Network Working Group
Request for Comments: 2693
Category: Experimental

C. Ellison
Intel
B. Frantz
Electric Communities
B. Lampson
Microsoft
R. Rivest
MIT Laboratory for Computer Science
B. Thomas
Southwestern Bell
T. Ylonen
SSH

September 1999

SPKI Certificate Theory

Status of this Memo

Copyright Notice

Abstract

The SPKI Working Group has developed a standard form for digital certificates whose main purpose is authorization rather than authentication. These structures bind either names or explicit authorizations to keys or other objects. The binding to a key can be directly to an explicit key, or indirectly through the hash of the key or a name for it. The name and authorization structures can be used separately or together. We use S-expressions as the standard format for these certificates and define a canonical form for those S-expressions. As part of this development, a mechanism for deriving authorization decisions from a mixture of certificate types was developed and is presented in this document.

This document gives the theory behind SPKI certificates and ACLs without going into technical detail about those structures or their uses.

Table of Contents

1. Overview of Contents

This document contains the following sections:

Section 2: history of name certification, from 1976 on.

Section 3: discussion of authorization, rather than authentication, as the desired purpose of a certificate.

Section 4: discussion of delegation.

Section 5: discussion of validity conditions: date ranges, CRLs, re-validations and one-time on-line validity tests.

Section 6: definition of 5-tuples and their reduction.

Section 7: discussion of key management.

Section 8: security considerations.

The References section lists all documents referred to in the text as well as readings which might be of interest to anyone reading on this topic.

The Acknowledgements section, including a list of contributors primarily from the start of the working group. [The archive of working group mail is a more accurate source of contributor information.]

The Authors' Addresses section gives the addresses, telephone numbers and e-mail addresses of the authors.

1.1 Glossary

We use some terms in the body of this document in ways that could be specific to SPKI:

ACL: an Access Control List: a list of entries that anchors a certificate chain. Sometimes called a "list of root keys", the ACL is the source of empowerment for certificates. That is, a certificate communicates power from its issuer to its subject, but the ACL is the source of that power (since it theoretically has the owner of the resource it controls as its implicit issuer). An ACL entry has potentially the same content as a certificate body, but has no Issuer (and is not signed). There is most likely one ACL for each resource owner, if not for each controlled resource.

CERTIFICATE: a signed instrument that empowers the Subject. It contains at least an Issuer and a Subject. It can contain validity conditions, authorization and delegation information. Certificates come in three categories: ID (mapping <name,key>), Attribute (mapping <authorization,name>), and Authorization (mapping <authorization,key>). An SPKI authorization or attribute certificate can pass along all the empowerment it has received from the Issuer or it can pass along only a portion of that empowerment.

ISSUER: the signer of a certificate and the source of empowerment that the certificate is communicating to the Subject.

KEYHOLDER: the person or other entity that owns and controls a given private key. This entity is said to be the keyholder of the keypair or just the public key, but control of the private key is assumed in all cases.

PRINCIPAL: a cryptographic key, capable of generating a digital signature. We deal with public-key signatures in this document but any digital signature method should apply.

SPEAKING: A Principal is said to "speak" by means of a digital signature. The statement made is the signed object (often a certificate). The Principal is said to "speak for" the Keyholder.

SUBJECT: the thing empowered by a certificate or ACL entry. This can be in the form of a key, a name (with the understanding that the name is mapped by certificate to some key or other object), a hash of some object, or a set of keys arranged in a threshold function.

S-EXPRESSION: the data format chosen for SPKI/SDSI. This is a LISP-like parenthesized expression with the limitations that empty lists are not allowed and the first element in any S-expression must be a string, called the "type" of the expression.

THRESHOLD SUBJECT: a Subject for an ACL entry or certificate that specifies K of N other Subjects. Conceptually, the power being transmitted to the Subject by the ACL entry or certificate is transmitted in (1/K) amount to each listed subordinate Subject. K of those subordinate Subjects must agree (by delegating their shares along to the same object or key) for that power to be passed along. This mechanism introduces fault tolerance and is especially useful in an ACL entry, providing fault tolerance for "root keys".

2. Name Certification

Certificates were originally viewed as having one function: binding names to keys or keys to names. This thought can be traced back to the paper by Diffie and Hellman introducing public key cryptography in 1976. Prior to that time, key management was risky, involved and costly, sometimes employing special couriers with briefcases handcuffed to their wrists.

Diffie and Hellman thought they had radically solved this problem. "Given a system of this kind, the problem of key distribution is vastly simplified. Each user generates a pair of inverse transformations, E and D, at his terminal. The deciphering transformation, D, must be kept secret but need never be communicated on any channel. The enciphering key, E, can be made public by placing it in a public directory along with the user's name and address. Anyone can then encrypt messages and send them to the user, but no one else can decipher messages intended for him." [DH]

This modified telephone book, fully public, took the place of the trusted courier. This directory could be put on-line and therefore be available on demand, worldwide. In considering that prospect, Loren Kohnfelder, in his 1978 bachelor's thesis in electrical engineering from MIT [KOHNFELDER], noted: "Public-key communication works best when the encryption functions can reliably be shared among

the communicants (by direct contact if possible). Yet when such a reliable exchange of functions is impossible the next best thing is to trust a third party. Diffie and Hellman introduce a central authority known as the Public File."

2.1 First Definition of CERTIFICATE

Kohnfelder then noted, "Each individual has a name in the system by which he is referenced in the Public File. Once two communicants have gotten each other's keys from the Public File they can securely communicate. The Public File digitally signs all of its transmissions so that enemy impersonation of the Public File is precluded." In an effort to prevent performance problems, Kohnfelder invented a new construct: a digitally signed data record containing a name and a public key. He called this new construct a CERTIFICATE. Because it was digitally signed, such a certificate could be held by non-trusted parties and passed around from person to person, resolving the performance problems involved in a central directory.

2.2 The X.500 Plan and X.509

Ten years after Kohnfelder's thesis, the ISO X.509 recommendation was published as part of X.500. X.500 was to be a global, distributed database of named entities: people, computers, printers, etc. In other words, it was to be a global, on-line telephone book. The organizations owning some portion of the name space would maintain that portion and possibly even provide the computers on which it was stored. X.509 certificates were defined to bind public keys to X.500 path names (Distinguished Names) with the intention of noting which keyholder had permission to modify which X.500 directory nodes. In fact, the X.509 data record was originally designed to hold a password instead of a public key as the record-access authentication mechanism.

The original X.500 plan is unlikely ever to come to fruition. Collections of directory entries (such as employee lists, customer lists, contact lists, etc.) are considered valuable or even confidential by those owning the lists and are not likely to be released to the world in the form of an X.500 directory sub-tree. For an extreme example, imagine the CIA adding its directory of agents to a world-wide X.500 pool.

The X.500 idea of a distinguished name (a single, globally unique name that everyone could use when referring to an entity) is also not likely to occur. That idea requires a single, global naming discipline and there are too many entities already in the business of defining names not under a single discipline. Legacy therefore militates against such an idea.

2.3 X.509, PEM and PGP

The Privacy Enhanced Mail [PEM] effort of the Internet Engineering Task Force [RFC1114] adopted X.509 certificates, but with a different interpretation. Where X.509 was originally intended to mean "the keyholder may modify this portion of the X.500 database", PEM took the certificate to mean "the key speaks for the named person". What had been an access control instrument was now an identity instrument, along the lines envisioned by Diffie, Hellman and Kohnfelder.

The insistence on X.509 certificates with a single global root delayed PEM's adoption past its window of viability. RIPEM, by Mark Riordan of MSU, was a version of PEM without X.509 certificates. It was distributed and used by a small community, but fell into disuse. MOSS (a MIME-enhanced version of PEM, produced by TIS (www.tis.com)) made certificate use optional, but received little distribution.

At about the same time, in 1991, Phil Zimmermann's PGP was introduced with a different certificate model. Instead of waiting for a single global root and the hierarchy of Certificate Authorities descending from that root, PGP allowed multiple, (hopefully) independent but not specially trusted individuals to sign a <name,key> association, attesting to its validity. The theory was that with enough such signatures, that association could be trusted because not all of these signer would be corrupt. This was known as the "web of trust" model. It differed from X.509 in the method of assuring trust in the <name,key> binding, but it still intended to bind a globally unique name to a key. With PEM and PGP, the intention was for a keyholder to be known to anyone in the world by this certified global name.

2.4 Rethinking Global Names

The assumption that the job of a certificate was to bind a name to a key made sense when it was first published. In the 1970's, people operated in relatively small communities. Relationships formed face to face. Once you knew who someone was, you often knew enough to decide how to behave with that person. As a result, people have reduced this requirement to the simply stated: "know who you're dealing with".

Names, in turn, are what we humans use as identifiers of persons. We learn this practice as infants. In the family environment names work as identifiers, even today. What we learn as infants is especially difficult to re-learn later in life. Therefore, it is natural for people to translate the need to know who the keyholder is into a need to know the keyholder's name.

Computer applications need to make decisions about keyholders. These decisions are almost never made strictly on the basis of a keyholder's name. There is some other fact about the keyholder of interest to the application (or to the human being running the application). If a name functions at all for security purposes, it is as an index into some database (or human memory) of that other information. To serve in this role, the name must be unique, in order to serve as an index, and there must be some information to be indexed.

The names we use to identify people are usually unique, within our local domain, but that is not true on a global scale. It is extremely unlikely that the name by which we know someone, a given name, would function as a unique identifier on the Internet. Given names continue to serve the social function of making the named person feel recognized when addressed by name but they are inadequate as the identifiers envisioned by Diffie, Hellman and Kohnfelder.

In the 1970's and even through the early 1990's, relationships formed in person and one could assume having met the keyholder and therefore having acquired knowledge about that person. If a name could be found that was an adequate identifier of that keyholder, then one might use that name to index into memories about the keyholder and then be able to make the relevant decision.

In the late 1990's, this is no longer true. With the explosion of the Internet, it is likely that one will encounter keyholders who are complete strangers in the physical world and will remain so. Contact will be made digitally and will remain digital for the duration of the relationship. Therefore, on first encounter there is no body of knowledge to be indexed by any identifier.

One might consider building a global database of facts about all persons in the world and making that database available (perhaps for a fee). The name that indexes that database might also serve as a globally unique ID for the person referenced. The database entry under that name could contain all the information needed to allow someone to make a security decision. Since there are multiple decision-makers, each interested in specific information, the database would need to contain the union of multiple sets of information. However, that solution would constitute a massive privacy violation and would probably be rejected as politically impossible.

A globally unique ID might even fail when dealing with people we do know. Few of us know the full given names of people with whom we deal. A globally unique name for a person would be larger than the full given name (and probably contain it, out of deference to a

person's fondness for his or her own name). It would therefore not be a name by which we know the person, barring a radical change in human behavior.

A globally unique ID that contains a person's given name poses a special danger. If a human being is part of the process (e.g., scanning a database of global IDs in order to find the ID of a specific person for the purpose of issuing an attribute certificate), then it is likely that the human operator would pay attention to the familiar portion of the ID (the common name) and pay less attention to the rest. Since the common name is not an adequate ID, this can lead to mistakes. Where there can be mistakes, there is an avenue for attack.

Where globally unique identifiers need to be used, perhaps the best ID is one that is uniform in appearance (such as a long number or random looking text string) so that it has no recognizable sub-field. It should also be large enough (from a sparse enough name space) that typographical errors would not yield another valid identifier.

2.5 Inescapable Identifiers

Some people speak of global IDs as if they were inescapable identifiers, able to prevent someone from doing evil under one name, changing his name and starting over again. To make that scenario come true, one would have to have assignment of such identifiers (probably by governments, at birth) and some mechanism so that it is always possible to get from any flesh and blood person back to his or her identifier. Given that latter mechanism, any Certificate Authority desiring to issue a certificate to a given individual would presumably choose the same, inescapable name for that certificate. A full set of biometrics might suffice, for example, to look up a person without danger of false positive in a database of globally assigned ID numbers and with that procedure one could implement inescapable IDs.

The use of an inescapable identifier might be possible in some countries, but in others (such as the US) it would meet strong political opposition. Some countries have government-assigned ID numbers for citizens but also have privacy regulations that prohibit the use of those numbers for routine business. In either of these latter cases, the inescapable ID would not be available for use in routine certificates.

There was a concern that commercial Certificate Authorities might have been used to bring inescapable names into existence, bypassing the political process and the opposition to such names in those countries where such opposition is strong. As the (name,key)

certificate business is evolving today, there are multiple competing CAs each creating disjoint Distinguished Name spaces. There is also no real block to the creation of new CAs. Therefore a person is able to drop one Distinguished Name and get another, by changing CA, making these names not inescapable.

2.6 Local Names

Globally unique names may be politically undesirable and relatively useless, in the world of the Internet, but we use names all the time.

The names we use are local names. These are the names we write in our personal address books or use as nicknames or aliases with e-mail agents. They can be IDs assigned by corporations (e.g., bank account numbers or employee numbers). Those names or IDs do not need to be globally unique. Rather, they need to be unique for the one entity that maintains that address book, e-mail alias file or list of accounts. More importantly, they need to be meaningful to the person who uses them as indexes.

Ron Rivest and Butler Lampson showed with SDSI 1.0 [SDSI] that one can not only use local names locally, one can use local names globally. The clear security advantage and operational simplicity of SDSI names caused us in the SPKI group to adopt SDSI names as part of the SPKI standard.

2.6.1 Basic SDSI Names

A basic SDSI 2.0 name is an S-expression with two elements: the word "name" and the chosen name. For example,

```
george:  (name fred)
```

represents a basic SDSI name "fred" in the name space defined by george.

2.6.2 Compound SDSI Names

If fred in turn defines a name, for example,

```
fred:  (name sam)
```

then george can refer to this same entity as

```
george:  (name fred sam)
```

2.7 Sources of Global Identifiers

Even though humans use local names, computer systems often need globally unique identifiers. Even in the examples of section 2.6.2 above, we needed to make the local names more global and did so by specifying the name-space owner.

If we are using public key cryptography, we have a ready source of globally unique identifiers.

When one creates a key pair, for use in public key cryptography, the private key is bound to its owner by good key safeguarding practice. If that private key gets loose from its owner, then a basic premise of public key cryptography has been violated and that key is no longer of interest.

The private key is also globally unique. If it were not, then the key generation process would be seriously flawed and we would have to abandon this public key system implementation.

The private key must be kept secret, so it is not a possible identifier, but each public key corresponds to one private key and therefore to one keyholder. The public key, viewed as a byte string, is therefore an identifier for the keyholder.

If there exists a collision-free hash function, then a collision-free hash of the public key is also a globally unique identifier for the keyholder, and probably a shorter one than the public key.

2.8 Fully Qualified SDSI Names

SDSI local names are of great value to their definer. Each local name maps to one or more public keys and therefore to the corresponding keyholder(s). Through SDSI's name chaining, these local names become useful potentially to the whole world. [See section 2.6.2 for an example of SDSI name chaining.]

To a computer system making use of these names, the name string is not enough. One must identify the name space in which that byte string is defined. That name space can be identified globally by a public key.

It is SDSI 1.0 convention, preserved in SPKI, that if a (local) SDSI name occurs within a certificate, then the public key of the issuer is the identifier of the name space in which that name is defined.

However, if a SDSI name is ever to occur outside of a certificate, the name space within which it is defined must be identified. This gives rise to the Fully Qualified SDSI Name. That name is a public key followed by one or more names relative to that key. If there are two or more names, then the string of names is a SDSI name chain. For example,

(name (hash sha1 |TLCgPLFlGTzgUbcaYLW8kGTEnUk=|) jim therese)

is a fully qualified SDSI name, using the SHA-1 hash of a public key as the global identifier defining the name space and anchoring this name string.

2.9 Fully Qualified X.509 Names

An X.509 Distinguished Name can and sometimes must be expressed as a Fully Qualified Name. If the PEM or original X.500 vision of a single root for a global name space had come true, this wouldn't be necessary because all names would be relative to that same one root key. However, there is not now and is not likely ever to be a single root key. Therefore, every X.509 name should be expressed as the pair

(name <root key> <leaf name>)

if all leaf names descending from that root are unique. If uniqueness is enforced only within each individual CA, then one would build a Fully Qualified Name chain from an X.509 certificate chain, yielding the form

(name <root key> <CA(1)> <CA(2)> ... <CA(k)> <leaf name>).

2.10 Group Names

SPKI/SDSI does not claim to enforce one key per name. Therefore, a named group can be defined by issuing multiple (name,key) certificates with the same name -- one for each group member.

3. Authorization

Fully qualified SDSI names represent globally unique names, but at every step of their construction the local name used is presumably meaningful to the issuer. Therefore, with SDSI name certificates one can identify the keyholder by a name relevant to someone.

However, what an application needs to do, when given a public key certificate or a set of them, is answer the question of whether the remote keyholder is permitted some access. That application must make a decision. The data needed for that decision is almost never the spelling of a keyholder's name.

Instead, the application needs to know if the keyholder is authorized for some access. This is the primary job of a certificate, according to the members of the SPKI WG, and the SPKI certificate was designed to meet this need as simply and directly as possible.

We realize that the world is not going to switch to SPKI certificates overnight. Therefore, we developed an authorization computation process that can use certificates in any format. That process is described below in section 6.

The various methods of establishing authorization are documented below, briefly. (See also [UPKI])

3.1 Attribute Certificates

An Attribute Certificate, as defined in X9.57, binds an attribute that could be an authorization to a Distinguished Name. For an application to use this information, it must combine an attribute certificate with an ID certificate, in order to get the full mapping:

authorization -> name -> key

Presumably the two certificates involved came from different issuers, one an authority on the authorization and the other an authority on names. However, if either of these issuers were subverted, then an attacker could obtain an authorization improperly. Therefore, both the issuers need to be trusted with the authorization decision.

3.2 X.509v3 Extensions

X.509v3 permits general extensions. These extensions can be used to carry authorization information. This makes the certificate an instrument mapping both:

authorization -> key

and

name -> key

In this case, there is only one issuer, who must be an authority on both the authorization and the name.

Some propose issuing a master X.509v3 certificate to an individual and letting extensions hold all the attributes or authorizations the individual would need. This would require the issuer to be an authority on all of those authorizations. In addition, this aggregation of attributes would result in shortening the lifetime of the certificate, since each attribute would have its own lifetime. Finally, aggregation of attributes amounts to the building of a dossier and represents a potential privacy violation.

For all of these reasons, it is desirable that authorizations be limited to one per certificate.

3.3 SPKI Certificates

A basic SPKI certificate defines a straight authorization mapping:

authorization -> key

If someone wants access to a keyholder's name, for logging purposes or even for punishment after wrong-doing, then one can map from key to location information (name, address, phone, ...) to get:

authorization -> key -> name

This mapping has an apparent security advantage over the attribute certificate mapping. In the mapping above, only the

authorization -> key

mapping needs to be secure at the level required for the access control mechanism. The

key -> name

mapping (and the issuer of any certificates involved) needs to be secure enough to satisfy lawyers or private investigators, but a subversion of this mapping does not permit the attacker to defeat the access control. Presumably, therefore, the care with which these certificates (or database entries) are created is less critical than the care with which the authorization certificate is issued. It is also possible that the mapping to name need not be on-line or protected as certificates, since it would be used by human investigators only in unusual circumstances.

3.4 ACL Entries

SDSI 1.0 defined an ACL, granting authorization to names. It was then like an attribute certificate, except that it did not need to be signed or issued by any key. It was held in local memory and was assumed issued by the owner of the computer and therefore of the resource being controlled.

In SPKI, an ACL entry is free to be implemented in any way the developer chooses, since it is never communicated and therefore does not need to be standardized. However, a sample implementation is documented, as a certificate body minus the issuer field. The ACL entry can have a name as a subject, as in SDSI 1.0, or it can have a key as a subject. Examples of the latter include the list of SSL root keys in an SSL capable browser or the file .ssh/authorized_keys in a user's home UNIX directory. Those ACLs are single-purpose, so the individual entries do not carry explicit authorizations, but SPKI uses explicit authorizations so that one can use common authorization computation code to deal with multiple applications.

4. Delegation

One of the powers of an authorization certificate is the ability to delegate authorizations from one person to another without bothering the owner of the resource(s) involved. One might issue a simple permission (e.g., to read some file) or issue the permission to delegate that permission further.

Two issues arose as we considered delegation: the desire to limit depth of delegation and the question of separating delegators from those who can exercise the delegated permission.

4.1 Depth of Delegation

There were three camps in discussing depth of delegation: no control, boolean control and integer control. There remain camps in favor of each of these, but a decision was reached in favor of boolean control.

4.1.1 No control

The argument in favor of no control is that if a keyholder is given permission to do something but not the permission to delegate it, then it is possible for that keyholder to loan out the empowered private key or to set up a proxy service, signing challenges or requests for the intended delegate. Therefore, the attempt to restrict the permission to delegate is ineffective and might back-fire, by leading to improper security practices.

4.1.2 Boolean control

The argument in favor of boolean control is that one might need to specify an inability to delegate. For example, one could imagine the US Commerce Department having a key that is authorized to declare a cryptographic software module exportable and also to delegate that authorization to others (e.g., manufacturers). It is reasonable to assume the Commerce Department would not issue permission to delegate this further. That is, it would want to have a direct legal agreement with each manufacturer and issue a certificate to that manufacturer only to reflect that the legal agreement is in place.

4.1.3 Integer control

The argument in favor of integer control is that one might want to restrict the depth of delegation in order to control the proliferation of a delegated permission.

4.1.4 The choice: boolean

Of these three, the group chose boolean control. The subject of a certificate or ACL entry may exercise any permission granted and, if delegation is TRUE, may also delegate that permission or some subset of it to others.

The no control argument has logical appeal, but there remains the assumption that a user will value his or her private key enough not to loan it out or that the key will be locked in hardware where it can't be copied to any other user. This doesn't prevent the user from setting up a signing oracle, but lack of network connectivity might inhibit that mechanism.

The integer control option was the original design and has appeal, but was defeated by the inability to predict the proper depth of delegation. One can always need to go one more level down, by creating a temporary signing key (e.g., for use in a laptop). Therefore, the initially predicted depth could be significantly off.

As for controlling the proliferation of permissions, there is no control on the width of the delegation tree, so control on its depth is not a tight control on proliferation.

4.2 May a Delegator Also Exercise the Permission?

We decided that a delegator is free to create a new key pair, also controlled by it, and delegate the rights to that key to exercise the delegated permission. Therefore, there was no benefit from attempting to restrict the exercise of a permission by someone permitted to delegate it.

4.3 Delegation of Authorization vs. ACLs

One concern with defining an authorization certificate is that the function can be performed by traditional <authorization,name> ACLs and <name,key> ID certificates defining groups. Such a mechanism was described in SDSI 1.0. A new mechanism needs to add value or it just complicates life for the developer.

The argument for delegated authorization as opposed to ACLs can be seen in the following example.

Imagine a firewall proxy permitting telnet and ftp access from the Internet into a network of US DoD machines. Because of the sensitivity of that destination network, strong access control would be desired. One could use public key authentication and public key certificates to establish who the individual keyholder was. Both the private key and the keyholder's certificates could be kept on a Fortezza card. That card holds X.509v1 certificates, so all that can be established is the name of the keyholder. It is then the job of the firewall to keep an ACL, listing named keyholders and the forms of access they are each permitted.

Consider the ACL itself. Not only would it be potentially huge, demanding far more storage than the firewall would otherwise require, but it would also need its own ACL. One could not, for example, have someone in the Army have the power to decide whether someone in the Navy got access. In fact, the ACL would probably need not one level of its own ACL, but a nested set of ACLs, eventually reflecting the organization structure of the entire Defense Department.

Without the ACLs, the firewall could be implemented in a device with no mass storage, residing in a sealed unit one could easily hold in one hand. With the ACLs, it would need a large mass storage device that would be accessed not only while making access control decisions but also for updating the ACLs.

By contrast, let the access be controlled by authorization certificates. The firewall would have an ACL with one entry, granting a key belonging to the Secretary of Defense the right to delegate all access through the firewall. The Secretary would, in

turn, issue certificates delegating this permission to delegate to each of his or her subordinates. This process would iterate, until some enlisted man would receive permission to penetrate that firewall for some specific one protocol, but not have permission to delegate that permission.

The certificate structure generated would reflect the organization structure of the entire Defense Department, just as the nested ACLs would have, but the control of these certificates (via their issuance and revocation) is distributed and need not show up in that one firewall or be replicated in all firewalls. Each individual delegator of permission performs a simple task, well understood. The application software to allow that delegation is correspondingly simple.

5. Validity Conditions

A certificate, or an ACL entry, has optional validity conditions. The traditional ones are validity dates: not-before and not-after. The SPKI group resolved, in discussion, that on-line tests of various kinds are also validity conditions. That is, they further refine the valid date range of a certificate. Three kinds of on-line tests are envisioned: CRL, re-validation and one-time.

When validity confirmation is provided by some online test, then the issuer of those refinements need not be the issuer of the original certificate. In many cases, the business or security model for the two issuers is different. However, in SPKI, the certificate issuer must specify the issuer of validity confirmations.

5.1 Anti-matter CRLs

An early form of CRL [Certificate Revocation List] was modeled after the news print book that used to be kept at supermarket checkout stands. Those books held lists of bad checking account numbers and, later, bad credit card numbers. If one’s payment instrument wasn’t listed in the book, then that instrument was considered good.

These books would be issued periodically, and delivered by some means not necessarily taking a constant time. However, when a new book arrived, the clerk would replace the older edition with the new one and start using it. This design was suited to the constraints of the implementation: use of physical books, delivered by physical means. The book held bad account numbers rather than good ones because the list of bad accounts was smaller.

An early CRL design followed this model. It had a list of revoked certificate identifiers. It also had a sequence number, so that one could tell which of two CRLs was more recent. A newer CRL would replace an older one.

This mode of operation is like wandering anti-matter. When the issuer wants to revoke a certificate, it is listed in the next CRL to go out. If the revocation is urgent, then that CRL can be released immediately. The CRL then follows some dissemination process unrelated to the needs of the consumers of the CRL. If the CRL encounters a certificate it has listed, it effectively annihilates that certificate. If it encounters an older CRL, it annihilates that CRL also, leaving a copies of itself at the verifiers it encounters.

However, this process is non-deterministic. The result of the authorization computation is at least timing dependent. Given an active adversary, it can also be a security hole. That is, an adversary can prevent revocation of a given certificate by preventing the delivery of new CRLs. This does not require cryptographic level effort, merely network tampering.

SPKI has ruled out the use of wandering anti-matter CRLs for its certificates. Every authorization computation is deterministic, under SPKI rules.

5.2 Timed CRLs

SPKI permits use of timed CRLs. That is, if a certificate can be referenced in a CRL, then the CRL process is subject to three conditions.

1. The certificate must list the key (or its hash) that will sign the CRL and may give one or more locations where that CRL might be fetched.

2. The CRL must carry validity dates.

3. CRL validity date ranges must not intersect. That is, one may not issue a new CRL to take effect before the expiration of the CRL currently deployed.

Under these rules, no certificate that might use a CRL can be processed without a valid CRL and no CRL can be issued to show up as a surprise at the verifier. This yields a deterministic validity computation, independent of clock skew, although clock inaccuracies in the verifier may produce a result not desired by the issuer. The CRL in this case is a completion of the certificate, rather than a message to the world announcing a change of mind.

Since CRLs might get very large and since they tend to grow monotonically, one might want to issue changes to CRLs rather than full ones. That is, a CRL might be a full CRL followed by a sequence of delta-CRLs. That sequence of instruments is then treated as a current CRL and the combined CRL must follow the conditions listed above.

5.3 Timed Revalidations

CRLs are negative statements. The positive version of a CRL is what we call a revalidation. Typically a revalidation would list only one certificate (the one of interest), although it might list a set of certificates (to save digital signature effort).

As with the CRL, SPKI demands that this process be deterministic and therefore that the revalidation follow the same rules listed above (in section 5.2).

5.4 Setting the Validity Interval

Both timed CRLs and timed revalidations have non-0 validity intervals. To set this validity interval, one must answer the question: "How long are you willing to let the world believe and act on a statement you know to be false?"

That is, one must assume that the previous CRL or revalidation has just been signed and transmitted to at least one consumer, locking up a time slot. The next available time slot starts after this validity interval ends. That is the earliest one can revoke a certificate one learns to be false.

The answer to that question comes from risk management. It will probably be based on expected monetary losses, at least in commercial cases.

5.5 One-time Revalidations

Validity intervals of length zero are not possible. Since transmission takes time, by the time a CRL was received by the verifier, it would be out of date and unusable. That assumes perfect clock synchronization. If clock skew is taken into consideration, validity intervals need to be that much larger to be meaningful.

For those who want to set the validity interval to zero, SPKI defines a one-time revalidation.

This form of revalidation has no lifetime beyond the current authorization computation. One applies for this on-line, one-time revalidation by submitting a request containing a nonce. That nonce gets returned in the signed revalidation instrument, in order to prevent replay attacks. This protocol takes the place of a validity date range and represents a validity interval of zero, starting and ending at the time the authorization computation completes.

5.6 Short-lived Certificates

A performance analysis of the various methods of achieving fine-grain control over the validity interval of a certificate should consider the possibility of just making the original certificate short-lived, especially if the online test result is issued by the same key that issued the certificate. There are cases in which the short-lived certificate requires fewer signatures and less network traffic than the various online test options. The use of a short-lived certificate always requires fewer signature verifications than the use of certificate plus online test result.

If one wants to issue short-lived certificates, SPKI defines a kind of online test statement to tell the user of the certificate where a replacement short-lived certificate might be fetched.

5.7 Other possibilities

There are other possibilities to be considered when choosing a validity condition model to use.

5.7.1 Micali’s Inexpensive On-line Results

Silvio Micali has patented a mechanism for using hash chains to revalidate or revoke a certificate inexpensively. This mechanism changes the performance requirements of those models and might therefore change the conclusion from a performance analysis [ECR].

5.7.2 Rivest’s Reversal of the CRL Logic

Ron Rivest has written a paper [R98] suggesting that the whole validity condition model is flawed because it assumes that the issuer (or some entity to which it delegates this responsibility) decides the conditions under which a certificate is valid. That traditional model is consistent with a military key management model, in which there is some central authority responsible for key release and for determining key validity.

However, in the commercial space, it is the verifier and not the issuer who is taking a risk by accepting a certificate. It should therefore be the verifier who decides what level of assurance he needs before accepting a credential. That verifier needs information from the issuer, and the more recent that information the better, but the decision is the verifier's in the end.

This line of thought deserves further consideration, but is not reflected in the SPKI structure definition. It might even be that both the issuer and the verifier have stakes in this decision, so that any replacement validity logic would have to include inputs from both.

6. Tuple Reduction

The processing of certificates and related objects to yield an authorization result is the province of the developer of the application or system. The processing plan presented here is an example that may be followed, but its primary purpose is to clarify the semantics of an SPKI certificate and the way it and various other kinds of certificate might be used to yield an authorization result.

There are three kinds of entity that might be input to the computation that yields an authorization result:

1. <name,key> (as a certificate)

2. <authorization,name> (as an attribute certificate or ACL entry)

3. <authorization,key> (as an authorization certificate or ACL entry)

These entities are processed in three stages.

1. Individual certificates are verified by checking their signatures and possibly performing other work. They are then mapped to intermediate forms, called "tuples" here.

   The other work for SPKI or SDSI certificates might include processing of on-line test results (CRL, re-validation or one-time validation).

   The other work for PGP certificates may include a web-of-trust computation.

   The other work for X.509 certificates depends on the written documentation for that particular use of X.509 (typically tied to the root key from which the certificate descended) and could

involve checking information in the parent certificate as well as additional information in extensions of the certificate in question. That is, some use X.509 certificates just to define names. Others use X.509 to communicate an authorization implicitly (e.g., SSL server certificates). Some might define extensions of X.509 to carry explicit authorizations. All of these interpretations are specified in written documentation associated with the certificate chain and therefore with the root from which the chain descends.

If on-line tests are involved in the certificate processing, then the validity dates of those on-line test results are intersected by VIntersect() [defined in 6.3.2, below] with the validity dates of the certificate to yield the dates in the certificate's tuple(s).

2. Uses of names are replaced with simple definitions (keys or hashes), based on the name definitions available from reducing name 4-tuples.

3. Authorization 5-tuples are then reduced to a final authorization result.

6.1 5-tuple Defined

The 5-tuple is an intermediate form, assumed to be held in trusted memory so that it doesn't need a digital signature for integrity. It is produced from certificates or other credentials via trusted software. Its contents are the same as the contents of an SPKI certificate body, but it might be derived from another form of certificate or from an ACL entry.

The elements of a 5-tuple are:

1. Issuer: a public key (or its hash), or the reserved word "Self". This identifies the entity speaking this intermediate result.

2. Subject: a public key (or its hash), a name used to identify a public key, the hash of an object or a threshold function of subordinate subjects. This identifies the entity being spoken about in this intermediate result.

3. Delegation: a boolean. If TRUE, then the Subject is permitted by the Issuer to further propagate the authorization in this intermediate result.

4. Authorization: an S-expression. [Rules for combination of Authorizations are given below.]

5. Validity dates: a not-before date and a not-after date, where "date" means date and time. If the not-before date is missing from the source credential then minus infinity is assumed. If the not-after date is missing then plus infinity is assumed.

6.2 4-tuple Defined

A <name,key> certificate (such as X.509v1 or SDSI 1.0) carries no authorization field but does carry a name. Since it is qualitatively different from an authorization certificate, a separate intermediate form is defined for it.

The elements of a Name 4-tuple are:

1. Issuer: a public key (or its hash). This identifies the entity defining this name in its private name space.

2. Name: a byte string

3. Subject: a public key (or its hash), a name, or a threshold function of subordinate subjects. This defines the name.

4. Validity dates: a not-before date and a not-after date, where "date" means date and time. If the not-before date is missing from the source credential then minus infinity is assumed. If the not-after date is missing then plus infinity is assumed.

6.3 5-tuple Reduction Rules

The two 5-tuples:

<I1,S1,D1,A1,V1> + <I2,S2,D2,A2,V2>

yield

<I1,S2,D2,AIntersect(A1,A2),VIntersect(V1,V2)>

provided

the two intersections succeed,

S1 = I2

and

D1 = TRUE

If S1 is a threshold subject, there is a slight modification to this rule, as described below in section 6.3.3.

6.3.1 AIntersect

An authorization is a list of strings or sub-lists, of meaning to and probably defined by the application that will use this authorization for access control. Two authorizations intersect by matching, element for element. If one list is longer than the other but match at all elements where both lists have elements, then the longer list is the result of the intersection. This means that additional elements of a list must restrict the permission granted.

Although actual authorization string definitions are application dependent, AIntersect provides rules for automatic intersection of these strings so that application developers can know the semantics of the strings they use. Special semantics would require special reduction software.

For example, there might be an ftpd that allows public key access control, using authorization certificates. Under that service,

```
(ftp (host ftp.clark.net))
```

might imply that the keyholder would be allowed ftp access to all directories on ftp.clark.net, with all kinds of access (read, write, delete, ...). This is more general (allows more access) than

```
(ftp (host ftp.clark.net) (dir /pub/cme))
```

which would allow all kinds of access but only in the directory specified. The intersection of the two would be the second.

Since the AIntersect rules imply position dependency, one could also define the previous authorization string as:

```
(ftp ftp.clark.net /pub/cme)
```

to keep the form compact.

To allow for wild cards, there are a small number of special S-expressions defined, using "*" as the expression name.

(*)
: stands for the set of all S-expressions and byte-strings. In other words, it will match anything. When intersected with anything, the result is that other thing. [The AIntersect rule about lists of different length treats a

list as if it had enough (*) entries implicitly appended to it to match the length of another list with which it was being intersected.]

(* set <tag-expr>*)
stands for the set of elements listed in the *-form.

(* prefix <byte-string>)
stands for the set of all byte strings that start with the one given in the *-form.

(* range <ordering> <lower-limit>? <upper-limit>?)
stands for the set of all byte strings lexically (or numerically) between the two limits. The ordering parameter (alpha, numeric, time, binary, date) specifies the kind of strings allowed.

AIntersect() is normal set intersection, when *-forms are defined as they are above and a normal list is taken to mean all lists that start with those elements. The following examples should give a more concrete explanation for those who prefer an explanation without reference to set operations.

```
AIntersect( (tag (ftp ftp.clark.net cme (* set read write))),
            (tag (*)) )

evaluates to (tag (ftp ftp.clark.net cme (* set read write)))

AIntersect( (tag (* set read write (foo bla) delete)),
            (tag (* set write read) ) )

evaluates to (tag (* set read write))

AIntersect( (tag (* set read write (foo bla) delete)),
            (tag read ) )

evaluates to (tag read)

AIntersect( (tag (* prefix http://www.clark.net/pub/)),
            (tag (* prefix http://www.clark.net/pub/cme/html/)) )

evaluates to (tag (* prefix http://www.clark.net/pub/cme/html/))

AIntersect( (tag (* range numeric ge #30# le #39# )), (tag #26#) )

fails to intersect.
```

6.3.2 VIntersect

Date range intersection is straight-forward.

V = VIntersect(X, Y)

is defined as

Vmin = max(Xmin, Ymin)

Vmax = min(Xmax, Ymax)

and if Vmin > Vmax, then the intersection failed.

These rules assume that daytimes are expressed in a monotonic form, as they are in SPKI.

The full SPKI VIntersect() also deals with online tests. In the most straight-forward implementation, each online test to which a certificate is subject is evaluated. Each such test carries with it a validity interval, in terms of dates. That validity interval is intersected with any present in the certificate, to yield a new, current validity interval.

It is possible for an implementation of VIntersect() to gather up online tests that are present in each certificate and include the union of all those tests in the accumulating tuples. In this case, the evaluation of those online tests is deferred until the end of the process. This might be appropriate if the tuple reduction is being performed not for answering an immediate authorization question but rather for generation of a summary certificate (Certificate Result Certificate) that one might hope would be useful for a long time.

6.3.3 Threshold Subjects

A threshold subject is specified by two numbers, K and N [0<K<=N], and N subordinate subjects. A threshold subject is reduced to a single subject by selecting K of the N subjects and reducing each of those K to the same subject, through a sequence of certificates. The (N-K) unselected subordinate subjects are set to (null).

The intermediate form for a threshold subject is a copy of the tuple in which the threshold subject appears, but with only one of the subordinate subjects. Those subordinate tuples are reduced individually until the list of subordinate tuples has (N-K) (null) entries and K entries with the same subject. At that point, those K tuples are validity-, authorization- and delegation- intersected to yield the single tuple that will replace the list of tuples.

6.3.4 Certificate Path Discovery

All reduction operations are in the order provided by the prover. That simplifies the job of the verifier, but leaves the job of finding the correct list of reductions to the prover.

The general algorithm for finding the right certificate paths from a large set of unordered certificates has been solved[ELIEN], but might be used only rarely. Each keyholder who is granted some authority should receive a sequence of certificates delegating that authority. That keyholder may then want to delegate part of this authority on to some other keyholder. To do that, a single additional certificate is generated and appended to the sequence already available, yielding a sequence that can be used by the delegatee to prove access permission.

6.4 4-tuple Reduction

There will be name 4-tuples in two different classes, those that define the name as a key and those that define the name as another name.

1. [(name K1 N) -> K2]

2. [(name K1 N) -> (name K2 N1 N2 ... Nk)]

As with the 5-tuples discussed in the previous section, name definition 4-tuples should be delivered in the order needed by the prover. In that case, the rule for name reduction is to replace the name just defined by its definition. For example,

```
(name K1 N N1 N2 N3) + [(name K1 N) -> K2]

     -> (name K2 N1 N2 N3)
```

or, in case 2 above,

```
(name K1 N Na Nb Nc) + [(name K1 N) -> (name K2 N1 N2 ... Nk)]

     -> (name K2 N1 N2 ... Nk Na Nb Nc)
```

With the second form of name definition, one might have names that temporarily grow. If the prover is providing certificates in order, then the verifier need only do as it is told.

If the verifier is operating from an unordered pool of tuples, then a safe rule for name reduction is to apply only those 4-tuples that define a name as a key. Such applications should bring 4-tuples that started out in class (2) into class (1), and eventually reduce all names to keys. Any naming loops are avoided by this process.

6.4.1 4-tuple Threshold Subject Reduction

Some of a threshold subject's subordinate subjects might be names. Those names must be reduced by application of 4-tuples. The name reduction process proceeds independently on each name in the subordinate subject as indicated in 6.3.3 above.

One can reduce individual named subjects in a threshold subject and leave the subject in threshold form, if one desires. There is no delegation- or authorization-intersection involved, only a validity-intersection during name reduction. This might be used by a service that produces Certificate Result Certificates [see 6.7].

6.4.2 4-tuple Validity Intersection

Whenever a 4-tuple is used to reduce the subject (or part of the subject) of another tuple, its validity interval is intersected with that of the tuple holding the subject being reduced and the intersection is used in the resulting tuple. Since a 4-tuple contains no delegation or authorization fields, the delegation permission and authorization of the tuple being acted upon does not change.

6.5 Certificate Translation

Any certificate currently defined, as well as ACL entries and possibly other instruments, can be translated to 5-tuples (or name tuples) and therefore take part in an authorization computation. The specific rules for those are given below.

6.5.1 X.509v1

The original X.509 certificate is a <name,key> certificate. It translates directly to a name tuple. The form

```
[Kroot, (name <leaf-name>), K1, validity]
```

is used if the rules for that particular X.509 hierarchy is that all leaf names are unique, under that root. If uniqueness of names applies only to individual CAs in the X.509 hierarchy, then one must generate

[Kroot, (name CA1 CA2 ... CAk <leaf-name>), K1, validity]

after verifying the certificate chain by the rules appropriate to that particular chain.

6.5.2 PGP

A PGP certificate is a <name,key> certificate. It is verified by web-of-trust rules (as specified in the PGP documentation). Once verified, it yields name tuples of the form

[Ki, name, K1, validity]

where Ki is a key that signed that PGP (UserID,key) pair. There would be one tuple produced for each signature on the key, K1.

6.5.3 X.509v3

An X.509v3 certificate may be used to declare a name. It might also declare explicit authorizations, by way of extensions. It might also declare an implicit authorization of the form (tag (*)). The actual set of tuples it yields depends on the documentation associated with that line of certificates. That documentation could conceptually be considered associated with the root key of the certificate chain. In addition, some X.509v3 certificates (such as those used for SET), have defined extra validity tests for certificate chains depending on custom extensions. As a result, it is likely that X.509v3 chains will have to be validated independently, by chain validation code specific to each root key. After that validation, that root-specific code can then generate the appropriate multiple tuples from the one certificate.

6.5.4 X9.57

An X9.57 attribute certificate should yield one or more 5-tuples, with names as Subject. The code translating the attribute certificate will have to build a fully-qualified name to represent the Distinguished Name in the Subject. For any attribute certificates that refer to an ID certificate explicitly, the Subject of the 5-tuple can be the key in that ID certificate, bypassing the construction of name 4-tuples.

6.5.5 SDSI 1.0

A SDSI 1.0 certificate maps directly to one 4-tuple.

6.5.6 SPKI

An SPKI certificate maps directly to one 4- or 5- tuple, depending respectively on whether it is a name certificate or carries an authorization.

6.5.7 SSL

An SSL certificate carries a number of authorizations, some implicitly. The authorization:

```
(tag (ssl))
```

is implicit. In addition, the server certificate carries a DNS name parameter to be matched against the DNS name of the web page to which the connection is being made. That might be encoded as:

```
(tag (dns <domain-name>))
```

Meanwhile, there is the "global cert" permission -- the permission for a US-supplied browser to connect using full strength symmetric cryptography even though the server is outside the USA. This might be encoded as:

```
(tag (us-crypto))
```

There are other key usage attributes that would also be encoded as tag fields, but a full discussion of those fields is left to the examples document.

An ACL entry for an SSL root key would have the tag:

```
(tag (* set (ssl) (dns (*))))
```

which by the rules of intersection is equivalent to:

```
(tag (* set (ssl) (dns)))
```

unless that root key also had the permission from the US Commerce Department to grant us-crypto permission, in which case the root key would have:

```
(tag (* set (ssl) (dns) (us-crypto)))
```

A CA certificate, used for SSL, would then need only to communicate down its certificate chain those permissions allocated in the ACL. Its tag might then translate to:

```
(tag (*))
```

A leaf server certificate for the Datafellows server might, for example, have a tag field of the form:

```
(tag (* set (ssl) (dns www.datafellows.com)))
```

showing that it was empowered to do SSL and to operate from the given domain name, but not to use US crypto with a US browser.

The use of (* set) for the two attributes in this example could have been abbreviated as:

```
(tag (ssl www.datafellows.com))
```

while CA certificates might carry:

```
(tag (ssl (*))) or just (tag (*))
```

but separating them, via (* set), allows for a future enhancement of SSL in which the (ssl) permission is derived from one set of root keys (those of current CAs) while the (dns) permission is derived from another set of root keys (those empowered to speak in DNSSEC) while the (us-crypto) permission might be granted only to a root key belonging to the US Department of Commerce. The three separate tests in the verifying code (e.g., the browser) would then involve separate 5-tuple reductions from separate root key ACL entries.

The fact that these three kinds of permission are treated as if ANDed is derived from the logic of the code that interprets the permissions and is not expressed in the certificate. That decision is embodied in the authorization code executed by the verifying application.

6.6 Certificate Result Certificates

Typically, one will reduce a chain of certificates to answer an authorization question in one of two forms:

1. Is this Subject, S, allowed to do A, under this ACL and with this set of certificates?

2. What is Subject S allowed to do, under this ACL and with this set of certificates?

The answer to the second computation can be put into a new certificate issued by the entity doing the computation. That one certificate corresponds to the semantics of the underlying certificates and online test results. We call it a Certificate Result Certificate.

7. Key Management

Cryptographic keys have limited lifetimes. Keys can be stolen. Keys might also be discovered through cryptanalysis. If the theft is noticed, then the key can be replaced as one would replace a credit card. More likely, the theft will not be noticed. To cover this case, keys are replaced routinely.

The replacement of a key needs to be announced to those who would use the new key. It also needs to be accomplished smoothly, with a minimum of hassle.

Rather than define a mechanism for declaring a key to be bad or replaced, SPKI defines a mechanism for giving certificates limited lifetimes so that they can be replaced. That is, under SPKI one does not declare a key to be bad but rather stops empowering it and instead empowers some other key. This limitation of a certificate's lifetime might be by limited lifetime at time of issuance or might be via the lifetime acquired through an on-line test (CRL, revalidation or one-time). Therefore, all key lifetime control becomes certificate lifetime control.

7.1 Through Inescapable Names

If keyholders had inescapable names [see section 2.5, above], then one could refer to them by those names and define a certificate to map from an inescapable name to the person's current key. That certificate could be issued by any CA, since all CAs would use the inescapable name for the keyholder. The attribute certificates and ACLs that refer to the keyholder would all refer to this one inescapable name.

However, there are no inescapable names for keyholders. [See section 2.5, above.]

7.2 Through a Naming Authority

One could conceivably have a governmental body or other entity that would issue names voluntarily to a keyholder, strictly for the purpose of key management. One would then receive all authorizations through that name. There would have to be only one such authority,

however. Otherwise, names would have to be composed of parts: an authority name and the individual's name. The authority name would, in turn, have to be granted by some single global authority.

That authority then becomes able to create keys of its own and certificates to empower them as any individual, and through those false certificates acquire access rights of any individual in the world. Such power is not likely to be tolerated. Therefore, such a central authority is not likely to come to pass.

7.3 Through <name,key> Certificates

Instead of inescapable names or single-root naming authorities, we have names assigned by some entity that issues a <name,key> certificate. As noted in sections 2.8 and 2.9, above, such names have no meaning by themselves. They must be fully qualified to have meaning.

Therefore, in the construct:

```
(name (hash sha1 |TLCgPLFlGTzgUbcaYLW8kGTEnUk=|) jim)
```

the name is not

```
"jim"
```

but rather

```
"(name (hash sha1 |TLCgPLFlGTzgUbcaYLW8kGTEnUk=|) jim)"
```

This name includes a public key (through its hash, in the example above). That key has a lifetime like any other key, so this name has not achieved the kind of permanence (free from key lifetimes) that an inescapable name has. However, it appears to be our only alternative.

This name could easily be issued by the named keyholder, for the purpose of key management only. In that case, there is no concern about access control being subverted by some third-party naming authority.

7.4 Increasing Key Lifetimes

By the logic above, any name will hang off some public key. The job is then to increase the lifetime of that public key. Once a key lifetime exceeds the expected lifetime of any authorization granted through it, then a succession of new, long-lifetime keys can cover a keyholder forever.

For a key to have a long lifetime, it needs to be strong against cryptanalytic attack and against theft. It should be used only on a trusted machine, running trusted software. It should not be used on an on-line machine. It should be used very rarely, so that the attacker has few opportunities to find the key in the clear where it can be stolen.

Different entities will approach this set of requirements in different ways. A private individual, making his own naming root key for this purpose, has the advantage of being too small to invite a well funded attack as compared to the attacks a commercial CA might face.

7.5 One Root Per Individual

In the limit, one can have one highly protected naming root key for each individual. One might have more than one such key per individual, in order to frustrate attempts to build dossiers, but let us assume only one key for the immediate discussion.

If there is only one name descending from such a key, then one can dispense with the name. Authorizations can be assigned to the key itself, in raw SPKI style, rather than to some name defined under that key. There is no loss of lifetime -- only a change in the subject of the certificate the authorizing key uses to delegate authority.

However, there is one significant difference, under the SPKI structure. If one delegates some authorization to

```
(name (hash sha1 |TLCgPLFlGTzgUbcaYLW8kGTEnUk=|) carl)
```

and a different authorization to

```
(hash sha1 |TLCgPLFlGTzgUbcaYLW8kGTEnUk=|)
```

directly, both without granting the permission to delegate, that key can delegate at will through <name,key> certificates in the former case and not delegate at all in the latter case.

In the case of key management, we desire the ability to delegate from a long lived, rarely used key to a shorter lived, often used key -- so in this case, the former mechanism (through a SDSI name) gives more freedom.

7.6 Key Revocation Service

In either of the models above, key |TLCgPLFlGTzgUbcaYLW8kGTEnUk=| will issue a certificate. In the first model, it will be a <name,key> certificate. In the second, it will be an authorization certificate delegating all rights through to the more temporary key.

Either of those certificates might want an on-line validity test. Whether this test is in the form of a CRL, a re-validation or a one-time test, it will be supplied by some entity that is on-line.

As the world moves to having all machines on-line all the time, this might be the user's machine. However, until then -- and maybe even after then -- the user might want to hire some service to perform this function. That service could run a 24x7 manned desk, to receive phone calls reporting loss of a key. That authority would not have the power to generate a new key for the user, only to revoke a current one.

If, in the worst case, a user loses his master key, then the same process that occurs today with lost wallets would apply. All issuers of authorizations through that master key would need to issue new authorizations through the new master key and, if the old master key had been stolen, cancel all old authorizations through that key.

7.7 Threshold ACL Subjects

One can take extraordinary measures to protect root keys and thus increase the lifetimes of those keys. The study of computer fault-tolerance teaches us that truly long lifetimes can be achieved only by redundancy and replacement. Both can be achieved by the use of threshold subjects [section 6.3.3], especially in ACL entries.

If we use a threshold subject in place of a single key subject, in an ACL (or a certificate), then we achieve redundancy immediately. This can be redundancy not only of keys but also of algorithms. That is, the keys in a threshold subject do not need to have the same algorithm.

Truly long lifetimes come from replacement, not just redundancy. As soon as a component fails (or a key is assumed compromised), it must be replaced.

An ACL needs to be access-controlled itself. Assume that the ACL includes an entry with authorization

```
(tag (acl-edit))
```

Assume also that what might have been a single root authorization key, K1, is actually a threshold subject

(k-of-n #03# #07# K1 K2 K3 K4 K5 K6 K7)

used in any ACL entry granting a normal authorization.

That same ACL could have the subject of an (acl-edit) entry be

(k-of-n #05# #07# K1 K2 K3 K4 K5 K6 K7)

This use of threshold subject would allow the set of root keys to elect new members to that set and retire old members. In this manner, replacement is achieved alongside redundancy and the proper choice of K and N should allow threshold subject key lifetimes approaching infinity.

8. Security Considerations

There are three classes of information that can be bound together by public key certificates: key, name and authorization. There are therefore three general kinds of certificate, depending on what pair of items the certificate ties together. If one considers the direction of mapping between items, there are six classes: name->key, key->name, authorization->name, name->authorization, authorization->key, key->authorization.

The SPKI working group concluded that the most important use for certificates was access control. Given the various kinds of mapping possible, there are at least two ways to implement access control. One can use a straight authorization certificate:

(authorization->key)

or one can use an attribute certificate and an ID certificate:

(authorization->name) + (name->key)

There are at least two ways in which the former is more secure than the latter.

1. Each certificate has an issuer. If that issuer is subverted, then the attacker can gain access. In the former case, there is only one issuer to trust. In the latter case, there are two.

2. In the second case, linkage between the certificates is by name. If the name space of the issuer of the ID certificate is different from the name space of the issuer of the attribute

certificate, then one of the two issuers must use a foreign name space. The process of choosing the appropriate name from a foreign name space is more complex than string matching and might even involve a human guess. It is subject to mistakes. Such a mistake can be made by accident or be guided by an attacker.

This is not to say that one must never use the second construct. If the two certificates come from the same issuer, and therefore with the same name space, then both of the security differentiators above are canceled.

References

[Ab97] Abadi, Martin, "On SDSI's Linked Local Name Spaces", Proceedings of the 10th IEEE Computer Security Foundations Workshop (June 1997).

[BFL] Matt Blaze, Joan Feigenbaum and Jack Lacy, "Distributed Trust Management", Proceedings 1996 IEEE Symposium on Security and Privacy.

[CHAUM] D. Chaum, "Blind Signatures for Untraceable Payments", Advances in Cryptology -- CRYPTO '82, 1983.

[DH] Whitfield Diffie and Martin Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, November 1976, pp. 644-654.

[DvH] J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations", Communications of the ACM 9(3), March 1966.

[ECR] Silvio Micali, "Efficient Certificate Revocation", manuscript, MIT LCS.

[ELIEN] Jean-Emile Elien, "Certificate Discovery Using SPKI/SDSI 2.0 Certificates", Masters Thesis, MIT LCS, May 1998, <http://theory.lcs.mit.edu/~cis/theses/elien-masters.ps> [also .pdf and

[HARDY] Hardy, Norman, "THE KeyKOS Architecture", Operating Systems Review, v.19 n.4, October 1985. pp 8-25.

[IDENT] Carl Ellison, "Establishing Identity Without Certification Authorities", USENIX Security Symposium, July 1996.

[IWG] McConnell and Appel, "Enabling Privacy, Commerce, Security and Public Safety in the Global Information Infrastructure", report of the Interagency Working Group on Cryptography Policy, May 12, 1996; (quote from paragraph 5 of the Introduction).

[KEYKOS] Bomberger, Alan, et al., "The KeyKOS(r) Nanokernel Architecture", Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, USENIX Association, April 1992. pp 95-112 (In addition, there are KeyKOS papers on the net available through <http://www.cis.upenn.edu/~KeyKOS/#bibliography>).

[KOHNFELDER] Kohnfelder, Loren M., "Towards a Practical Public-key Cryptosystem", MIT S.B. Thesis, May. 1978.

[LAMPSON] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice", ACM Trans. Computer Systems 10, 4 (Nov. 1992), pp 265-310.

[LANDAU] Landau, Charles, "Security in a Secure Capability-Based System", Operating Systems Review, Oct 1989 pp 2-4.

[LEVY] Henry M. Levy, "Capability-Based Computer Systems", Digital Press, 12 Crosby Dr., Bedford MA 01730, 1984.

[LINDEN] T. A. Linden, "Operating System Structures to Support Security and Reliable Software", Computing Surveys 8(4), December 1976.

[PKCS1] PKCS #1: RSA Encryption Standard, RSA Data Security, Inc., 3 June 1991, Version 1.4.

[PKLOGIN] David Kemp, "The Public Key Login Protocol", Work in Progress.

[R98] R. Rivest, "Can We Eliminate Revocation Lists?", to appear in the Proceedings of Financial Cryptography 1998, <http://theory.lcs.mit.edu/~rivest/revocation.ps>.

[RFC1114] Kent, S. and J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", RFC 1114, August 1989.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, December 1996.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, December 1996.

[RFC2047] K. Moore, "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, December 1996.

[RFC2065] Eastlake, D. and C. Kaufman, "Proposed Standard for DNS Security", RFC 2065, January 1997.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[SDSI] Ron Rivest and Butler Lampson, "SDSI - A Simple Distributed Security Infrastructure [SDSI]", <http://theory.lcs.mit.edu/~cis/sdsi.html>.

[SET] Secure Electronic Transactions -- a protocol designed by VISA, MasterCard and others, including a certificate structure covering all participants. See <http://www.visa.com/>.

[SEXP] Ron Rivest, code and description of S-expressions, <http://theory.lcs.mit.edu/~rivest/sexp.html>.

[SRC-070] Abadi, Burrows, Lampson and Plotkin, "A Calculus for Access Control in Distributed Systems", DEC SRC-070, revised August 28, 1991.

[UPKI] C. Ellison, "The nature of a useable PKI", Computer Networks 31 (1999) pp. 823-830.

[WEBSTER] "Webster's Ninth New Collegiate Dictionary", Merriam-Webster, Inc., 1991.

Acknowledgments

Several independent contributions, published elsewhere on the net or in print, worked in synergy with our effort. Especially important to our work were: [SDSI], [BFL] and [RFC2065]. The inspiration we received from the notion of CAPABILITY in its various forms (SDS-940, Kerberos, DEC DSSA, [SRC-070], KeyKOS [HARDY]) can not be over-rated.

Significant contributions to this effort by the members of the SPKI mailing list and especially the following persons (listed in alphabetic order) are gratefully acknowledged: Steve Bellovin, Mark Feldman, John Gilmore, Phill Hallam-Baker, Bob Jueneman, David Kemp, Angelos D. Keromytis, Paul Lambert, Jon Lasser, Jeff Parrett, Bill Sommerfeld, Simon Spero.

Authors' Addresses

Carl M. Ellison
Intel Corporation
2111 NE 25th Ave M/S JF3-212
Hillsboro OR 97124-5961 USA

Phone: +1-503-264-2900
Fax: +1-503-264-6225
EMail: carl.m.ellison@intel.com
cme@alum.mit.edu
Web: http://www.pobox.com/~cme

Bill Frantz
Electric Communities
10101 De Anza Blvd.
Cupertino CA 95014

Phone: +1 408-342-9576
EMail: frantz@netcom.com

Butler Lampson
Microsoft
180 Lake View Ave
Cambridge MA 02138

Phone: +1 617-547-9580 (voice + FAX)
EMail: blampson@microsoft.com

Ron Rivest
Room 324, MIT Laboratory for Computer Science
545 Technology Square
Cambridge MA 02139

Phone: +1-617-253-5880
Fax: +1-617-258-9738
EMail: rivest@theory.lcs.mit.edu
Web: http://theory.lcs.mit.edu/~rivest

Brian Thomas
Southwestern Bell
One Bell Center, Room 34G3
St. Louis MO 63101 USA

Phone: +1 314-235-3141
Fax: +1 314-235-0162
EMail: bt0008@sbc.com

Tatu Ylonen
SSH Communications Security Ltd.
Tekniikantie 12
FIN-02150 ESPOO
Finland

EMail: ylo@ssh.fi

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.