What can you say and what does it mean?

Jon A. Solworth
University of Illinois at Chicago
851 S. Morgan Street, M/C 152
1120 SEO
Chicago IL 60607-7053

solworth@rites.uic.edu

Abstract

This paper examines some of the criteria for a certificate architecture that supports trusted collaboration. Key to this support are (1) the separation of statements from the actions they engender and (2) the ability to make arbitrary statements. The separation of statement from action enables organizations to set their own policies and therefore control what is authorized in response to a statement. The open ended nature to allow arbitrary statements to be created enables the signer to fully specify her intentions. It should be possible to make these statements arbitrarily precise, by tying the statements to their (semantic) contexts. Furthermore, we argue that it is important not only to support completely formalizable statements but also informal statements as well.

We then describe an architecture we are building which meets the above criteria and give examples of its efficacy.

1. Introduction

Trusted Collaboration must deal with entities which may trust each other to varying degrees. Trust by its nature is not universal, I may trust my government but not trust yours (or trust mine more) while you may trust your government but not mine. A trust architecture must therefore allow the organization to determine which entities it will trust and for what purposes. Hence, trust inherently involves both *authentication* (what is requested by a user) and *authorization* (what is allowed).

In the seminal paper *Authentication in Distributed Systems: Theory and Practice*, Lampson et al. defines a certificate as a (digitally) signed statement [LABW92]. The signature binds an entity (such as a user or computer) to a statement so that under reasonable conditions the statement cannot be repudiated by its apparent signer. And hence, the certificate can be used—even after the fact—to determine that a signer "says" a statement[1]. Lampson's definition of authentication is very close to its legal definition, in which authentication ties, for example, a will, a painting, or an historical document to its creator or signer. Hence, it seems that certificates are an ideal foundation on which to build trusted collaboration.

Fundamentally, signed statements will be used as justification by remote hosts that the actions they perform have been requested by the signer. A remote host which performs an action based on these statements can perform them without a proper certificate—just as, for example a driver's license might be issued without the proper documentation by the Department of Motor Vehicles—since it is the remote host that authorizes and then performs an action.

We note that the ability to make arbitrary, non-repudiatable statements is severely limited in non-distributed systems. This is because a computer which creates a statement cannot use it to prove that a user said it, since if it can create the statement *with* user approval then it can also create the statement *without* user approval[2]—and hence in non-distributed systems authentication is limited to its most basic form, identification. But in distributed systems, the signing computer and relying computer are distinct, and hence the relying computer cannot counterfeit statements from the signing computer. Interestingly, the protections become stronger with more mistrust between signing and relying systems, for example when these computers belong to different organizations, since it becomes more difficult for the relying computer to extract the keys or trick the signing computer. (This mirrors the physical world, where contracts are most important between organizations which distrust each other). In addition, as mistrust grows it becomes increasingly important to make the statements precise, and thus limiting what the relier is justified in doing.

Despite Lampson's et al. evocative characterization of certificates, certificates have traditionally only been used to make very limited types of statements. We believe that the limited expressiveness have curtailed the usefulness—and decreased the popularity—of certificates. More expressive certificates could form a ubiquitous building block for constructing secure distributed systems.

In Lampson's model, the statements ultimately are used to determine who can access a given object. His PKI describes the ability to delegate and allow access to objects; that statements are trusted if there is a chain leading from a user who has permission on the specified object to the use of that object; the logic determines which are legitimate statements; and then an access is allowed.

Several difficulties result from applying Lampson's model to trusted collaboration, including:

semantic power: because these systems are derived from DAC systems, there are properties that cannot be enforced, including information flow properties [BL76, Bib77].

semantic mismatch: the statements that can be made with their certificates are not those which are natural for a user to make. For example, a human may want to make a statement such as

[1] Of course, there are many assumptions needed for distributed authentication to be non-repudiatable, including trusting the computer that the user directly interacts with and the safeguarding of private keys.

[2] Once the computer has created one statement for a user, it can create arbitrarily more. We assume, of course, that it is impractical for the user to create a signed statement by manually performing cryptography.

"buy Ross Anderson's Security Engineering book from Amazon for $59.30". The variety of statements that a user might want to make is very large, see Ellison's SPKI requirements for a substantial enumeration of examples [Ell99].

delegation: far too much power is transferred to the delegatee since the delegator can only limit what happens by time, object accessed, or permission. Broad (unrestricted) delegations weaken non-repudiation since they provide more latitude for a delegatee to exceed the intentions of the principal.

frequency: low-level statements may be continuously required, overwhelming a human's ability to make them. To cope, the human increases delegation, but this reduces control.

structural dependencies: because certificates are specified in terms of system objects, the certificate signer must know what objects to specify. However, systems which are run by other organizations have internal structure which is not, and should not be, visible to outsiders. When interacting with these remote systems, the user doesn't care and cannot specify how something is done; the user cares only about its external effect.

Traditional certificate systems do not distinguish authorization from authentication. Here authentication ensures that the statements are made by the signer and are believed to be valid; these statements are the basis of performing actions on a remote system and hence must be authorized by that system's owner. This limits the way in which autonomous organizations can protect themselves, since they cannot control which statements (certificates) they will accept (that is, act upon). Hence, separating authorization from authentication can increase the quality of both, more precisely denoting what should be done, and thus providing greater security.

The problem with supporting arbitrary statements are two-fold, (1) What does the statement mean? and (2) What should happen as a result of some statements? Of course, for such certificates to be useful it must be possible to automate (2) which consists of (a) authorization and (b) actions to be performed. And this requires a new certificate system architecture.

In this paper we examine some of the consequences of making this separation and describe how we address the above issues in the design of the *SayAnything* Certificate Architecture. Our focus here is on the high level architecture of our system, a paper describing the implementation is in preparation.

The rest of the paper is organized as follows: Section 2 describes some of the issues we considered in designing SayAnything; and Section 3 describes our design. Section 4 illustrates the usefulness of our architecture through various examples. In Section 5, we describe related work and finally in Section 6 we conclude.

2. Design Considerations for a Certificate Architecture

In designing the SayAnything Certificate Architecture, a number of practical issues were considered:

Structured vs. non-structured statements If any type of statement can be written, then the statement must be written in English or some other natural language. Unfortunately, natural language statements cannot be automatically processed. If instead of a natural language, a structured form is used then flexibility is lost. How can these issues be reconciled?

Centralized vs. distributed authority Centralized authorities—such as the Internet Assigned Number Authority[3] (IANA), which assigns port numbers to services (among other things)—become cumbersome with (1) a growing number of entities and (2) protocols (certificates) used by a small number of organizations. On the other hand, a totally distributed means of tracking new types of statements (certificates), such as two organizations mutually numbering the certificates which they use, can result in confusion when the certificate is adopted by a third organization, whose certificate numbering is inconsistent with the newly adopted certificate.

Associating actions with statements Ultimately, the purpose of making statements in the context of a computer system are to cause or allow some actions to occur. In general, these actions cause state changes on remote machines. Hence, sequences of statements must automatically trigger actions on systems remote from the statement issuer.

Trust issues A certificate serves the trust needs of both the signer and the verifier. The signer wants to limit the actions that the verifier takes on her behalf. The verifier wants to ensure that he is justified in the actions he performs, so that the signer bears the responsibility for them.

Debugging, dispute resolution, and forensics A primary purpose of certificates is to prove what was said, and hence what should have happened. Therefore, certificates are a means for debugging distributed systems, for resolving disputes (and thus eliminating "he said", "she said" difficulties), as well as for providing logs for analyzing the system. Certificates must be comprehensible to signer and relier as well as dispute resolution arbitrators.

3. The SayAnything Certificate Architecture

The SayAnything Certificate Architecture uses two cryptographic primitives:

A digital signature over a text can be created only by the holder of a private key, who is presumably its owner. The corresponding publicly available key is used to *verify* that the signature does indeed match the text.

A (strongly) collision resistant one-way hash is used to identify a text. The one-wayness ensures that, for practical purposes, it is impossible from the hash code to determine the corresponding text. The (strong) collision resistance means that effectively it is impossible to find two different texts which hash to the same value. In the rest of the paper, we shall informally use the term one-way hash function to mean (strongly) collision resistent one-way hash function.

We next describe the SayAnything Certificate Architecture.

Certificate Meaning In the SayAnything Certificate Architecture each certificate is associated with two descriptions, its syntax and its semantics.

The syntax description is written in our *Certificate Definition Language (CDL)*, which is similar to an XML schema; it describes the certificate as a sequence of fields, each field having a name, a type, and a range of valid values. Each type is machine independent, so that for example an unsigned 32-bit integer (uint32) has the same properties on every computer architecture. From the syntactic description, a parser is generated for the certificate, so that the certificate's constituent fields can be extracted by name. In this paper, for brevity we will not use the CDL, but instead use a somewhat simplified tabular format to describe certificate syntax.

The schemes discussed here do not require certificates to be coded in CDL; they could be encoded as XML certificates or X509v3 certificates. XML has been used in trust system, see for instance [BFS04b] while X509v3 have been used in others [HMM$^+$00]. In fact, it is possible to support all three certificate types in the same system.

[3] `http://www.iana.org`

The semantic description contains the meaning of the certificate, it is parameterized with the names of fields of a particular certificate. It is modeled on blank legal forms, such as a real estate purchase form, which can be used by simply filling in its fields. The semantic description corresponds to the blank form; the certificate contains the *parameters* to fill in the blanks; and the syntactic definition allows the fields to be extracted from the certificate.

Other than some minor syntactic conventions to distinguish certificate parameters from other text in the semantic statement, no other requirement is placed on the semantics definition. In particular, there is no requirement that the statement is in any way reasonable.

Hence, some of the statements might make no sense whatsoever or might have unknown or unintended effect. But in return, the certificate form is completely general. SayAnything certificates can represent existing certificates, including formally defined ones. Arbitrary new statements (certificates) can be represented by creating new syntactic and semantic definitions. In addition to being an easily extensible system, we argue in the next section that this representation is also a fundamental benefit in representing trust.

We note that while it is possible to embed both the syntactic and semantic description in the certificate, it is undesirable to do so for at least two reasons. The first, and most obvious, reason is that it is awkward and may dramatically increase the size of the certificate. The second, is that the semantic definition may include information which is confidential. While it would be foolhardy to sign or rely upon a statement in which the meaning was unavailable, it is also desirable to limit who can view its semantic description.

Hence, the certificate incorporates both the syntactic and semantic definition by reference. We avoid the use of a naming authority by the simple expedient of using a strongly collision resistant one-way hash on the definition to compute the reference. This ensures, with overwhelming probability, that if two entities have the same syntactic and semantic hashes, then they have the same syntactic and semantic definition.

Certificate Structure In Figure 1, the *certificate header* syntax—that part which is given in every certificate—is shown in tabular form. In addition to the `size`, certificate encoding `version`, and the aforementioned `syntaxHash` and `semanticHash`, the certificate header contains fields which will be used in determining the certificate validity.

The `version` field can be used to create suitable cryptographic signatures for different certificate uses. For example, `version 1` might be used to sign short-term certificates (those for which use and adjudication would take less than a year) while `version 2` might be intended for use and adjudication for up to ten years, and use considerably larger (and hence more secure) signatures and hashes. Also key sizes and cryptographic algorithms must be changed as attacks become more effective, and hence versions will, of necessity, change over time.

The `revocationServerPublicKey` can either be (1) a key for signing revocation certificates or, (2) a key which is used to sign a statement that the current revocation key is k, and then k is used to sign the revocation certificate. The latter is particularly useful for long lived certificates, since it prevents "wearing out" of the private key corresponding to the `revocationServerPublicKey`.

Because the header is valid for all certificates, determining certificate validity is standardized. A certificate is valid exactly when all of the following conditions hold: (1) the current time is between `validFrom` and `validTo`; (2) the fields all contain valid values as specified in CDL; (3) if a revocation server is specified, it provides a non-revoked certificate; and (4) the signature verifies using the public key on the certificate. Invalid certificates obviously cannot be used as a justification for any action.

Authorization It is not necessary to understand the full meaning of a statement before either signing it or relying on it. People regularly sign statements they don't understand, and not just for unimportant things. For example, to buy a house you might enter into a mortgage agreement. The mortgage agreement is written in a specialized language which may be unfamiliar to you. Even if you understand the language its meaning cannot be interpreted independently of legislation and case law. Moreover, its meaning is subject to change, for example, by the outcome of a lawsuit.

Why would anyone sign such a statement? It would traditionally be reviewed by a lawyer who is believed to be independent of the party writing the contract. Much of the contract no doubt would deal with unlikely contingencies, and therefore not be of great concern. A signer would also rely on law to protect them to some degree from predatory practices and general experience that they know quite a few people who have signed mortgage agreements without ill effect. In short, they would rely on the (well-founded) belief that the mortgage document was unlikely to harm them.

For all these reasons, a full understanding of the meaning of statements is not only impossible, it is, in general, not even attempted by those either signing or relying on a statement. Of course, if the most important statements people sign are not fully understood and open ended, formal semantics to compute what will happen under all circumstances is not possible. (However, when possible formal semantics should be used, as it enhances the ability to understand and verify systems.) In general, we believe that it is better to state imprecisely what is meant rather than to be limited to only statements which can be made precisely.

There is another reason why fully formalizing statements can be inappropriate between entities which don't trust each other: One person's statements cannot compel another to rely on it. (There may be consequences for refusing to perform the implied action, but it is up to the relying organization to decide whether or not to do so.) The decision to accept or reject a statement might be based on who makes them (eg. "Dick always lies."), on what their subject is (eg. "Dick is always reliable except about the weather."), may have limits as to size (eg. "Dick can be trusted with amounts less than \$10."), or about external factors (eg. "Accept the withdrawal if the account balance covers the amount.").

Of course, this does not preclude the construction of formally defined system using SayAnything certificates. We could, for example, define an SDSI style certificate system by creating a certificate type for each of the SDSI certificates, and by specifying in the semantic statement its formalization.

Hence, use of SayAnything certificates must inherently deal with the question "What actions will be performed given a set of statements?" rather than "What do the statements mean?".

We note that while organizations make their own decisions as to what action to perform, these decision may be constrained by law or by agreement. For example, the organization may be legally prohibited from discrimination, or it may have signed a credit card agreement which requires it to accept credit cards under certain conditions. In this case, an abiding organization could use rules provided by third parties which would perform these actions.

The rule-based mechanism we are using is based on trust languages, and have been developed for distributed trust negotiation. These languages are used in attribute-based systems to grant permissions. Our use is a bit modified from this since we must (1) use it to produce certificates (for example, when one end is an on-line store) and (2) perform other actions. It enables the enforcement of strong authorization such as Mandatory Access Controls (MACs), which is not possible with other certificate architectures.

Architecture In Figure 2, the architectural flow and major components are specified. There are three inputs, a certificate's syntax description, a certificate's semantic description, and an organiza-

Field	Type	Description
`size`	uint64	size in bytes of the certificate
`version`	uint32	version describes remaining header fields hash and signature algorithms
`syntaxHash`	hashType	one-way hash of syntactic description
`semanticsHash`	hashType	one-way hash of semantic description
`publicKey`	publicKeyType	public key of the signer
`validFrom`	recentDateType	date and time from which certificate valid
`validTo`	recentDateType	date and time to which certificate valid
`revocationServer`	revocationServerType	address of revocation server
`revocationServerPublicKey`	publicKeyType	public key of revocation server
`signature`	signatureType	digital signature over all other certificate fields

Figure 1. Certificate header

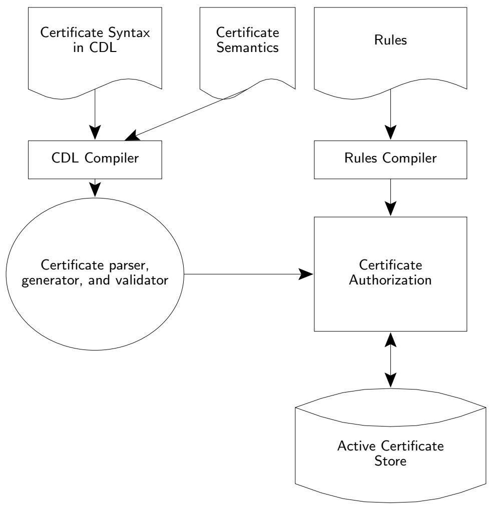
Certificate Syntax in CDL
Certificate Semantics
Rules
CDL Compiler
Rules Compiler
Certificate parser, generator, and validator
Certificate Authorization
Active Certificate Store

Figure 2. Architectural flow and major components

tion's authorization rules of what to do on the receipt of a certificate.

The syntax and semantic descriptions are input to the CDL compiler. The semantic description is only used to generate a hash. The syntax description is used to generate code which parses, generates, and validates certificates. The organization's rules are input to a *rules compiler* which produces the runtime engine for processing certificates. The runtime engine is invoked on each certificate arrival and for certificates which match a rule (1) some action is performed and (2) they are automatically stored in the current certificate database. This enables rule processing of a certificate to base its authorization decisions, in part, on past certificates.

The Rules description consists of a sequence of rules, each rule is of the following form:

```
Rule{cert}{cond}{action}
```

where `cert` is the name of the certificate, `cond` is the authorization conditions which must evaluate to true in order for the `action` to be performed.

We have completed the implementation of all components in Figure 2, except for the Rule Compiler which we recently began building. The components built so far comprise about 6,000 lines of code.

eula certificate	
field	type
`programFile`	string
`programHash`	hash

eulaAccept certificate	
field	type
`eulaHash`	hashType

Figure 3. Non-header fields of eula and eulaAccept certificates

4. Examples

Theoretical arguments are helpful for determining whether the architecture which we propose is appropriate. But a far greater measure is the *usefulness* of this architecture, which is illustrated next by a sequence of examples.

End-User Licence Agreement (EULA) Traditionally, each EULA for commercial software is handcrafted by lawyers; for open source software, there is also a large variety (although there is much more reuse of licenses than on the commercial side). Dealing with EULAs is much like dealing with mortgage agreements—the vast majority of people do not read them.

In a SayAnything Certificate, the EULA would be encoded in the certificate's semantic definition, the certificate would just name the software product and give a hash code over the file so that the file is uniquely defined by the `eula` certificate (see Figure 3). To develop an informed opinion on which licences are appropriate, one might go to one or more web sites which reviews such licences, and choose the licenses (semantic hashes) whose reviews matched acceptable policies. (For a corporation, there would be a more detailed list of semantic hashes perhaps combined with other information).

The rules for processing the `eula` certificate would ensure that the software installed on one's system matches the organization's policy, and would be significantly easier than today's "pop up" license. The licence could be obtained before purchasing the software. Another advantage of this scheme is that it would provide incentives to vendors for using standardized agreements, as they would be more readily acceptable to system owners.

Figure 4 contains a rule for the EULA certificate. There is a single certificate type, `eula`, and hence a single rule. The condition specifies the certificate version to be accepted, that the public key must be from a list of approved software vendors, and that the one-way hash of the program to be installed matches that specified on the certificates. The action taken is to accept the EULA (via signing a certificate, of course) and then to mark (label) the program file as installable.

```
Rule{eula}{eula.version==1 &
     eula.publicKey in ApprovedSoftwareVendors
  & eula.programHash==hash(eula.programFile)}
{
     ...set header fields of eulaAccept...
     eulaAccept.hash = hash(eula);
     sign(eulaAccept);
     label(programFile, installable);
}
```

Figure 4. EULA Acceptance conditions

rfq certificate	
field	type
`deliverByDate`	dateType
`terms`	hashType
`itemDescription`	hashType

quote certificate	
field	type
`deliverByDate`	dateType
`terms`	hashType
`itemDescription`	hashType
`quote`	usCurrency

purchase certificate	
field	type
`quote`	signatureType
`billingInfo`	hash

Figure 5. Non-header fields of rfq, quote, and purchase certificates

The `eulaAccept` certificate is "chained" to the `eula` certificate since one of its fields is the `eula` certificate's hash, which is guaranteed to be unique. This chaining prevents replay attacks where an attacker might try to use the `eulaAccept` certificate to claim that a different `eula` certificate had been accepted.

The set `ApprovedSoftwareVendors` could either be provided by external files or by one or more certificate services which can answer (via certificate) the question is $x \in y$. This makes it possible to use sets defined by external services. For example, an external rating organization might maintain a large and dynamic set of well reputed vendors.

In addition to these requirements, other requirements include determining the permissions an application has based on it vendor and on its purpose. This scheme enables the privileges to be determined based on these and other factors.

Transactional statements It is desirable to ensure that statements which request significant actions, such as a purchase order, are not used in the wrong context (misplay attacks) or re-used (replay attacks). One form of protection is to narrow the scope of the statement as much as possible to prevent misuse, such as naming the vendor, the item, having a tight time limit on when it is to be used, etc.

To prevent replay, the certificate should be revoked after first use. The mechanism for providing this is a *transactional revocation server*, which remembers the certificates it has seen and having seen a certificate before, revokes future presentations of that certificate. There is one technical point that enters here, and that is that the revocation server check should occur as the *last* check before accepting the certificate (after all of the rule's conditions) to prevent a certificate which will not actually be used from being invalidated. A revocation server's proof of validity (for first use) is a certificate signed by the revocation server and chained to the referenced certificate.

The certificates used in a simple transaction are shown in Figure 5. They consist of a purchaser *request for quote* `rfq` certificate, the vendor `quote` certificate, and then the purchaser's `purchase` certificate. The sequence of certificate exchanges is as follows:

1. The `rfq` specifies a delivery date, terms of the purchase, and a description of the item to be purchased. We assume that the terms and item description are contained in databases available to both parties, and hence they can be referred to by reference, i.e., by hash.
2. The vendor responds by providing a `quote` in `usCurrency`. Such a quote could have been crafted on the fly, but an alternative used here stores the quote certificates in the database and just returns the appropriate quote.
3. Typically, the best quote would be accepted and the purchase certificate would reference it by hash. There is also some billing information contained in the certificate.

delegate certificate	
field	type
`account`	accountId
`publicKey`	publicKeyType
`monthlyAmount`	usCurrency

revokeDelegation certificate	
field	type
`account`	accountId
`publicKey`	publicKeyType
`monthlyAmount`	usCurrency

delegateeCheck certificate	
field	type
`date`	dateType
`amount`	usCurrency
`fromAccount`	accountType
`toAccount`	accountType

Figure 6. Non-header fields of delegation, revokeDelegation, and delegateeCheck certificates

Delegation It is useful to be able to write arbitrarily limited delegations, or as they are called in the legal world, powers of attorney. For example, "allow Joe to write checks up to $500 per month". It is up to the bank to specify which delegation types will be accepted and under what conditions. The revocation server can be used to terminate such delegations, should it become necessary.

There are three certificate types used for the above delegation examples, as shown in Figure 6. The first certificate, `delegate`, establishes a delegation, with limits on the total dollar amount of checks which can be written per month. The second certificate, `revokeDelegation`, removes a delegation. The third certificate, `delegateeCheck`, is to write a check by the delegatee.

The `delegate` certificate typically would be established far in advance of the writing of the delegatee checks. Since all accepted certificates are stored in the organization's certificate store, the delegation certificate can be looked up as needed when a `delegateeCheck` certificate arrives.

Finally, the revoke certificate, once it is checked that it is properly formed, simply removes the delegate certificate from the active certificate store. Of course, all valid certificates should be kept until a dispute over its use is no longer possible.

```
Rule{rfq}{rfq.version==1 &
  m=bestMatch(rfq.item, rfq.deliverByDate,
               rfq.terms, rfq.publicKey)}
{
   send(m); // pre-signed certificate
}

Rule{purchase}{purchase.version==1 &
  find(purchase.quote)}
{
    label(purchase, accepted);
}
```

Figure 7. Transactional Rules

In Figure 8, the delegation rules are given. These rules are similar to those given in previous examples except for the `delegateeCheck` in which (1) the aforementioned certificate lookup is used to ensure that the delegation is in effect and (2) the balance information is checked to ensure that the delegation is not exceeded and that the balance is sufficient to cover the check. This second issue is not an authorization issue but is rather an application logic issue.

5. Related work

The notion of a certificate goes back to Kohnfelder [Koh78], and is, of course, built upon public key cryptography. Kohnfelder's certificates were used to bind a name to a public key, so that statements signed by the corresponding private key could be verified as coming from the named entity.

One of the earliest distributed authentication systems is Kerberos [MNSS87, KNT91]. Kerberos provides user authentication and cryptographic protection of communications between clients and servers. Because of the relative low performance of computers at the time it was developed, Kerberos used symmetric (secret key) rather than public key encryption. Kerberos provides a Ticket Granting Ticket upon password authentication, and that ticket can be used for a period of time to obtain other tickets for specific services. These tickets act as certificates and are used to ensure that there is a chain from the user to the use of a resource on which that user has some permission.

Lampson et al. [LABW92] provided a logic for reasoning about authentication. One of the systems described by this logic was a distributed system built at Digital Equipment Corporation. This system used a hierarchical structure to describe trusted sources of information, and has many similarities to Kerberos including the use of private key to simulate public key (assuming a fully trusted intermediary).

X.509 are the most popular certificates. They are considerably narrower in purpose than Lampson's certificates since they are used only to bind a public key to a description. X.509 certificates are used in SSL to identify the servers (typically a corporation willing to pay for a certificate from a CA); clients are not expected to have certificates and hence authentication is not mutual.

PGP, an email authentication system, [ASZ96] introduced a web of trust, in which each user could specify other users that they would rely upon for trust information. The web of trust allows every entity to determine which other entities can be used to build more extensive trust relationships than otherwise possible.

Rivest and Lampson proposed a certificate system called SDSI [RL96], for Simple Distributed System Infrastructure, which latter became known as SPKI, for Simple Public Key Infrastructure (it is also called SDSI 2) [EFL$^+$99]. SDSI's name space is a graph in which names are specified by the path (names of edges) from a current node, thus avoiding the problems of global name spaces including both trust issues and ambiguity of names. The SDSI graph name space builds on the PGP web of trust. SDSI is oriented towards a DAC design model, whose ultimate permission is determined by access control lists on objects.

Blaze et. al. introduced KeyNote [BFL96, BFIK99], and coined the phrase trust management system. KeyNote departs from the DAC orientation of earlier systems and instead relies on checking that the credentials of a principal match some criteria for using a network service. KeyNote allows the specification of which authorities can be used for which purposes.

KeyNote contains (1) credentials, (2) policy statements, and (3) trust statements. The credentials have a domain in which they are meaningful. Those with universal meaning would have to be allocated by IANA. KeyNote describes the inadequacy of using global certificate authorities, and provides mechanisms so that the certificate depends only on the organizations (i.e. domains) in which the certificate is used. We have already noted that this is awkward to manage and change for multi-organizational, but non-global, domains.

A large number of trust management languages have been proposed, see for example [BS00, WYS$^+$02, Jim01, HMM$^+$00, LM03]. The trust management examples in this paper are simpler than many of these other proposals, primarily because we are not concerned with the type privacy issues often explored (where both a user's credentials and the organization's policy are not intended to be disclosed). Hence, the examples used here allow for a fixed exchange a certificates, resulting in a very efficient processing scheme. For a comparison of trust management systems see [BFS04a] and [SWY$^+$02].

We note that the trust management language is local to the organization that employs it, and hence different organizations could use different trust management languages. These systems are interoperable if they have the same certificates exchanged and the correct local action performed.

Park and Sandhu [PS00] have investigated the relative merits of different ways of binding identity and attribute certificates using monolithic, autonomic, and chained signatures.

Crispo and Lomas investigated accountability issues having to do with certificate authorities and how to use separation of duty in order to reduce the assumptions needed for non-repudiation [CL96].

Li et al. have described a flexible delegation logic for determining attributes and proving safety in a distributed system [LGF03, LWM03].

Another interesting line of work concerns proving that a certificate says something without disclosing the (entire) contents of the certificate. These mechanisms are primarily oriented towards attribute certificates, for example, see [LL05, BCC04]. We have not yet looked into providing these facilities within the SayAnything Certificate Architecture, although we expect to in the future.

6. Conclusions

We believe that a trusted collaboration architecture will need to support a wide range of statements. With a large and extensible set of statements, it is impractical to centrally manage them.

The ability to make just the right statement protects both the signer of the statement and the relier upon the statement. The signer is sure that the ability to use the statement can be suitably limited, thus reducing vulnerability in making such statements. The relier has non-repudiatable statements to justify in the actions she takes. The overall system becomes more efficient because dispute resolution becomes more effective and system use is increased.

```
Rule{delegate}{delegate.version==1 &
  owner(delegate.account)==delegate.publicKey))
{
  ...send ok...
}

Rule{delegateeCheck}{delegateeCheck.version==1 &
  delegation=find(delegateeCheck.fromAccount, delegateeCheck.publicKey)}
{
  monthlySpend = delegateeCheck.amount +
                 accountSpend(delegateeCheck.fromAccount,
                              delegateeCheck.publicKey);
  if (monthlySpend<delegation.monthlyLimit)
    ...check balance and allow/deny the check...
}

Rule{revokeDelegation}{revokeDelegation.version==1 &
  delegation=find(revokeDelegation(revokeDelegation.account,
                  revokeDelegation.publicKey) &
  owner(delegation.acount)==delegate.publicKey}
{
   moveToInactive(delegation);
}
```

Figure 8. Delegation rules

Providing such an extensible system comes at a cost; we cannot expect all of the statements to have a formal semantics. This is a benefit as well since more statements can be expressed—as not all statements that are possible or important to express are formalizable. To allow certificates to be automatically processed, without requiring a formal meaning, it is necessary to associate actions with the certificates. Because these actions are, in general, policy decisions made by organizations the rules of the organization must be encoded to perform these operations.

The SayAnything Certificate Architecture implements these properties. It uses (strong) collision resistant one-way hash functions to provide a completely distributed and succinct encoding of the semantics of a certificate. It uses a rule-based language—based on attribute-based trust management systems—to provide both authorization (what is allowed) and the actions to be performed. Thus, authentication (the producing and verification of certificates) is cleanly separated from authorization (the acceptance and acting upon certificates).

We are planning on building a variety of distributed systems based on the SayAnything Certificate Architecture and will make our software freely available so that others can use it. We believe that this type of software can become a ubiquitous building block for secure distributed systems.

References

[ASZ96] D. Atkins, W. Stallings, and P. Zimmermann. RFC 1991: PGP message exchange formats, August 1996. Status: INFORMATIONAL.

[BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *11th ACM Conference on Computer and Communications Security (CCS)*. ACM SIGSAC, 2004.

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. RFC 2704: The KeyNote Trust-Management System Version 2, September 1999.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proc. IEEE Symp. Security and Privacy*, 1996.

[BFS04a] Elisa Bertino, Elena Ferrari, and Anna Squicciarini. Trust negotiations: Concepts, systems, and languages. *Computing in Science and Engg.*, 6(4):27–34, 2004.

[BFS04b] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-$\mathcal{X}$: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, July 2004.

[Bib77] K. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, MITRE Corp, Bedford, MA, 1977.

[BL76] D. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE Corp., Bedford, MA, July 1976.

[BS00] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2000. ACM Press.

[CL96] Bruno Crispo and T. Mark A. Lomas. A certification scheme for electronic commerce. In *Security Protocols Workshop*, pages 19–32, 1996.

[EFL$^+$99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. RFC 2693: SPKI certificate theory, September 1999.

[Ell99] Carl M. Ellison. *RFC 2692: SPKI requirements*. The Internet Society, September 1999.

[HMM$^+$00] Amir Herzberg, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, page 2, Washington, DC, USA, 2000. IEEE Computer Society.

[Jim01] Trevor Jim. Sd3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P 2001)*, page 106, Washington, DC, USA, 2001. IEEE Computer Society.

[KNT91] John T. Kohl, B. Clifford Neuman, and Theodore Y. Ts'o. The evolution of the kerberos authentication service. In *Proceedings of the Spring EurOpen'91 Conference*, Tromso, 1991.

[Koh78] Loren M. Kohnfelder. Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman, May 1978.

[LABW92] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[LGF03] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *j-TISSEC*, 6(1):128–171, February 2003.

[LL05] Jiangtao Li and Ninghui Li. OACerts: Oblivious attribute certificates. In *ACNS*, pages 301–317, 2005.

[LM03] Ninghui Li and John C. Mitchell. A role-based trust-management framework. *discex*, 01:201, January 19 2003.

[LWM03] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proc. IEEE Symp. Security and Privacy*, 2003.

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Technical report, 1987.

[PS00] Joon S. Park and Ravi S. Sandhu. Binding identities and attributes using digitally signed certificates. In *ACSAC*, pages 120–127. IEEE Computer Society, 2000.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI — a simple distributed security infrastucture. Technical report, MIT, April 1996.

[SWY$^+$02] K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, 2002.

[WYS$^+$02] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.