Paul Madsen, NTT
David Chadwick, University of Kent
May 10, 2005

Authority Recognition

Status of This Memo

This document provides information to the Grid community. It does not define any standards or technical recommendations. Distribution is unlimited.

1. Abstract

Trust between entities in many Grid transactions is enabled by a separate authority issuing assertions (e.g. X.509 certificates, SAML assertions, Kerberos tickets, etc) regarding the identity and/or other characteristics of the actors involved. The assertions issued by an authority must be recognized as valid and appropriate to a relying party's requirements before that party will rely on them. Whether or not an assertion from a particular authority is appropriate will depend on a number of factors, including the commitments the authority makes with respect to the assertion, the liabilities the authority is willing to assume, and the obligations assumed by the relying party if they use the assertion. Existing mechanisms do not facilitate the dissemination of this information from the authority to the relying party to enable an informed recognition decision. This paper explores the concepts and issues involved in an authority making available to potential relying parties this information. Additionally, we propose a representative model for using XML and a publish-and-subscribe model to offer a simpler solution to the authority recognition problem.

Contents

2. Introduction

Web service clients and providers are connected together though more than the transactions in which they engage. Underlying this network of messages and transactions is another; a trust network that connects businesses together through the trust relationships that exist between them. In the emerging security infrastructure for Web Services, the hubs in these trust networks are called assertion authorities or security token services. It is these entities that make trust networks scalable – without them to mediate the trust between web service actors necessary for real business transactions, these actors would need to establish and manage trust with each other independently.

Security token services (STS) make trust networks scalable by mediating trust between nodes that would otherwise not be able to ascribe trust to another. Rather than maintaining pair wise trust with all potential partners, individual SOAP actors instead form a trust relationship with the STS and then rely on the STS to form sufficient other trust relationships such that the necessary indirect trust can be established. Based on the trust that an SOAP actor places in the STS, it may choose to extend trust to another entity in which the STS places its confidence.

This begs the question(s) – how and why do the SOAP actors come to place trust in the 3rd party STS? Since the individual SOAP actors will subsequently make decisions to trust multiple security tokens (and consequently the bearers of those tokens) based on their trust in these authorities; the decision to trust a STS is significant and not one to be taken lightly. So how does a SOAP actor make this decision? What questions does it ask of the STS before doing so? What are the criteria by which a SOAP actor measures such an STS?

This document examines these issues.

3. Conceptual Overview

3.1 Trust & Expectations

X.509 [X509] defines trust as:

"Generally, an entity can be said to trust a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects".

The first entity makes this assumption only about a relevant area of the second entity's behavior, and so the trust between them is confined to that area. According to this definition, the essence of trust lies in a trusting party not being disappointed in it's reasonable expectation of another's behavior. Trust then is dependent on each actor being confident that the other actors will behave in the prescribed manner for specific circumstances and, importantly, the protections afforded to those impacted parties should another not perform as expected.

The diagram below demonstrates some of the expectations that the actors in a simple PKI might have of each other.

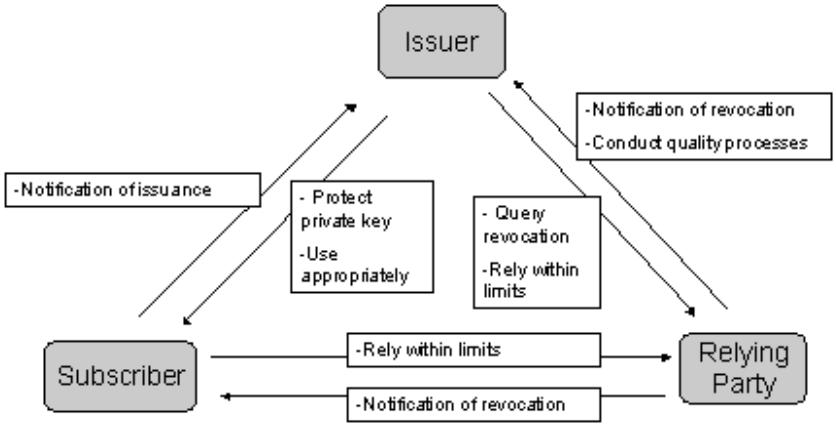

Figure 1: Expectations in simple PKI

3.2 Expectations & Obligations

An obligation is a legally binding commitment to an enforceable duty. An expectation can become an obligation if the actor concerned makes a legally binding commitment to perform (or not) the relevant operation.

3.3 Obligations & Liabilities

If an obligation is breached, then it may be decided that a liability is assumed by the negligent party. The conversion of an obligation into a liability requires a dispute-resolution system; the output of this system will be a judgment or reward. Participants may attempt to constrain the liability they are willing to accept

The conversion of expectation to obligation, and obligation to liability is shown below

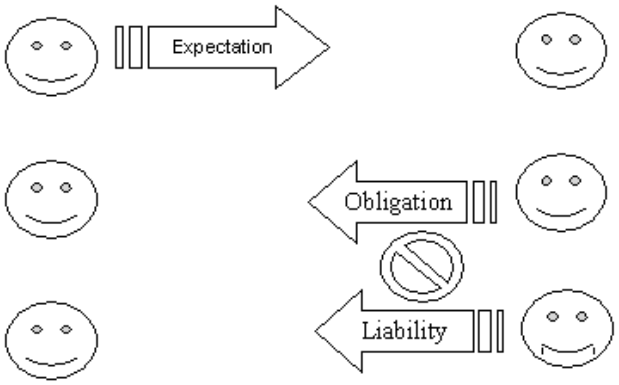

Figure 2: Expectation to Obligation to Liability

An expectation ‘A has of B’ becomes an obligation ‘B undertakes to A’ when B makes a commitment to that extent. An unmet obligation becomes a liability if a dispute resolution mechanism determines it to be appropriate.

3.4 Trust Through Assertions

Trust between entities in many web services transactions is enabled by a separate authority issuing assertions/security tokens (e.g. X.509 certificates, SAML assertions, Kerberos tickets, etc) regarding the identity and/or other characteristics of the actors involved. Senders and recipients are typically able to ascribe sufficient level of trust in a security token because they can be confident of its origin, i.e. they know who created the token. It is through the existing trust they already have in the 3rd party security token issuer that they are able to derive indirect trust in the holder of a security token created by that issuer.

3.4.1 Actors

There are multiple parties involved in the creation, use, and validation of assertions.

- The individual, company or application identified by the assertion
- The entity that identifies and authenticates the information that will go in the assertion (e.g.
- The issuer of the assertion (e.g. a Certificate Authority in X.509 and an Identity Provider in SAML)
- The individual, company or application that relies on the assertion.

- The entity that defines the rules that govern the other actors' activities with respect to the assertion (e.g. the Policy Authority)

Depending on the business model and/or deployment scenario, some of these roles may be combined. For instance, in an enterprise scenario in which a company issues certificates to its employees, the policy authority and certificate authority would likely be one and the same.

3.4.2 Scenarios

The following sections present examples of how trust between entities can be mediated through the involvement of a 3rd party authority issuing assertions.

3.4.2.1 SAML-based SSO

Single SignOn (SSO) refers to a user authenticating to a single entity and then, based on that authentication, be automatically logged-in at the other resources they will subsequently try to access. SSO is typically enabled through the entity that originally authenticated the user creating an assertion to that effect and communicating that assertion to the subsequent resources at which the user attempts to access protected resources. For SSO on the Web, SAML (Security Assertion Markup Language) standardizes both the XML structure of this assertion as well as a request/response protocol for communicating the assertion between the two web sites involved. The Liberty Alliance bases its framework for federated identity on SAML – extending the SAML assertions and protocols to better support real-world scenarios of federated identity.

The second Web site logs the user into their account without requiring them to explicitly authenticate (e.g. with an account name and password) again as they would normally require. It is willing to do so because the SAML authentication assertion it receive attests to the fact that the user did indeed authenticate at the first site within a certain period of time. Since the second web site can be confident of the origin and integrity of the SAML assertion, and because of the confidence it has that the first web site will only create such an assertion after the user did actually authenticate to it, the second site can be confident that the user is indeed the individual whose identity is asserted to in the SAML assertion.

3.4.2.2 WS-Trust 'Introduction'

For Web service clients, as opposed to individual browser users, SSO is not a concern. A SOAP client will not forget its 'password' (and will not depend on sticky notes to achieve this) and will not balk at authenticating over and over to different web service providers. But for SOAP clients, a sequence of events comparable to the SSO scenario described for SAML above, is still relevant – but as a mechanism for brokering trust rather than user convenience.

A SOAP client, desiring to engage in some secure transaction with an arbitrary service provider, may be motivated to go first to some 3rd party authority not for convenience but because, by default, there would likely be no trust relationship between itself and the service provider. However, if there were trust between the client and some 3rd party, and between the 3rd party and the service provider, then indirect trust can be established between the client and the provider – hopefully enabling the desired business transaction.

WS-Trust is a proposal (primarily from IBM & Microsoft) that defines how the client and 3rd party would communicate in order to enable subsequent secure and trusted communication between the client and the provider. The SOAP client would send a message to the WS-Trust Security Token Service (STS) requesting that the STS issue it a token which, when subsequently presented to the service provider by the client (as part of the actual business transaction), will

allow the provider to place sufficient trust in the (theoretically unknown) client to proceed with the transaction. The trust that the provider has in the STS allows it to assign trust to the tokens issued by that STS and consequently, to the SOAP clients to which the tokens are issued.

Note: although SAML was described in the context of a browser SSO scenario, and WS-Trust in the context of a SOAP transaction, neither are restricted to this application and can be used in either case. Indeed, the Liberty Alliance's ID-Web Services Framework uses SAML in the manner described for WS-Trust and WS-Federation (another IBM/Microsoft proposal) profiles how WS-Trust can be applied to the browser SSO case.

3.4.2.3 X.509 Certificate Authority

An X.509 certificate is an assertion that cryptographically binds together some entity's identity (e.g. a user's name or an SSL server's IP address) with a public key. By proving that they are in possession of the private key associated with the public key contained within, the user can lay claim to the asserted identity when they present the certificate to some other party in the context of an application.

Anybody can create a certificate for themselves in which they could claim any identity. Certificates will only be trusted (and thereby provide value) if a relying party can trust the issuer – typically a trusted 3rd party called a Certificate Authority (CA). The relying party will either directly trust the issuing CA for a certificate presented to it, or may trust that CA because its own CA (that issued it a certificate) trusts that other CA.

Trust between certificate holders and relying parties is made possible by the (potentially long) set of other trust relationships that exist between the two fundamental entities and their associated CAs. It is through the trust that relying parties have for CAs that they will be willing to ascribe trust to the subjects of certificates issued by those CAs (or other CAs for which a chain of trust can be built).

3.4.3 Trust Layers

The previous sections demonstrated how the concept of trust being enabled through a logical STS is applicable for all of SAML-based SSO, WS-Trust Introductions, and X.509 Certificate Authorities. Perhaps more interesting than this parallel is that the trust established through one mechanism can be the basis of trust for another mechanism. For instance, the trust established between two certificate holders based on their shared trust with the same X.509 Certificate Authority could enable SAML-based SSO between them as they would be able to verify and trust the digital signatures calculated over the SAML assertions. This scenario is shown in the diagram below

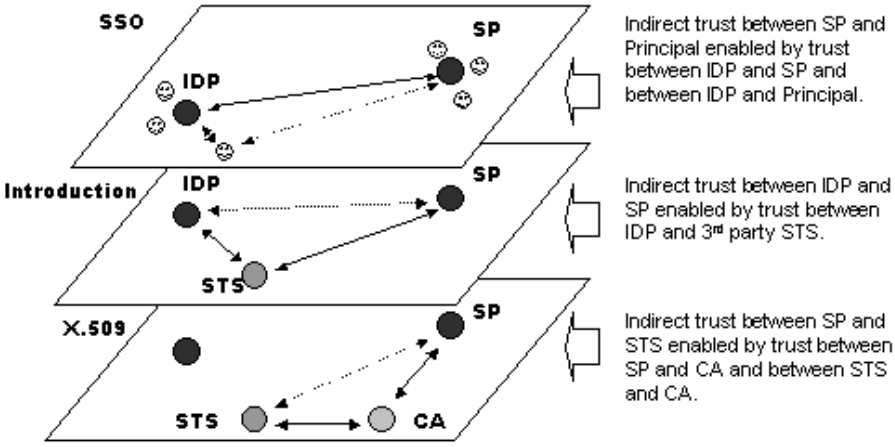

Figure 3: Trust Layers

Trust between two entities that is *indirect* at one level may be *direct* at a higher level.

4. Authority Recognition

Since an entity, in trusting an authority (or equivalently, deciding to recognize the assertions issued by that authority), will consequently assign trust to other entities *certified* by that authority - the trust that an entity has in an authority will be based on its fundamental expectation that the authority will only place its own trust in valid entities.

An entity, in determining whether or not to trust a particular STS for a given context, will likely require more information than simply the implicit belief that the ST will issue tokens only to valid entities – the details will likely matter.

The suitability of an assertion to a particular application may depend on a number of factors, including

- The nature of the application (some applications will be more sensitive, implying more stringent criteria)
- The technical capabilities of the relying party (certain obligations imposed on it by other actors may be beyond its capabilities)
- The relying party's risk tolerance (obviously related to the nature of the application but different relying party's may have difference tolerances for the same application)
- The level of liability the authority is willing to assume (all else being equal, a relying party will prefer an assertion for which their financial risk is minimized).

Consequently, the STS should be able to provide to a potential relying party sufficient information to facilitate the decision (assuming that the STS does indeed want the 'business'). It is from the information that the STS makes available that an entity will, at least partially, base its expectations - and consequently its trust.

The nature of this information can vary. Ultimately, the question a relying party must answer in determining whether or not to trust an STS and recognize the tokens it issues as valid for a particular application is:

"*Knowing what I know about the other participants, is such a token/assertion appropriate for the application for which I intend to use it*"

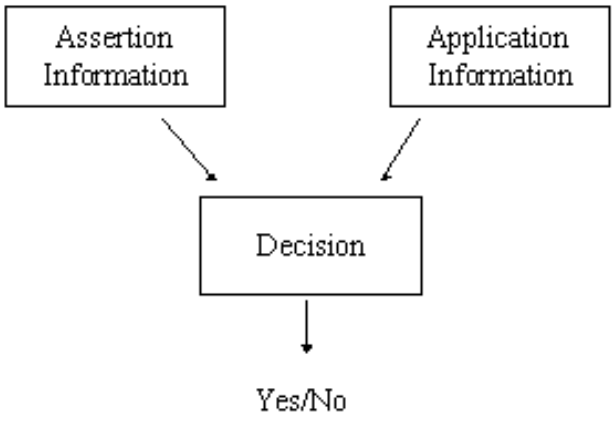

Figure 4: Assertion Recognition Decision

4.1 Assertion Information

What the relying party "knows" about the STS's assertions can take two general forms, differing in the nature of the assertion/commitment (with respect to the assertions it issues) that the STS makes available

1. *They assert that their processes and practices under which this token was issued are as follows…*

An example of this category of information is the Certificate Practices Statement (CPS) by which an X.509 Certificate Authority advertises the details of the processes through which it issues certificates. Another example is the Liberty Alliance's Authentication Context mechanism by which an Identity Provider describes the details underlying the SAML assertion enabling SSO. A variant of this model is for the STS to assert that their processes and practices are such that the assertions are of a certain *assertion class*. An example of this category is the Liberty Alliance's Authentication Context class mechanism by which a Liberty Service Provider can indicate to an Identity Provider its high-level requirements for the processes that underlie the SAML authentication assertion being requested.

2. *They assert their commitments with respect to the assertion along with the obligations assumed by any entity that uses the assertion.*

In this model, the STS does not divulge the details of their technical infrastructure and processes to a potential relying party. Rather, it describes the degree to which it 'will stand behind' the assertions that it issues – the assumption is that this commitment is more fundamental to enabling confidence in the authority's assertions than the specific processes followed.

An example of this class of information is a Certificate Policy statement by which an X.509 Certificate Authority lists the applications for which a certificate is approved and may limit its liability with respect to use of those certificates.

Whether the 'process-oriented' or 'business-oriented' approach is appropriate to a given situation may depend on the nature of the application for which an authority's assertions are being used and the relationships between the actors involved.

4.2 Application Information

The other key category of information that the assertion recognition decision will require as input are the details of the application for which the use of the assertion is being contemplated.

If the STS is oblivious to the potential applications of its assertions - it will be unable to assert/claim that its assertions are appropriate for any specific application. The relying party will necessarily provide the application information as input to the decision process

4.3 Risk analysis

Upon receiving the authority information, a relying-party must determine whether or not the assertion is appropriate for the intended application, i.e. the relying party must decide whether or not to recognize that authority's assertions for that application.

4.4 Policy Authority

The **policy authority** is the actor that defines the responsibilities of the other actors with respect to issuance, use, and reliance of the assertions. This role may or may not be distinct from others. The diagram below shows three scenarios for the relationship between the policy authority and the other participants:

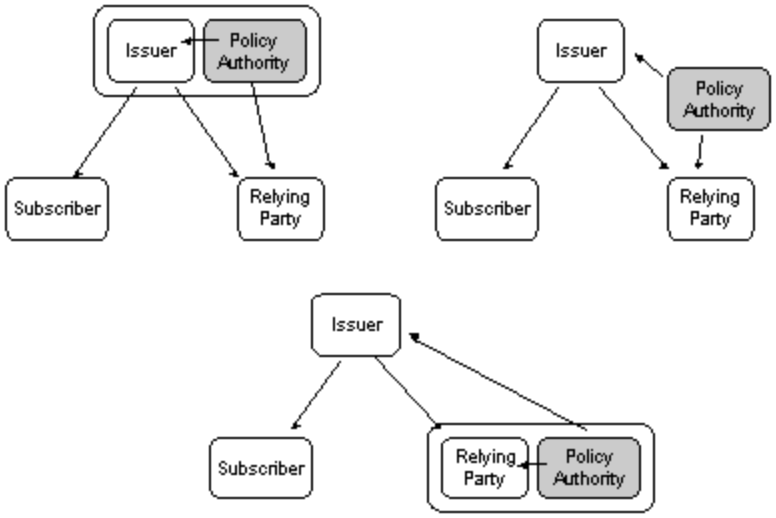

Figure 5: Policy Authority

In the top two models, the *separation* between the policy authority and the relying party may prevent the assertion policy from being tailored to the applications for which the assertion will be used - the assertion policy will necessarily be *generic* with respect to particular applications. It will be the relying party that, presented with an assertion and a potential application for it, will face the decision of determining whether the assertion is appropriate for that application, using the generic assertion policy as input to the decision.

In the bottom model, as the policy authority is co-located with the relying party, it is more likely to have knowledge of the specific applications for which assertions will be used and, consequently, the assertion policy it creates need not be generic to these applications. Indeed, it may be the case that each application has its own assertion policy, tailored to the details (e.g. sensitivity,

community, etc) of that application. When the assertion policy is communicated to the certificate Issuer, the burden of determining whether or not a particular assertion is appropriate for an application is removed from the relying party and placed on the assertion Issuer; it is they who will have to decide whether or not they should issue a assertion under a specific policy - this based on their (presumably) in depth understanding of their processes and practices.

5. Existing Mechanisms

5.1 RFC-2527

Public key certificates can be characterized by a number of different aspects – including the nature of the registration process that preceded their issuance, how active is their life cycle management, and the level of protection afforded to the associated private key.

The relevance of a particular certificate to a given application may depend on these characteristics, as they will together determine the level of assurance that a relying party can place in that certificate. Other factors that may impact whether or not a certificate is appropriate to a given application include: the level of liability the issuing CA is willing to assume if the certificate is shown to be fraudulent and the obligations of entities (subject's and relying parties) who may use that certificate.

RFC-2527 [RFC2527] defines a framework for expressing this information in two related but separate documents - the Certificate Policy (CP) and Certificate Practices Statement (CPS) – a common (but not universally accepted) distinction is that the CP contains **what** is to be adhered to, while the CPS states **how** it is adhered to. In this interpretation then, the CP is thus seen as more fundamental; a Relying Party decision to trust a particular certificate is ultimately determined by the assertions that the CA makes with respect to the CP requirements, the Relying Party could choose not to concern itself with the details of how the CA meets these requirements.

Another interpretation of the difference between the CPS and the CP is that the former is "owned" by the issuing certification authority and the latter by the entities that will use the issued certificates. Certificate users define CPs, and certification authorities (with different CPSs) attest that a particular certificate is conforms to that certificate CP.

While in principle there is clear separation between CPs and CPSs – in practice their roles become blurred.

5.2 PKI Disclosure Statement

The legal complexity and size (sometime on the order of 100 pages) of CPs and CPs prevent them from being readily read and understood; consequently they may not adequately support the necessary legal concept of clear 'disclosure' to be legally meaningful. This issue, as well as some CAs being reluctant to publish the details of their internal security practices for external review through a CPS, has motivated the specification of a simpler mechanism for policy disclosure – the so called PKI Disclosure Statement (PDS) [PDS] The PDS is a more succinct representation (typically two pages) of the most salient information of both the CP and the CPS.

While the PDS addresses the complexity of the CP and CPS model, it still suffers from many of the other limitations of these documents, Including

- There is no binding between the statements made within the PDS (some implying a legal obligation) and the CA. Consequently, the CA could later deny making those statements.
- It is not machine processable – preventing both automated discovery and interpretation.

- It does not define how it might be published to facilitate discovery by potential relying parties
- Reflecting its authority-centric nature, it is unable to provide detail on the applications for which a particular certificate class is appropriate - this because the authority may have little or no insight into the eventual business applications for which its certificates may be used. Consequently, although it is able to list the obligations of the actors who will use the certificates, it is unable to list these obligations against particular key applications.

6. Qualified Installation of Keys

The emerging Web Services architecture, built around XML and a publish-and-subscribe model, may offer a simpler alternative to the CP/CPS/PDS model for enabling efficient and inexpensive authority recognition. We propose such a model here as a possible solution.

We have named this model Qualified Installation of Keys (QIK), 'qualified' because the trust that a Relying Party may place in a particular assertion authority can be qualified by the applications for which it is appropriate, and 'installation' because a Relying Party decision to recognize an authority will often be made manifest in the installation of a public key for that authority into some 'trusted store'.

6.1 QIK model

The QIK model has an authority publish an XML document that captures the information on which a relying party will base, in part, its authority recognition decision. When asymmetric cryptography is the mechanism by which subsequently issued assertions can be authenticated & verified, the authority can publish its public verification key in the XML document as well. For this scenario, a relying party decision to recognize an authority's assertions for a particular application will be manifested as a decision to subsequently trust assertions signed by the associated private key.

QIK is an XML Schema by which the key can be published along with the key-owners commitments and associated conditions for use.

The basic principle behind QIK is a simple one, illustrated in Figure 6.

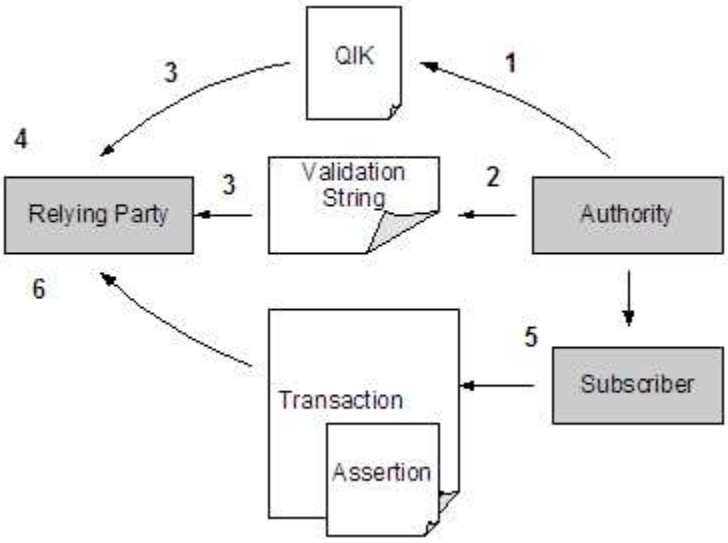

Figure 6: QIK process flow

The process proceeds by the following steps:

1. An assertion authority creates a XML-based QIK statement containing the policy statements for a given assertion category for which it issues assertions. It includes its public verification key within the QIK statement and uses the associated private key to sign the QIK statement. It publishes the QIK statement, either on the Web or by some other means, e.g. through WSDL or UDDI.
2. It creates a validation string by digesting the QIK statement and makes the digest available by an authentic channel, such as by email, telephone, company letterhead, business card, https, etc. (similar to PGP's fingerprint mechanism)
3. The relying party retrieves and validates the QIK statement, using the digest.
4. The relying party examines the QIK statement and confirms the suitability of its conditions of use to the intended application and, if these checks pass, installs the key – qualified for use by mapping to appropriate internal policies. It is expected that this decision will be accomplished through a combination of machine and human processing of the information within the QIK statement.
5. Subsequent to the installation of the key, the key owner issues signed assertions to a subscriber, who then sends these assertions to the relying party as part of some transaction.
6. The relying party validates the assertions using the previously installed public verification key and can be confident in the subscriber and the transaction.

6.2 Relying-party Key Installation

Upon receiving the QIK statement of the key-owning assertion authority, a relying-party must compare the approved uses and obligations within the QIK statement to both their intended application and their ability to satisfy the stated relying-party obligations in order to determine whether or not the key within the QIK statement is appropriate. This process is shown below:

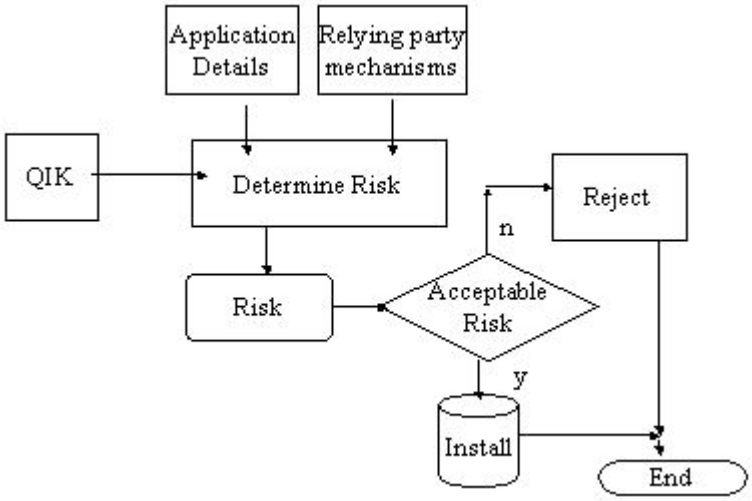

Figure 7: Key installation

This process would likely be a combination of machine processing and human interpretation – machine processing to determine if the QIK statement is compatible (i.e. does it allow the intended application, is the stated key owner liability limit acceptable) but the ultimate trust decision being made by some administrator.

It is expected that the key, once installed by the relying-party, would be indexed by the applications for which it would be appropriate. This is the qualified nature of the trust that the relying-party is able to place in the key; trust in the key is not absolute but relative to the nature of the application for which that key might be used in deriving security. When the relying-party subsequently receives a signed assertion from the key-owner (this message in the context of a specific business transaction) the relying-party is able to determine if the previously installed key is appropriate for this context. If so, the signature is considered to be valid for this context (beyond any revocation status checking etc) and the transaction is allowed to proceed; if not, the request is denied.

6.3 Variants

6.3.1 End-entity

The QIK model could be applied to end-entity's as well as assertion authorities. In this scenario, the end-entity key-owner would create a verification key-pair themselves and publish the public key within a QIK statement in order to facilitate establishment of bilateral trust with relying parties Alternatively, an end-entity key-owner could theoretically publish the public verification key extracted from certificate issued to them by a 3rd party CA. In this case, any policy commitments/obligations within the QIK statement could conflict with those of associated with the original certificate.

6.4 Multi-pass protocols

In the QIK protocol, as described, the key-owning assertion authority has no record of the relying-parties that have installed its key. In addition, the relying-party has no explicit acknowledgment from the key-owning assertion authority that they qualify as a relying-party. These features require a two- and three-step protocol respectively.

6.4.1 Two-step protocol

In the two-step protocol, the relying-party returns a message to the assertion authority. The message represents a formal acceptance by the relying party to be bound by the conditions expressed in the key-owner's QIK statement.

The two-step protocol could be achieved through either a Web services interface or a browser-based session.

6.4.2 Three-step protocol

The three-step protocol builds on the two-step protocol to provide the relying-party with a token formally acknowledging that the relying-party is acceptable to the assertion authority as a relying-party.

6.5 QIK Data model

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c
http://www.w3.org/2001/XMLSchema-->
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
     elementFormDefault = "qualified"
     attributeFormDefault = "unqualified">
    <xs:import namespace = "http://www.w3.org/2000/09/xmldsig#"
schemaLocation = "http://www.w3.org/TR/xmldsig-core/xmldsig-core-
schema.xsd"/>
    <xs:element name = "QikSpec" type = "QIkSpecType"/>
    <xs:complexType name = "QIkSpecType">
           <xs:sequence>
                  <xs:element ref = "ds:KeyInfo"/>
                  <xs:element ref = "KeyOwnerCategory"/>
                  <xs:element ref = "KeyOwnerIds"/>
                  <xs:element ref = "KeyApplications"/>
                  <xs:element ref = "KeyAcceptance"/>
                  <xs:element ref = "Enquiries"/>
                  <xs:element ref = "KeyStatus"/>
                  <xs:element ref = "DisputeResolution"/>
                  <xs:element ref = "ValidationStringLocator"/>
           </xs:sequence>
           <xs:attribute ref = "ID"/>
    </xs:complexType>
    <xs:element name = "KeyOwnerCategory" type =
"KeyOwnerCategoryType"/>
    <xs:element name = "KeyOwnerIds" type = "KeyOwnerIdsType"/>
    <xs:element name = "KeyApplications" type =
"KeyApplicationsType"/>
    <xs:element name = "KeyOwnerApplications" type =
"KeyOwnerApplicationsType"/>
    <xs:element name = "SubscriberApplications" type =
"SubscriberApplicationsType"/>
    <xs:element name = "KeyAcceptance" type = "KeyAcceptanceType"/>
    <xs:element name = "Enquiries" type = "EnquiriesType"/>
    <xs:element name = "KeyStatus" type = "KeyStatusType"/>
    <xs:element name = "ValidationStringLocator" type =
"ValidationStringLocatorType"/>
    <xs:element name = "DisputeResolution" type =
"DisputeResolutionType"/>
    <xs:complexType name = "KeyOwnerCategoryType">
           <xs:choice>
                  <xs:element ref = "Authority"/>
                  <xs:element ref = "End-user"/>
           </xs:choice>
```

```
    </xs:complexType>
    <xs:complexType name = "KeyOwnerIdsType">
           <xs:sequence>
                  <xs:element ref = "KeyOwnerId" maxOccurs =
"unbounded"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "KeyOwnerApplicationsType">
           <xs:sequence>
                  <xs:element ref = "KeyCommitments"/>
                  <xs:element ref = "KeyObligations"/>
                  <xs:choice>
                         <xs:element ref = "ExclusiveList"/>
                         <xs:element ref = "KeyApplication" maxOccurs =
"unbounded"/>
                  </xs:choice>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "SubscriberApplicationsType">
           <xs:sequence>
                  <xs:element ref = "KeyCommitments"/>
                  <xs:element ref = "KeyObligations"/>
                  <xs:choice>
                         <xs:element ref = "ExclusiveList"/>
                         <xs:element ref = "KeyApplication" maxOccurs =
"unbounded"/>
                  </xs:choice>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "KeyApplicationsType">
           <xs:sequence>
                  <xs:element ref = "KeyOwnerApplications"/>
                  <xs:element ref = "SubscriberApplications" minOccurs
= "0"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "KeyAcceptanceType">
           <xs:sequence>
                  <xs:element name = "KeyAcceptanceURI" type =
"xs:anyURI"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "EnquiriesType">
           <xs:sequence>
                  <xs:element name = "EnquiriesURI" type =
"xs:anyURI"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "KeyStatusType">
           <xs:sequence>
                  <xs:element name = "StatusURI" type = "xs:anyURI"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "DisputeResolutionType">
           <xs:sequence>
                  <xs:element name = "Arbitrator" type = "xs:string"
minOccurs = "0"/>
                  <xs:element name = "GoverningLaw" type = "xs:string"
minOccurs = "0"/>
           </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "ValidationStringLocatorType">
           <xs:sequence>
```

```
                <xs:element name = "Locator" type = "xs:anyURI"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name = "End-user" type = "End-userType"/>
        <xs:element name = "Authority">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref = "AuthorityType"/>
                    <xs:element ref = "LogoType"/>
                    <xs:element ref = "SubjectsType"/>
                    <xs:element ref = "SubjectsNamespace"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:complexType name = "End-userType"/>
        <xs:element name = "KeyOwnerId" type = "KeyOwnerIdType"/>
        <xs:complexType name = "KeyOwnerIdType">
            <xs:sequence>
                <xs:element ref = "IdType"/>
                <xs:element ref = "IdValue" minOccurs = "0" maxOccurs
= "unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name = "IdType" type = "xs:anyURI"/>
        <xs:element name = "IdValue" type = "IdValueType"/>
        <xs:complexType name = "IdValueType">
            <xs:sequence>
                <xs:any namespace =
"http://www.w3.org/2001/XMLSchema" processContents = "strict"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name = "AuthorityType" type = "AuthorityTypeType"/>
        <xs:element name = "LogoType" type = "LogoTypeType"/>
        <xs:element name = "SubjectsType" type = "SubjectsTypeType"/>
        <xs:element name = "SubjectsNamespace" type =
"SubjectsNamespaceType"/>
        <xs:complexType name = "AuthorityTypeType">
            <xs:choice>
                <xs:element ref = "Root"/>
                <xs:element ref = "Bridge"/>
                <xs:element ref = "Spoke"/>
            </xs:choice>
        </xs:complexType>
        <xs:complexType name = "LogoTypeType">
            <xs:sequence>
                <xs:any namespace =
"http://www.w3.org/2001/XMLSchema" processContents = "strict"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name = "SubjectType" type = "xs:anyURI"/>
        <xs:complexType name = "SubjectsTypeType">
            <xs:sequence>
                <xs:element ref = "SubjectType" maxOccurs =
"unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name = "SubjectsNamespaceType">
            <xs:sequence>
                <xs:element name = "SubjectNamespaceType" type =
"xs:anyURI" maxOccurs = "unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name = "Root" type = "RootType"/>
```

```
    <xs:element name = "Bridge" type = "BridgeType"/>
    <xs:element name = "Spoke" type = "SpokeType"/>
    <xs:complexType name = "RootType">
          <xs:sequence>
                <xs:element name = "HierarchyLevel" type =
"xs:nonNegativeInteger"/>
          </xs:sequence>
    </xs:complexType>
    <xs:complexType name = "BridgeType"/>
    <xs:complexType name = "SpokeType"/>
    <xs:complexType name = "KeyApplicationType">
          <xs:sequence>
                <xs:element ref = "KeyApplicationId"/>
                <xs:element ref = "Description"/>
                <xs:element ref = "KeyCommitments"/>
                <xs:element ref = "KeyObligations"/>
          </xs:sequence>
    </xs:complexType>
    <xs:element name = "KeyApplication" type = "KeyApplicationType"/>
    <xs:element name = "KeyApplicationId" type = "xs:anyURI"/>
    <xs:element name = "KeyCommitments" type = "KeyCommitmentsType"/>
    <xs:element name = "KeyObligations" type = "KeyObligationsType"/>
    <xs:complexType name = "KeyApplicationIdType"/>
    <xs:complexType name = "KeyCommitmentsType">
          <xs:choice maxOccurs = "unbounded">
                <xs:element ref = "KeyCommitment"/>
                <xs:element ref = "KeyCommitmentRef"/>
          </xs:choice>
    </xs:complexType>
    <xs:complexType name = "KeyObligationsType">
          <xs:choice maxOccurs = "unbounded">
                <xs:element ref = "KeyObligation"/>
                <xs:element ref = "KeyObligationRef"/>
          </xs:choice>
    </xs:complexType>
    <xs:complexType name = "KeyCommitmentType">
          <xs:sequence>
                <xs:element ref = "Description"/>
                <xs:element ref = "Liability" minOccurs = "0"
maxOccurs = "unbounded"/>
          </xs:sequence>
          <xs:attribute ref = "ID"/>
    </xs:complexType>
    <xs:complexType name = "KeyObligationType">
          <xs:sequence>
                <xs:element ref = "Description"/>
                <xs:element ref = "Liability" minOccurs = "0"
maxOccurs = "unbounded"/>
          </xs:sequence>
          <xs:attribute ref = "ID"/>
    </xs:complexType>
    <xs:element name = "KeyCommitment" type = "KeyCommitmentType"/>
    <xs:element name = "KeyObligation" type = "KeyObligationType"/>
    <xs:element name = "Description" type = "DescriptionType"/>
    <xs:complexType name = "DescriptionType">
          <xs:sequence>
                <xs:any namespace =
"http://www.w3.org/2001/XMLSchema" processContents = "strict"/>
          </xs:sequence>
    </xs:complexType>
    <xs:element name = "KeyCommitmentRef">
          <xs:complexType>
                <xs:attribute ref = "URI"/>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name = "KeyObligationRef">
            <xs:complexType>
                  <xs:attribute ref = "URI"/>
            </xs:complexType>
      </xs:element>
      <xs:attribute name = "URI" type = "xs:anyURI"/>
      <xs:attribute name = "ID" type = "xs:ID"/>
      <xs:element name = "ExclusiveList">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref = "KeyApplication" maxOccurs =
"unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name = "Liability">
            <xs:complexType>
                  <xs:attribute name = "minimum" use = "optional" type
= "xs:string"/>
                  <xs:attribute ref = "scope"/>
                  <xs:attribute ref = "maximum"/>
            </xs:complexType>
      </xs:element>
      <xs:attribute name = "maximum" type = "xs:integer"/>
      <xs:attribute name = "minimum" type = "xs:integer"/>
      <xs:attribute name = "scope">
            <xs:simpleType>
                  <xs:restriction base = "xs:string">
                        <xs:enumeration value = "perTransaction"/>
                        <xs:enumeration value = "aggregate"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:attribute>
</xs:schema>
```

6.6 X.509 Profile of QIK

6.6.1 Root import

The relying party is an end-entity and the authority recognition decision is whether or not the CA's public verification key should be imported into a 'Trusted Root' store.

6.6.2 Cross-certification

When the relying-party is a certification authority, a decision to assign some qualified trust to a public key contained within a QIK statement it has received from another CA may involve the relying-party CA issuing a cross-certificate to the key-owner CA. The issued cross-certificate would contain the key and other information extracted from the QIK statement, the QIK assertions as to commitments, obligations, and application usage mapped into the appropriate policy extensions within the issued cross-certificate. Once issued, the cross-certificate would be made available to the relying-party CA's community through existing mechanisms such that this community would be able to place qualified trust in the subscribers of the key-owner CA. This concept is shown in below.

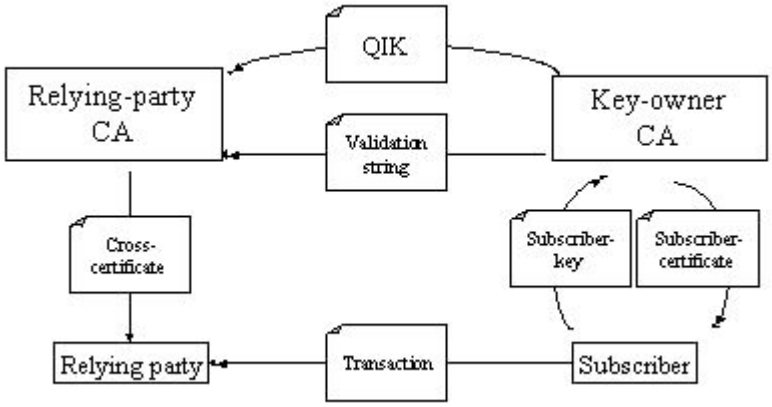

Figure 8: QIK enabled cross-certification

The CA's QIK statement describes the nature of the trust relationships that the certification authority engages in (at the time of the creation of the QIK statement). A relying party certification authority cross-certifying with the first certification authority may wish to constrain the trust that its relying-party community places in the cross-certified certification authority only to these existing relationships (and thereby prevent unexpected trust relationships). They can do so by issuing the cross-certificate with corresponding restrictions in the certificate extensions defined for this purpose (they may alternatively wish to constrain trust with even more restrictive settings)

6.7 Symmetric key systems and QIK

In a well-designed symmetric system of key management, the keys that are shared between pairs of parties are unique. For instance, a Kerberos [RFC1510] Key Distribution Center shares a unique symmetric key with each of its constituent clients and services, these shared symmetric keys used to securely distribute short-lived session keys to these clients and services. Therefore, there is no single key that can be distributed by publication. Similarly, the holder of a symmetric key cannot directly publish this key to potential relying-parties as this would necessarily mean that the required confidentiality of this secret key is lost.

Nevertheless, the basic principle behind QIK, that of publishing information about the warranties of a business entity with respect to transactions it engages in and the obligations of relying parties, can still be applied to symmetric-key management systems. The fundamental difference is that these assertions cannot be cryptographically bound to a particular key – this because symmetric key technology necessarily (or should) require that a unique key be created for every pair-wise relationship. Consequently, the owner of the QIK statement is unable to publish a key (or even a digest of a key) along with its QIK statement for consumption by a relying-party because no such key yet exists; if a QIK consumer decides to engage with the QIK owner, it is only then that key exchange will occur through some other secure channel.

7. Security Considerations

The discussion of security concerns is intrinsic to this document. Relying parties must ensure that they place trust only in assertions from valid and appropriate authorities, this usually achieved by ensuring that they place trust only in verification keys from those authorities. Consequently, the mechanism by which the authority's verification key is communicated to the relying party must be verifiable.

In the QIK model, the replying party must ensure that a QIK statement that purports to be from a particular authority is indeed from that authority, this achieved by ensuring that the calculated digest of the QIK statement matches the validation string retrieved through some authentic channel (e.g. from the authority's company letterhead or web site).

8. Author Information

David Chadwick
University of Kent
d.w.chadwick@kent.ac.uk

Paul Madsen
NTT
Ottawa, Canada
paulmadsen@rogers.com

9. Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

10. Full Copyright Notice

11. References

[RFC1510]
Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993.

[PDS]
PKIX working Group, May 10, 2000, Internet X.509 Public Key Infrastructure PKI Disclosure Statement: draft-ietf-pkix-pds-00.txt: http://www.verisign.com/repository/pds.txt

[RFC2527]
S. Chokhani, W. Ford, “Internet X.509 Public Key Infrastructure, Certificate Policy and Certification Practices Framework”, RFC 2527, March 1999

[SAML]
Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML), Version 2.0, March 15, 2005
http://www.oasis-open.org/apps/org/workgroup/security

[WSTRUST]
Web Services Trust Language, Version 1.1, May 2004,
ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf

[X509]
ITU-T Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.