that subject. The server of the subject will use the password, if provided, to authenticate the request; clearly this is a low security mechanisms as the relying party is trusted to maintain security of this password. A `use once` password is also possible, which will be used only for the specific transaction, and communicated securely between the subject and her server. A `use once` password may be sent by the collector or received by the collector from the server, in which case it is returned to the relying party application (which should then use it to authenticate the request). Another low-security but easy to implement solution is to use an e-mail address as the identifier; the e-mail address can be validated by a challenge-response exchange prior to calling the Credentials Manager.

The output of the Credential Manager is provided using standard interface, possibly as a file specified using Credential Markup Language (CML), an XML format that separates between the properties being attributed to the subject and other information related to the credential. Each application can specify in advance a certain type (or types) of credentials that it knows to understand, and the Credential Manager will attempt to provide these types of credentials, using pre-defined mappings between different credential types. This allows an application to handle only one (or few) credential types, with automated mappings from the potentially many different formats and types of credentials issued by different organizations.

2.2 Credentials Framework

The Credentials Framework is the central component of the Credential Manager. The framework receives credentials with different formats and types, and makes them all available via a common, simplified interface, also mapping them to the types known to the credential relying application. See Figure 2.

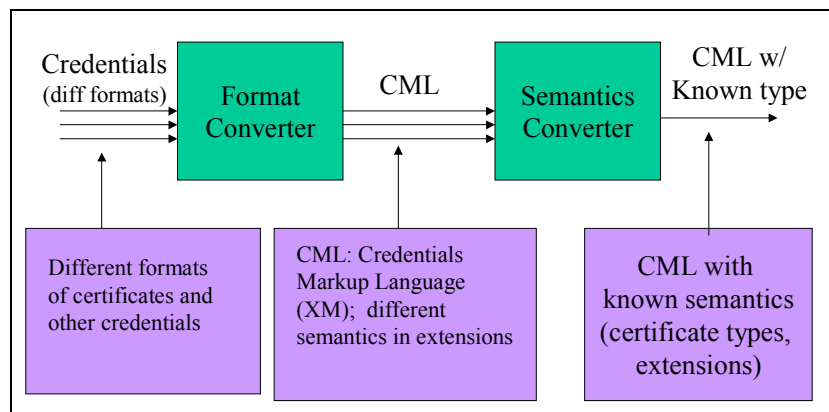


Fig. 2. Credentials Framework – High Level View

The credential framework consists of two software modules, the format converter and the semantic converter. The format converter receives credentials in different formats, and converts them to a common interface format – the credential markup language (CML). This is a pretty ‘mechanical’ transformation; in our approach, there is very limited significance to the particular format, and for example in [8,17] we used X.509 certificates although we do not follow the distinguished name or the root CA notions. The Semantic converter focuses on extracting the properties and credential type meaningful to the relying application. In the next sections we describe the Credential Markup Language (CML), the format converter and the semantic converter.

3 Credential Markup Language (CML)

The Credential Markup Language (CML) is designed to capture all types of credentials into a common format. Notice this is not a certificate format; CML is simply a common format to allow multiple applications of the relying party to handle credentials (in particular, it simplifies the semantic converter plug-ins presented in section 5). We now present some ideas on a potential design of CML. We used XML, which is convenient as there are tools to access and manipulate XML objects. More work is required to decide on the best format (XML or otherwise). CML has two parts:

1. A header which is common to all credential types.
2. A body which is credential type specific.

3.1 CML header

The CML header contains the following fields:

- **Issuer** – the issuer of the credential. The field should identify the issuer so that the relying party can then verify that the issuer is really the one who issued the credential, and can determine whether (and how much) it is trusted. Possible options for the issuer are public key (or hash of it), name, e-mail or a URL. The issuer field would also include a handle (pointer) allowing the credential relying application to

- query the Credentials Manager (or DB directly) for credentials of the issuer, to determine whether it can be trusted.
- **Subject** – information relevant to identifying the subject of the credential. The subject field should include one or more identifiers for the subject. We describe it in details below.
 - **Security** – describes the security type and level of this credential. It can be either *signed* or *authenticated*. Example of a signed credential is Public Key/Attribute certificate or a signed XML document. Example of an authenticated credential is an XML document that was retrieved through a secure channel from some data repository.
 - **Type** – A field that describes the type of the credential. The credential type defines which data is expected to appear in the credential body. Example types can be *Company rating* or a *SET* [23] credential. This field may have the value *Unknown* in the case that the credential does not have a predefined type, or its type was not identified.
 - **Capture and conversions history** – identifies the procedure(s) used to capture the credential, i.e. how it was received (e.g. certificate received from web server using SSL, or query against a database). Also lists the procedures used to convert the credential from its initial type to its present type. The capture and conversion history field may be used, for example, to avoid an unreliable conversion.
 - **Validity period** – dates and times during which the credential is valid (typically, a beginning and end date for validity, or only an end/expiration date).

- We now elaborate on the subject field. The subject field should include one or more identifiers for the subject. There are several types of identifiers: *direct*, *indirect*, *composite*, or *informational*. If a subject contains multiple identifiers, then identification is achieved when any of them, except informational, matches.
- Direct identifiers may be directly used to identify the subject, e.g. a public key (or hash of it), a picture or other biometrics, or an e-mail address (allowing weak identification).
 - Indirect identifiers - some credentials, e.g. an attribute certificate, may not contain any direct identifier, but contain only indirect identifiers, such as a certificate issuer and serial number, distinguished name, URL or role. An indirect identifier cannot be used directly to identify the subject. Instead, the subject will be identified using some other means or credentials, which will establish it as having this indirect identifier. An indirect identifier should usually specify acceptable ways to establish the identity, typically by listing specific certification authorities. If this is unspecified, it is up to the relying party to resolve the indirect identification properly.
 - Composite identifier is a list of several identifiers, requiring that all of the identifiers in the list are matched for identification to be confirmed. For example, an electronic passport device may require that the user possesses a secret code as well as pass biometrical test.
 - An informational identifier is provided only as hint, and is not necessary or sufficient for identification.

An identifier is often also a property, which means that if the credential is properly authenticated, then the issuer provides this as a potential (indirect) identifier for this subject.

We now give some examples of the subject field. We begin with the subject field for a typical X.509 public key certificate. Many issuers, as in our example, will consider the DistinguishedName field as only informational identifying property, namely they allow only identification by the public key – not by DistinguishedName issued by another CA.

```
<Subject>
<identifier type=informational property=yes>
  <Name type= DistinguishedName><CN>IBM</CN><ORG>IBM</ORG></Name>
</identifier>
<identifier type=direct>
  <PublicKeyInfo>
    <Algorithm> "RSA/512/F4" </Algorithm>
    <PublicKey encoding="base64">MEgCQQCRWa71T1fcnWxYJ6NzIXqpeYJnfUsJgfTXp2s1Rcb
  </PublicKey>
  </PublicKeyInfo>
</identifier>
</Subject>
```

We now give an example of the subject for a typical X.509v4 attribute certificate. In [19], there are three ways to identify the subject (referred to as Holder). A typical way is to use baseCertificateID, which specifies the issuer and the certificate serial number. Another way is to give the name of the entity, but [19] notes that this introduces complexities, e.g., which names can be used and who would be responsible for assigning the names. Therefore, if both a name and certificate ID are used, the name becomes informational, as in the example we give.

```
<Subject>
<identifier type=informational property=yes>
  <Name type= rfc822Name>foo@fee.com</Name>
</identifier>
<identifier type=direct>
  <baseCertificateID>
    <Issuer><Name type=DistinguishedName><CN>IBM</CN><ORG>IBM</ORG></Name>
    </Issuer>
    <serial>387857</serial>
  </baseCertificateID>
</identifier>
</Subject>
```

3.2 CML Body

The CML Body contains the fields (extensions, attributes, etc.) from the credential, each mapped in a well-defined way using XML tags, in a way which allows uniform handling by any application (or `plug-in`) that knows the relevant credential type. Types may be general (e.g. X.509 certificate, PICS rating) or specific (PKIX compliant identify certificate, BBB rating record).

4 The Format Converter

The Format Converter (FC) is a module that accepts different credential formats and converts them into a common interface, e.g. to the CML format. In the suggested framework, there will be converters from different credential formats into CML and the framework can be extended by new converter plug-ins that can be added to it. Figure 3 illustrates the format converter architecture.

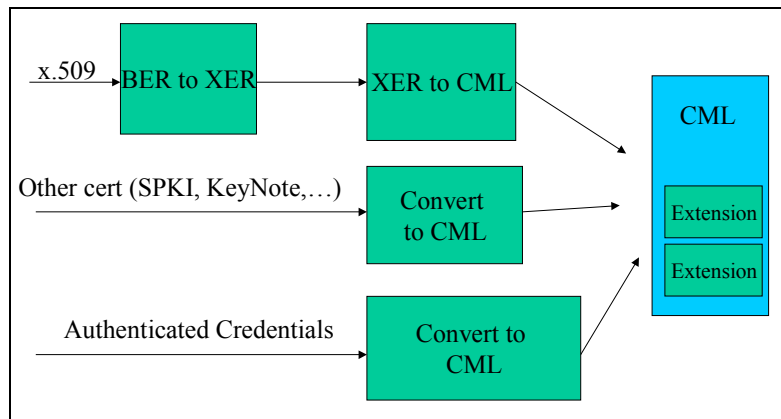


Fig. 3. Format converter architecture

When a new credential is given as input to the FC, the first step of the FC is to decide on the format of the given input and decide which converter to apply on it. This may be done by trying to apply each converter on the given input and the one that matches the format will do the work.

For example consider a BER (Basic Encoding Rule) encoded x509 certificate which is given as input to the format converter. BER encoding is the standard encoding for ASN.1 (Abstract Syntax Notation One), the syntax for X509v3 certificates.

We use the BER to XML translator of the IBM Tokyo Research Lab which is part of their XML Security Suite [20]. The output of the BER to XML translator is an XER (XML Encoding Rule) format, which describes the X.509 certificate in XML. The next converter that is applied is the XER to CML converter that creates the CML format by copying *the issuer, subject & validity* fields from the XER format and creating the CML *Security* tag with value type=signed and with the details of the signing algorithm as extracted from the XER format.

We believe that other converters to CML will appear for other formats e.g. for SPKI. Notice that authorizations in SKPI certificate (as well as some other formats) will be mapped to properties in the body of the credential.

We will also have converters for XML documents extracted from databases. Such converters will get as input the issuer, the subject and the XML document. A simple such converter will create the CML header with the given *issuer & subject*, with *Security = authenticated*, *type = unknown* (or a specific type if identified) and will leave the body as the original document.

5 The Semantic Converter

The last conversion step in the credential framework (see figure 2 above) is the Semantic Converter. The purpose of this conversion is to convert a CML credential into another CML credential in which the fields are now understandable by the relying party.

We give now some semantic converter examples. The first example is a user's e-mail address. In an X509v3 certificate this field may appear in the subjectAltName field as an rfc822Name while in another XML signed document it may appear under some EMAIL tag. Without the semantic converter an application that wants to use the e-mail address will have to understand various credential formats. With the semantic conversion, the e-mail address appears in a fixed field, hence it easy to be used by an application.

Another example is an AC (Attribute Certificate), which describes a role of a user participating in a marketplace. One AA (Attribute Authority) may issue values 'buyer' and 'seller' in a field named 'role' while another AA may issue values 'customer' and 'vendor' in a field named 'member type'. The semantic converter converts these two formats into some common format known to the relying party so that both AC are mapped to the same field.

Another example is a Trust Policy engine as in the Trust Establishment toolkit [17]. The toolkit allows a company to define a policy for mapping users to roles based on credentials. The policy language is rule based and it filters credentials assuming that each credential has some known type field (e.g. a *recommendation* credential, a *credit history* credential etc). The semantic converter can be used to assign a type to each credential even if the credential does not come with a predefined type field.

A natural question is, why do we need this conversion. Why can't we decide on some profile for each type of credential? The answer is that it is hard to decide on some common format that is acceptable on every issuing authority. Moreover, some credentials are extracted from legacy databases and it is almost impossible to force common fields.

The reason we believe the semantic converter will achieve the desired interoperability between the various credential formats is since it converts credentials only for a specific relying party which defines the conversion rules for itself and it does not try to coordinate all the issuing authorities in the world.

5.1 The Semantic Converter Architecture

It is not expected that one can write general-purpose software that can convert any given two similar credentials to a common known credential. Instead, we create a framework where customized converters can be plugged in. See Figure 4 below.

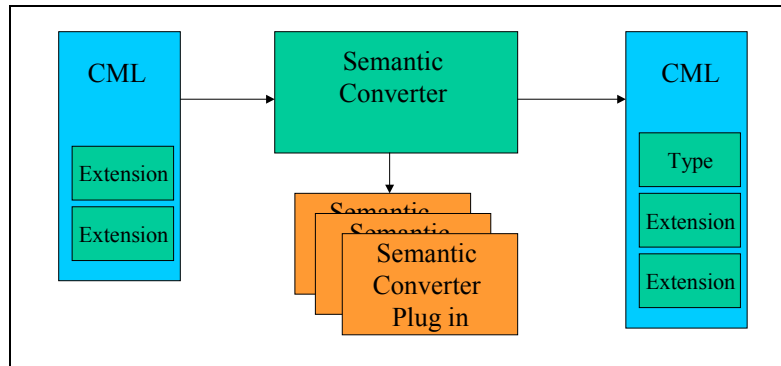


Fig. 4. Semantic Converter

Each Semantic Converter Plug-In (SCP) has to first register with the framework and has to implement some defined SPI (Service Provider Interface). On registration, the SCP informs the framework of which credential types does it know to accept as inputs, and which credential types does it know to generate as outputs. Some SCP may be willing to accept the `unknown` credential type, and then they will try to map it to a known credential type. When a new CML credential is given to the framework, the framework will invoke SCP modules to convert it from its initial type (or `unknown`) to the types meaningful to the relying applications.

6 Summary

We presented a framework for managing credentials needed by applications that consume credentials in order to make decisions. We have parts of the architecture already implemented as part of the Trust Establishment [17] toolkit while other elements still need to be developed.

Acknowledgements

We thank Tom Gindin for helpful discussions and comments, and providing [19].

References

1. M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis, The KeyNote Trust-Management System, <http://www.cis.upenn.edu/~angelos/keynote.html>
2. M. Blaze, J. Feigenbaum, and J. Lacy, Decentralized Trust Management, In Proc. of the 17th Symposium on Security and Privacy, pp 164-173, 1996
3. [A Technical Overview of the eccelerate.com Solution](http://www.Eccelerate.com), from <http://www.Eccelerate.com>.
4. C. Ellison, "The nature of a usable PKI", Computer Networks 31 (1999) pp. 823-830
5. Carl Ellison and Bruce Schneier, "[10 Risks of PKI](#)", Computer Security Journal, v 16, n 1, 2000, pp. 1-7.

6. S. Farrell and R. Housley, [An Internet Attribute Certificate Profile for Authorization](#). July 2000.
7. [Overview Of Certification Systems: X.509, PKIX, CA, PGP and SKIP](#), by Ed Gerck. [THE BELL](#), ISSN 1530-048X, July 2000, Vol. 1, No. 3, p. 8.
8. Access control meets Public Key Infrastructure, or: how to establish trust in strangers, A. Herzberg, Y. Mass, J. Mihaeli, D. Naor and Y. Ravid, IEEE Symp. on Security and Privacy, Oakland, California, May 2000.
9. Kohlas and U. Maurer, Reasoning about public-key certification - on bindings between entities and public keys, IEEE JSAC, vol. 18, no. 4, Apr, 2000.
10. [Internet X.509 Public Key Infrastructure: Certificate Policy and Certification Practices](#), S. Chokani and W. Ford, March 1999.
11. [Internet X.509 Public Key Infrastructure: Certificate and CRL Profile](#), R. Housley, W. Ford, N. Polk, D. Solo, Jan. 1999.
12. [SPKI Certificate Theory](#). C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. September 1999.
13. [PICS: Internet Access Controls Without Censorship](#), Paul Resnick and Jim Miller, *Communications of the ACM*, 1996, vol. 39(10), pp. 87-93.
14. M. K. Reiter and S. G. Stubblebine. [Path independence for authentication in large-scale systems](#). Proc. 4th ACM Conf. on Computer and Comm. Security, pp. 57-66, Apr. 1997
15. Simple Public Key Infrastructure (15), <http://www.ietf.org/html.chapters/15-chapter.html>
16. SSL 3.0 Specification, Netscape, <http://home.netscape.com/eng/163/index.html>
17. Trust Establishment toolkit, see at <http://www.hrl.il.ibm.com/TrustEstablishment>.
18. ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.
19. ITU-T recommendation X.509 | ISO/IEC 9594-8: "Information technology – open systems interconnection – the directory: public-key and attribute certificate frameworks".
20. XML Security Suite <http://www.alphaworks.ibm.com/tech/xmlsecuritysuite>
21. Extensible Markup Language W3C Recommendation: XML 1.0, <http://www.w3.org/TR/WD-xml-lang.html>.
22. P. Zimmerman, The Official PGP User's Guide, MIT Press, Cambridge, 1995.
23. SET Secure Electronic Transaction <http://www.setco.org>