Architecture for using the *idemix* Private Certificate System for Identity Federation

Jan Camenisch Dieter Sommer
IBM Research
Zurich Research Laboratory

Draft - Revision 0.6

The idemix system is a powerful *private certificate system* that enables better privacy in identity federation scenarios than today's approaches while at least maintaining the same security properties.

Private certificate systems have additional architectural requirements regarding client-side architectures compared to what is needed for traditional identity federation protocols. This is due to the difference in the paradigms of private certificate systems and traditional identity federation approaches. This document briefly outlines key concepts of private certificate systems and a basic architecture for implementing identity federation using a private certificate system.

We think that an architecture should be able to handle both traditional federation approaches as well as those using private certificates.

1 Private Certificate Systems

Private certificate systems are a generalization of anonymous credential systems as put forth by Chaum. They offer all privacy advantages of anonymous credential systems, but provide additional functionality to what anonymous credentials provide, in particular arbitrary attributes and proofs that can be made over the attributes of multiple private certificates. A private certificate system can be characterized as follows:

- A private certificate is a set of attributes signed by an identity provider with a special signature scheme.
- A user obtains a private certificate once and can use it in many transactions with relying parties. That is, private certificates can have longer lifetimes than security tokens used in today's federation protocols.
- A private certificate is never sent to a relying party. Instead, a new security token is generated from it by the user for each transaction. This newly generated security token is sent to the relying party. This new token verifies under the issuer's

public key as would a traditional certificate and it can not be generated without being privy to a private certificate. Moreover, possession of a private certificate is sufficient to create the security token for the transaction; therefore the identity provider needs not be involved in the transaction of the user and the relying party.

- All transactions performed with one particular private certificate are unlinkable. This holds, of course, only if the released attribute information does not allow for linkability.
- In one transaction, an arbitrary subset of the attributes of multiple private certificates can be released. It is even possible that for a specific attribute only partial information on this attributes is released, e.g., that the balance attribute of a bank statement is greater than 3000 is released, even though the attribute value in the private certificate is 4230. This is possible in a secure manner, that is, the user can only release (partial) attribute information that is consistent with the attributes in the certificate. Also, polynomial expressions over attributes can be released in a transaction, e.g.,

$$bankstatement_1.balance + bankstatement_2.balance > 4000 \ .$$

- The security token generated in a transaction verifies against the public certificate verification keys of the issuers of the involved certificates.
- Attribute information as contained in certificates can be encrypted by the user. In addition, a proof is provided that the ciphertext contains specified attribute information of certificates the user has. This process does not involve the party the attributes are encrypted for, but only its public encryption key. This mechanism allows for conditional anonymity by encrypting identifying attributes that will—if the decryption condition is met—be decrypted by the trusted party responsible for anonymity revocation.

2 High Level Requirements

The use of private certificate system for federated identity management requires a few particulars. Foremost, the private certificates require the user wallet to (be able to) transform a certificate into one that can be sent to a relying party. Guiding this transformation requires languages that describe into what the certificate should be transformed, i.e., what kind of token of certificate the relying party requires to grant access to some resource or service. We call such a statement *security policy* in the following. This in turn requires a matching by user's wallet to determine which (private) certificate can be used to statisfy the security policy and some components on the relying party's side to generate and send this policy to the user. Furthermore, this matching might not be fully automatic but might require the user's interaction, in particular if there are multiple certificates that could be used or if there is no certificate available. Similarly, a when a certificate is issued, a language is required to enable the issuing of certificates. This language will describe what certificate an issuer can provide. Here there might of course be some option possible, e.g., whether or not a driver's license should include the holder's address. Again, the issuing party will require some components to handle the specification and the issuing of the credentials. Thus, when a user requests such a certificate, she must be able to specify which of the options she wants. All the languages mentioned are quite similar and in practice will be subsets of a single language. Notice that these requirements, however, make the use of traditional credential easier as well. The only requirement that is particular to private certificates is the one that the user's wallet needs to be able to transfrom the certificates obtained into certificates (or tokens) sent to a relying party.

3 Concepts and Terms

We outline the key concepts we later use in the explanation of the architecture.

Transaction. A transaction is composed of the steps required for executing either a protocol for a WS-Security-based claim statement to assert claims or a WS-Trust exchange for obtaining a security token (in particular a private certificate). Transactions can be nested within each other. In such a case, the nested transaction is a sub-transaction, and the outer transaction a super-transaction.

Rollback semantics of a sub-transaction is that once it has been successfully terminated, it cannot be rolled back, even though its super transaction is aborted. For example, if a super-transaction is executed for making a claim based on a private certificate that is not available yet, then the certificate can be obtained in a sub-transaction. Once the certificate has been obtained, this transaction cannot be rolled back any more.

Transactions can be nested up to arbitrary depths, but only small depths will make sense for the average users and for many practical scenarios for identity federation.

Security Policy. A Security policy P defines what claims need to be provided in order that access to the requested resource or service be granted (where the service includes issuing of a security token or certificate). A security policy can express disjunctions of conjunctions of predicates. A predicate is specified over at least one attribute of a certificate.

Consider the following example of a security policy containings two alternative parts.

$$(\mathit{age} > 21 \wedge \mathit{firstname}) \vee \mathit{membershipcert}$$

The predicate $\mathit{age} > 21$ has semantics that the age of the requester has to be proved to be greater than 21, the predicate *firstname* requires that the firstname be provided in a certified fashion. The predicate *membershipcert* requires that holdership of a membership certificate has to be proved.

Claim request. A claim request $\widehat{P}$ is an internal representation of a security policy. This representation is used for processing.

Annotated claim request. An annotated claim request $\widetilde{P}$ is a claim request annotated with information on how the different predicates of the claim request can be fulfilled.

Claim selection. A claim selection R is a subset of an annotated claim request $\widetilde{P}$ that indicates the choice of the user on what option of $\widetilde{P}$ shall be fulfilled and what certificates to use for the fulfilling.

Claim specification. A claim specification is a precise specification what attribute information to release in a particular transaction. A claim specification can be transformed 1:1 into an idemix assertion.

Idemix assertion. An idemix assertion A defines claims that are made to an STS, e.g., $\mathit{age} > 21 \wedge \mathit{firstname}$.

Idemix proof. An idemix proof p_A provides assurance of a third-party's endorsement for an idemix assertion A. Such a proof is computed based on the assertion and from private certificates (which could be issued by different parties). Such a proof is (embedded in) a security token that is sent to an STS within a WS-Security-protected message.

Ontology. An ontology that specifies how attributes of certificates can be used for proving attribute requests of a security policy must be available in the system and agreed by each system participant.

The assumption to have such an ontology is realistic, in particular, if only the most-commonly used attributes and certificates are concerned. For proprietary attributes, individual organizations may define their own attribute types and ontology that extends the system ontology. A first implementation of the ontology can be simple and need not account for a completely generic model of handling ontologies.

4 Architecture

This section outlines a basic architecture for using private certificates for identity federation protocols for the client side. See Figure 1 for the high-level component architecture.

4.1 Data Abstraction Layer

The Data Abstraction Layer provides an abstract interface to all kinds of data stores and different implementations thereof.

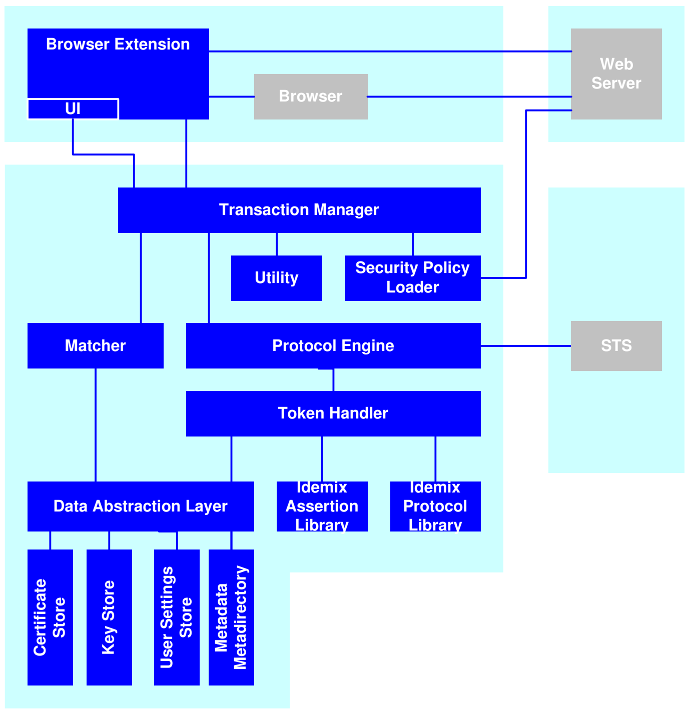

Figure 1: High-level architecture

4.2 Certificate Store

An implementation of the Certificate Store provides a sufficiently secure storage facility for private certificates of the party.

4.3 Key Store

An implementation of the Key Store provides a sufficiently secure storage for the user's private key and the public certificate verification keys.

4.4 User Settings Store

The User Settings Store provides functionality for keeping the user's settings and preferences she defines in her interactions.

4.5 Transaction Manager

The Transaction Manager handles the flow for executing transactions inside the client-side software system. Multiple concurrent transactions can be handled at a time. Nested transactions can be handled as well to support full-fledged WS-Trust flows for obtaining new private certificates within identity provisioning transactions. Thus, the component has to keep state and can be seen as a central flow orchestrator within the client-side architecture.

A transaction is triggered when the user starts a WS-Security or WS-Trust flow. The Transaction Manager controls the flow for the transaction at an early stage of the transaction.

4.6 Matcher

The Matcher receives a claim request $\widehat{P}$, that is an internal representation of a security policy P, as input. The Matcher determines the certificates that can be used to fulfill the individual parts of $\widehat{P}$. For each predicate, this includes the certificate type, a certificate identifier for retrieving the certificate from the Certificate Store, the attribute, and possibly a transformation. The transformation states how the attribute of the certificate can be used to fulfill the predicate for the request. For example, if a predicate requires that $age > 21$ be proved, this would lead to $passport.birthdate < currentdate - 21 years$ as proof specification in case the user possess a passport certificate that contains the *birthdate* attribute. This assumes that an ontology is available for providing the semantics and mapping rules.

The information computed by the Matcher is attached to $\widehat{P}$ yielding an annotated claim request $\widetilde{P}$. This annotated claim request still contains all options of the original security policy.

4.7 Protocol Engine

The Protocol Engine is the protocol machine for executing the WS-Security and WS-Trust protocols with an STS of an other party. This includes establishing connections to STSs, and sending and receiving web services messages.

4.8 Token Handler

The Token Handler provides the functionality for assembling and verifying web services tokens for the WS/idemix token type. WS-Security-protected and, more specifically, WS-Trust exchanges are supported.

4.9 Idemix Assertion Library

The Idemix Assertion Library is used for transforming a claim specification into an idemix assertion.

4.10 Idemix Protocol Library

The Idemix Protocol Library implements the cryptographic functionality specified by the idemix private certificate system. This is mainly the protocol for obtaining a private certificate and the protocol for making claims using private certificates.

For making claims with a private certificate, the component receives an idemix assertion, the key material, and the required private certificates as input. The output is a proof token p_A.

4.11 Browser Extension

The Browser Extension hooks the browser to the identity management application and provides a UI. All identity provisioning or certificate issuing transactions are triggered by the UI.

4.12 Utility

The Utility component provides different utility functionalities and it is conceivable that the component be split up or the functionality be distributed to other components.

5 Discussion

Key differences to an architecture for traditional security tokens are the following:

- Private certificates are never sent to any other party once they have been obtained. Instead, a proof is computed using private certificates and then sent (with an assertion) to the other party.
- Private certificates can be reused in many transactions without these transactions becoming linkable unless the revealed attributes establishing a link by their uniqueness. When using traditional approaches for federating identity, a new security token has to be obtained for each new identity provisioning transaction.

The following issues have not yet been accounted for in the architecture as they are commodity features and not different for private certificate systems:

- An infrastructure to distribute public keys and onotologies.
- Trust decision on keys (e.g., based on PKCs and key chain validation).

Below, an example execution sequence for a protocol for provisioning identity information using idemix claims (any WS-Security or a WS-Trust RST message) is presented. Note that the resulting claims are third party-endorsed by the issuers of the private certificates used in the claims although the issuers are not involved in the generation of the idemix proof token.

1.1 Transaction Manager: is triggered by the Browser Extension with a security Policy P or the location of a security policy (we assume the policy has already been obtained by the Browser Extension in our example and is already provided to the transaction manager); the Utility component is used to transform the security policy P into a claim request $\widehat{P}$.

1.2 Matcher: receives a claim request $\widehat{P}$; computes the possible fulfillments of $\widehat{P}$ based on the available private certificates; the resulting annotated claim request $\widetilde{P}$ is returned.

1.3 Transaction Manager.

1.4 User Interface: displays $\widetilde{P}$ to the user in appropriate form.

1.5 User: chooses how to fulfill the request; result is the claim selection R.[1] If the user chooses to obtain a non-available private certificate, a sub-transaction can be triggered at this point.

1.6 Transaction Manager: calls on the Utility component to get R transformed into $\widetilde{R}$, the release specification.

1.7 Protocol Engine.

1.8 Token Handler: receives the release specification $\widetilde{R}$ as input and starts building the WS message; retrieves the required keys, and private certificates using the Data Abstraction Layer.

1.9 Idemix Assertion Library: transforms the release specification $\widetilde{R}$ into an idemix assertion A.

1.10 Token Handler.

1.11 Idemix Protocol Library: computes the idemix proof p_A.

1.12 Token Handler: Generates the WS/idemix message from p_A and A and other required elements; returns the message.

1.13 Protocol Engine: Connects to the STS and sends the WS/idemix message Receives a response message (handling of this message is not further discussed).

Nested transactions: A nested transaction can be triggered by the UI in step 1.5 when the user chooses to obtain a non-locally available private certificate from the appropriate identity provider. This starts the sub-transaction for a WS-Trust exchange similarly to the transaction above, but nested into the above transaction. When the sub-transaction has been completed, the super-transaction will proceed, adding the newly obtained private certificate to the annotated claim request by restarting at step 1.2. The concept of sub-transactions applies recursively.

[1]The selection process can potentially be supported by an additional component or the functionality is provided in the Browser Extension or UI.