

# Artificial Intelligence Laboratory

**Author: Eric Knauss**

The goal of this laboratory was to create a secure method of exchanging (and storing) credentials between peers of a peer to peer network that participate in automated trust negotiation. This method shall extend a prototype of an automated proving engine for automated trust negotiation and enable it, to easily encrypt credentials before sending them down the network as well as handle digital signatures.

## ***Credentials in automated trust negotiation***

As already mentioned a trust negotiation engine already exists. This engine decides whether a certain credential is sufficient to grant its owner access to a resource. What makes this scenario a bit more complicated is that credentials also are handled as resources and access to them may be restricted, too. Therefore a client could decide to only give a certain credential to a server, if this server would give a proof of its trustworthiness (i.e. a credential) first.

## ***Cryptographic aspects of automated trust negotiation***

In this laboratory certain technical cryptographic aspects should be solved. These are digital signatures, encryption and storage. Digital signatures are a very important part, because an electronic credential consists just of a piece of digitally signed data. Imagine a server of a university granting access to a script only to a student of this university. To prove this fact, a piece of data is needed that

1. states that a certain person is student at this university
2. could not easily be created by third persons.

To achieve this, digital signatures are used. In this example the university states the fact “John is student of this university”. The byte values of this text string would then be used to compute a hash value of the fact. This hash value is encrypted with the private key of the university and the result is added to the fact. In this way a credential is created that consists of two parts: the readable fact “John is student” and a digital signature. Everyone who has a public key of the university could compute the hash value of the fact and test, if the digital signature could possibly be created by the corresponding private key from this hash value. A positive result can only be achieved if the text string was not changed after being signed and if both keys that were used belong to a key pair of an asymmetric crypto algorithm.

Annotation 1: The step of computing the hash value is a standard procedure that is needed for large texts. The hash function computes a value of fixed length, no matter how large the text is. This is important, because asymmetric crypto algorithms are rather slow and should only be used on short messages. On the other hand, it is important, that the message is not short, because otherwise security cannot be granted. That is why even short messages like credentials should be hashed.

Annotation 2: An important thing in this scenario is a good public key infrastructure. A digital signature will only make sense, if the person checking it can be sure that the public key really belongs to the expected signer. Otherwise a third person could claim to be the university, sign

the fact with his own private key and distribute the corresponding public key as if it belonged to the university. Public key infrastructures are not discussed in detail in this work.

Encryption is the next important aspect. It is especially important with credentials that are restricted and should only be available to a small circle of persons. It would not make sense to send these credentials as clear text down a network. Fortunately, many good solutions exist for this problem. The method of choice is the use of the Secure Socket Layer (SSL) that provides a high level of security (and again utilizes the public key infrastructure mentioned above). It is also easy to use, because most programming languages provide libraries for SSL. The general idea is to build a data stream that handles the cryptographic issues in a transparent way. The only thing to be done here is to agree upon the type of data that should be sent. The alternatives are to serialize java objects to this stream (and thus make java the only possible programming language for participants of automated trust negotiation) or to send data in a international standardized format (which is more troublesome).

The last cryptographic aspect is the storage of credentials. Because the slightest change in a credential would break the digital signature, credentials have to be handled with care. Also sensitive credentials should be stored in an encrypted file. More important is the format in which credentials are stored. The alternatives are again serialized java objects or international standard formats. In this case an international standard is highly preferable because it makes the problem of distributing (including creating) credentials easier.

## **Alternatives**

The most important thing is to decide upon the format of credentials. The format should support all the cryptographic aspects. In this section three alternatives are discussed.

### **Java classes**

The main advantage in using java classes is that it is very easy to integrate them into the existing project. The fact of a credential can be stored as a string and the signature is another field of this class. The credential class could be made Serializable and send over a secure network connection or written to file. This file could even be encrypted to provide a secure storage. All other alternatives would also require an interface of java classes, because they need to be integrated into the existing java project. The disadvantage is the lack of interoperability with other programming languages. This is an important issue because the need of electronic credentials exists in many other fields, too. Therefore a standard for credentials might be invented.

### **Secure Xml**

There are many projects concerning security in respect to XML documents. Important issues in these projects are role based security models that allow making parts of XML documents only editable by certain persons. Also it is possible to sign only certain parts of a XML document, for example the text a user inserted into a form. With these cryptographic extensions it is possible to use a XML document for trust negotiation: The client would send a request in a new XML document. The server would read this request, add an empty credential field for each needed credential to the document and send it back. The client would recognize the initial request it posted and handle the empty credential fields as requests of its own. It

might attach new credential requests to them or insert the required data. Every change to the document is signed on the fly before sending it and checked for plausibility each time it arrives. The disadvantages of this approach are:

1. The amount of cryptographic operations needed in each step.
2. No international accepted standard exists
3. A lot of work still remains to be done to use this cryptographic extensions

The main advantages are the good readability of the documents and that the structure might reflect certain parts of the reasoning engine. All in all it seems too complicated for the expected gain.

## **X.509 certificates**

X.509 is a wide spread standard for distribution of public keys in a public key infrastructure. A certificate contains the public key of its owner that is signed to prove the identity of the owner. Also the certificate of the person that signed the public key is included. This certificate has the same structure and vice versa. In this way a certificate consists of a chain of certificates. The top certificate of this chain is a self-signed certificate of a certified authority. The identity of the owner of a certificate can be accepted, if this certificate authority can be trusted. What makes this standard interesting for credentials is the possibility to store additional information into a certificate. This certain type of certificates, sometimes called attribute certificate, stores additional information like an email address or an url. A character string can be included and it exists an usage indicator that prohibits the usage of this certificate to all applications that do not understand this field. All this data is encoded device independently by an international standard called ASN.1. Thus it is ensured certificates can be used on every operating system that supports this standard. The advantage of this alternative is that it is really an international standard. It can be used independently in virtually any programming language on any computer. Also a wide spread public key infrastructure can easily be used in this way. The disadvantage is that it is not easy to create certificates because in standard java there exist simply no methods to create them. Command line tools like java keytool only allow the creation of certain types of certificates. It is not possible to create attribute certificates that contain the fact of a credential. Therefore a security provider must be used making the methods available to create attribute certificates. Fortunately, a provider could be found that not only provides these methods but also is an open source project that is free to use: Bouncy Castle.

## **Conclusion**

Certainly X.509 seems to be the best alternative for now. But as other technologies like secure XML promise advantages for the future, the extension written in this laboratory needs to be designed in a way that allows changing the underlying format of the credential easily.

## ***How to use the new package***

The credential package consists of the two main classes: Credential and Credential Store. A credential is loaded into the reasoning engine via its string representation. For convenience the Credential Store has a get All Credential Store method that returns an array of strings.

## **The Credential class**

This class is meant as an interface between the reasoning engine and the format of the credential. The format of the credential is encapsulated and not visible to the application. This class is declared abstract and must be extended by a subclass that handles the translation from

the chosen credential format to the string representation. The method `getEncoded` returns the credential in its original format. This object could be send to the other peer. The alternative is to send the `Credential` itself which is serializable. The only implemented subclass so far is the `X509Credential`.

### **The CredentialStore class**

The `CredentialStore` is also an abstract class that manages a set of credentials. Because the reasoning engine works with string representations of credentials a method is needed, that fetches the right real credential for a given string representation. Implementations of this class also manage the storage of credentials.

### **How to load credentials into the prolog engine**

By giving a filename to the `CredentialStore` it is able to load credentials into the memory. Then the method `getAllCredentials` is used to fetch the `String` representations and load them into the engine. Implementations of `CredentialStore` should also be able to import credentials that are in a given format. Such a function is implemented in the `X509CredentialStore`.

### **How to store credentials**

By calling the `storeCredentials` method of `CredentialStore` with a filename, all `Credentials` are written to this file. The format in which the credentials are stored depends upon the implementation of the subclass. An `X509CredentialStore` uses build-in java methods to store certificates into a `CertificateStore`.

### **How to send credentials over the network**

Here two alternatives are possible: On the one hand serializable `Credentials` could be serialized to a data stream if the communication partner uses java too. On the other hand the programming language independent encoded representation of a credential could be send. `X509` certificates is serializable itself.

### ***Appendix: Encoding a credential into a X.509 certificate***

`X509` certificates have a very strict syntax. There exists the possibility to write extensions and in this way a new type of certificates could be created, for example a credential certificate. In this case, however, the advantage of the `X.509` alternative, being a wide spread standard, would be lost. The idea of storing additional information into a certificate is not new. For internet applications extensions already exist, that makes it possible to encode for example urls and email addresses. This extension is called "subject alternative name". In this extension it is also possible to set a field called "Other Name" that accepts any text `String`. By utilizing this field there is no need to go deep into `ASN.1` syntax on how to write extensions and existing tools of security providers can be used to set this extension. ,