Storing and Retrieving Internet Certificates

Pekka Nikander
pekka.nikander@hut.fi
Helsinki University of Technology

Lea Viljanen
lea.viljanen@cs.helsinki.fi
University of Helsinki

Abstract

Effective storing, retrieval and interpretation of certificate chains is a difficult problem. The original X.500 and X.509 proposals, with their rigid global naming scheme and complex access protocols have proved to be less than optimal, leading to various short-cuts. For example, the de facto X.509 retrieval protocol appears to be TCP/IP based LDAP instead of the original OSI based Directory Access Protocol, DAP.

In this paper we present a completely new architecture for administration, storing and retrieval of digital certificates. Instead of X.509 certificates we base our architecture on SPKI, a more flexible certificate format proposed by the IETF. The new architecture allows complex certificate chains to be effectively and easily administered, using the Internet Domain Name Service, or DNS, as the certificate storage, replication and retrieval mechanism. The interpretation of the certificates is based on our Internet Security Policy Daemon architecture.

1 Introduction

A digital certificate is a signed statement that represents knowledge or belief expressed by its issuer. Traditionally, certificates have been used to express the issuer's knowledge (or belief) that the holder of the certified key has a certain name, e.g., a Distinguished Name, in some predefined domain. This naming information may also, implicitly, denote some kind of authorization or trust expressed by the issuer.

More recently, a number of independently developed alternatives to the identity certificate schemes manifested in X.509 [10] and PGP [11] have been presented. The PolicyMaker prototype [3] by Blaze, Feigenbaum and Lucy introduced the idea of certifying some kind of policy authority, or authorization, instead of a name. In a way, the PolicyMaker certificates represent capabilities. Simultaneously, the SDSI proposal by Rivest et al [9] brought forth the idea of linked, private namespaces instead of a single, global namespace. Originally developed independently by Ellison and others, the SPKI proposal [6] by an Internet Engineering Task Force (IETF) SPKI working group, took ideas from both of these developments, and along with some original ideas it is being developed into a comprehensive, flexible certificate system.

The Internet Domain Name System (DNS) [8] is a distributed, fault tolerant directory system originally developed for storing and retrieving information about Internet hosts. The main usage of the DNS was, and is, the conversion of Internet host names into addresses, and vice versa. However, from its very beginning, it has been possible to store all kinds of other information within the DNS infrastructure as well. Recently there are proposals that allow digital signatures and certificates to be stored in the DNS.

In this paper we show how the DNS certificate records [4] can be used to effectively store and retrieve SPKI certificates. Furthermore, we show how the certificates should be organized within the distributed DNS tree so that both their administration (i.e. addition and removal) and retrieval is practical and effective. Extending ideas presented by Aura [1][2], we also describe a retrieval algorithm with a number of heuristical improvements.

The rest of this paper is organized as follows. In Section 2 we describe the SPKI proposal in sufficient detail to base our future discussion on it. In

Section 3 we describe the Internet Domain Name System (DNS), the proposed certificate resource record format, and a method to store information about individual users (instead of host computers) within a DNS domain. Next, in Section 4, we show how the DNS may be used as a repository to effectively store and retrieve SPKI certificates. Section 5 outlines an example, where SPKI certificates are used to control access to a company extranet. Finally, in Section 6, we draw some conclusions.

2 SPKI

The Simple Public Key Format and Infrastructure (SPKI), is an Internet proposal (Internet draft), work in progress produced by an IETF working group. The ideas behind the SPKI proposal were partly originally developed by Carl Ellison, its primal promoter, partly drawn influence from the SPKI and PolicyMaker papers [6][3], and partly developed through the discussions and arguments by the working group.

So far, the SPKI has been developed into three separate draft documents, one describing the basic ideas behind the proposal, the second describing the certificate format and the third containing examples. In Section 2.1, we briefly describe the format. In Section 2.2, we discuss four different types of certificates that are typically needed to resolve security policy decisions. These types are used to express identity, permissions, delegation, and trust. In Section 2.3, we show how these four different types can be used to create so called certificate loops that are needed in resolving trust problems.

2.1 Certificate Format and Semantics

Conceptually, a SPKI certificate consists of five fields that have security relevance, and a signature. The five fields are used to denote the Issuer, the Subject, the Delegability, the Authority, and the Validity of the certificate. The Issuer and Subject are usually expressed as *public keys*, not as names as in e.g. X.509. This allows the Issuer and Subject to be relatively anonymous, if desired. The Delegability is a binary field that denotes whether the Subject may further delegate the Authority or not. The Authority field identifies some Authority granted by the Issuer to the Subject. The interpretation of this field is solely defined by the Issuer; we return back to this point later. Finally, the Validity field contains information about when the certificate is valid, how to retrieve a corresponding Certificate Revocation List (CRL), or how to otherwise check the certificate's validity on-line.

More formally, a SPKI certificate may be expressed as a five tuple (I,S,D,A,V), where I is the public key of the Issuer, or an one way hash of the public key, and S is the public key of the Subject, a hash of the public key, or a local name of the Subject in the Issuer's local name space. D is the Delegation bit, and either true, denoting that the Authority may be further delegated, or false, effectively forbidding delegation. A is the Authorization. V is the Validity; according to the SPKI proposal, it is an URI that provides information how to check the certificates validity. The on-line validation can be performed using DNS queries, too.

From a semantic point of view, the Authority is the most important and most versatile field in the certificate. Basically, the meaning of the Authority field is always primarily defined by the Issuer, i.e., the signer, of the certificate. That is, since the Issuer has signed the certificate, it must be assumed that it knows, exactly, what the certificate is supposed to express. However, for all practical purposes, we must assume some kind of standard format for the Authority so that delegated certificates may be effectively handled and reduced. The current SPKI proposal has resolved this problem by defining an abstract, solely syntactic tag algebra. This algebra, along with its so called star forms allows sets of named authorities to formed. The algebra allows the sets to be manipulated by forming unions, intersections and ordered sets.

Given two SPKI certificates, (I1, S1, true, A1, V1) and (I2, S2, D, A2, V2), where S1 = I2, i.e., where the subject of the first is the issuer of the second, one may create a new SPKI certificate (I1, S2, D, A, V). This certificate is the result of the resolving of the delegation present in the original certificate pair. In the resulting certificate, the issuer is that of the first certificate, the subject that of the second one. Delegation is directly inherited from the second certificate.

The authority and validation fields are more interesting. By definition, the result authority is the intersection of the authorities, i.e., A = intersection(A1, A2). Similarly, the validity is an intersection, V = intersection(V1, V2). In the original proposal, the authority intersection is defined by the tag algebra.

Local names and the SPKI group concept. As mentioned earlier, the Subject of an SPKI certificate may be a local name instead of a public key or a key hash. This feature allows late binding of keys by the certificate's issuer. It also allows the formation of SPKI groups.

Let us consider a simple example. Alice, the Issuer, wants to give Carol, a colleague of hers introduced to her by her very trusted fried Bob, the Authority A. If she knew Carols public key, $K_{Carol,}$ she could create a certificate (K_{Alice}, K_{Carol}, false, A, some validity). However, if she only knew Carol through Bob, and doesn't yet know Carol's public key, she can resort to trust Bob. A certificate (K_{Alice}, K_{Bob}'s Carol, false, A, some validity) effectively expresses the same authority by identifying Carol through a local name in Bob's name space. That is, Alice refer's to Carol by saying that Carol is someone in the name space maintained by Bob, i.e., the entity that possesses the ability to create certificates signed by K_{Bob}.

In the same way, Alice can create a group of people by creating a certificate of the format (K_{Alice}, K_{Alice}'s Group, D, A, V), and a number of identity certificates of the format (K_{Alice}, $K_{GroupMember}$, false, belongs_to_Group, V').

As a further refinement, SPKI allows the subject to be a treshold. Instead of being a key, a key hash, or a name, the Subject field may denote a group of N keys, K of which are needed *simultaneously* in order to allow the authority of the certificate to be executed. For example, if any two of the three top executives of a company are needed to approve purchases exceeding $100,000, this may be expressed with an SPKI certificate as ($K_{Company}$, 2-of-3 $K_{Manager1}$ $K_{Manager2}$ $K_{Manager3}$, D, may_approve_purhcase_exceeding_$100000, V).

2.2 Certificate Types

Based on our initial analysis, the practical usage of SPKI certificates in networked access control decisions seems to require at least four different kinds of certificates. These certificate functions express Identity, Permissions, Delegation, and Trust. The differences between these categories are more semantical than formal. We describe each of these categories in turn.

Identity. Basically, an Identity certificate denotes that the Subject has a certain name in the Issuer's name space. However, they can also be used to express more complex identities. For example, the Issuer may express its belief that a certain service, identified by a name in some foreign name space, is provided by the Subject. Naturally, such a certificate may not be blindly trusted, but its trustworthiness must first be resolved.

In this paper, we denote a simple identity certificate as (K_{Issuer}, $K_{Subject}$, false, K_{Issuer}'s Name, V). Similarly, a name claim can be expressed as (K_{Issuer}, $K_{Subject}$, false, K_{other}'s Name, V).

Permissions. A certificate may be used to express that the Subject has a certain right. If we consider an access control function performed by a network entity, this right may represent permission to access a facility. On the other hand, depending on application, such a right may express almost anything, e.g., permission to drive a vehicle[*].

A statement, expressed by the Issuer, that the Subject has a certain access permission, may be expressed as (K_{Issuer}, $K_{Subject}$, false, Perm, V). Here, the Authority Perm is understood by the eventual verifier to give permission to access the controlled facility.

Delegation. A delegation is a certificate that authorizes the Subject to issue certificates on the behalf of the Issuer. We distinguish it from the next type of certificates, or trust certificates, since the authority to issue certificates is usually somehow restricted. For example, the security administrator of a company A may issue a certificate that allows the security administrator of another company, B, to issue certificates that authorizes access to one of the computers owned by A, but only to employees of the company B, and only for a limited time.

Trust. The final form of certificates that we consider expresses trust. In this context we want to denote full or absolute trust by the Issuer on the Subject. That is, the Issuer trusts the Subject to be capable of creating any certificates on its behalf.

[*] Such an certificate would be the digital counterpart of a physical driver's license.

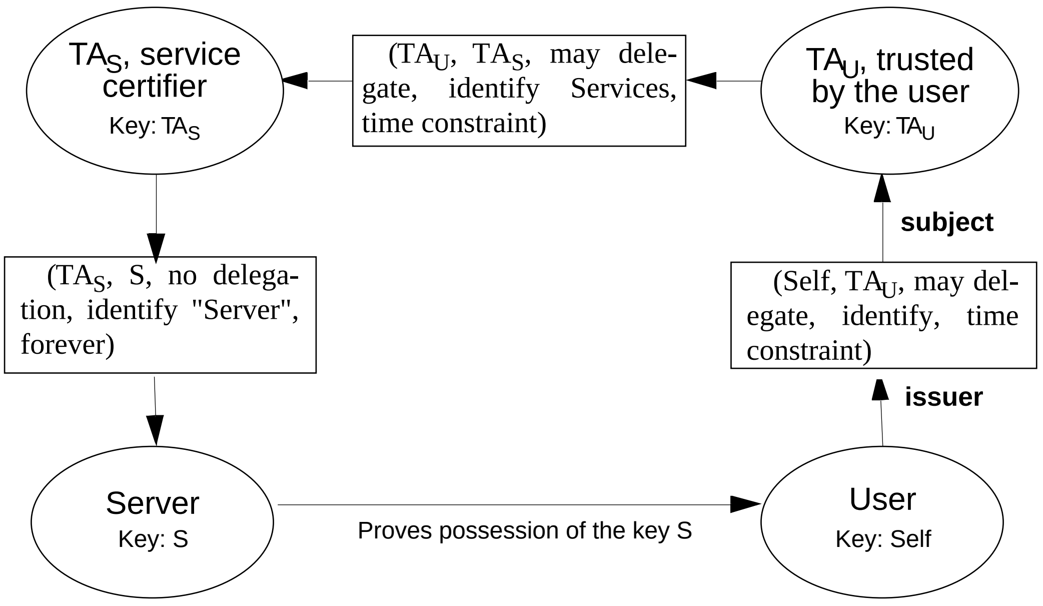
TA_S, service certifier
Key: TA_S
(TA_U, TA_S, may delegate, identify Services, time constraint)
TA_U, trusted by the user
Key: TA_U
subject
(TA_S, S, no delegation, identify "Server", forever)
(Self, TA_U, may delegate, identify, time constraint)
issuer
Server
Key: S
Proves possession of the key S
User
Key: Self

Figure 1: Basic service identification loop

Such a trust certificate may be issued for a limited time only, however. In other words, the real distinction between a trust and a delegation certificate is in the authority field. While a delegation certificate allows the Subject to issue certificates for a specific authority, a trust certificate allows the Subject to issue certificates for any purpose.

2.3 Certificate Loops

According to the idea of the SPKI proposal, certificates are chained together into sequences. Typically, the last certificate within a sequence is an identity or permission certificate, giving some identity or application specific authority to the final Subject. The final certificate is preceded by zero or more delegation certificates, passing the naming or permission authorization. The first certificate within the sequence must be issued by the verifier of the sequence. It is typically a trust or a delegation certificate.

When a certificate sequence is used in order to prove identity or permission to access, the final Subject of the sequence proves the possession of its private key using a conventional public key authentication protocol. In a way, the execution of the authentication protocol can be viewed as an on-line creation of a virtual certificate. The virtual certificate states, in a way, that the final Subject wants to use the authorization granted to it by the certificate sequence.

From a topological point of view, the execution of the authentication protocol closes a certificate sequence into a loop. That is, the first certificate in the sequence is issued by the verifier, who is also the subject of the virtual certificate created by the authentication protocol.

A certificate loop to verify service identity. Let us first consider the use of SPKI certificates for a more traditional certificate function, namely verifying the identity of a network service. This is a well known application domain, and even X.509 certificates have been successfully used to implement this function. The main benefit of the SPKI system is to make all trust relationships explicit [7].

In this example, a user U wants to gain assurance that a networked server S indeed provides the service that the user wants to access. Instead of a usual Certificate Authority (CA) hierarchy, we envision a mesh, or loosely coupled network of trust or identity authorities, or TAs. Basically, the user explicitly decides which of the TAs to trust, and how much. For example, the user may trust one authority to identify banking services and another authority to identify on-line stores that should be trusted to accept electronic money. The user may also control how much transitive trust to place on each of the trusted TAs. On the other hand, the identity of the service must be certified by one of the TAs that the user either directly or transitively trusts.

This situation is displayed in Figure 1. All of the parties, the user U, the Server S, the trust authority TA_U that the user directly trusts, and the trust authority TA_S certifying the service's identity, each have their corresponding key pair. The user has expressed its trust on TA_U by creating an appropriate trust certificate. TA_U has then delegated some of the trusted identity authority to TA_S, either directly or through some path that is acceptable to the user. Finally, TA_S certifies that the server S really provides the desired service.

A certificate loop to verify the user's access rights. Now, let us consider a slightly more complicated case. In this case the server S wants to verify that the user U really has right to access the service. Traditionally this has been accomplished by using an identity scheme similar to the one above, and a separate access control list (ACL) stored into the server. However, when using SPKI certificates the ACL is unnecessary. In fact, with SPKI certificates we can verify the client's access rights even when the client desires to stay anonymous.

In this example, the server S is administered by a policy administrator PA_S. Typically, the PA may be the security officer of the organization owning the server S. This relationship is represented digitally as a trust certificate signed with K_S, denoting that the server S (unconditionally) trusts on the policy administrator PA_S. This policy administrator, on its behalf, delegates a right to grant access to the server to the policy administrator of the user's organization, PA_U. PA_U in turn grants the user U a right to access the server S. This situation is displayed in Figure 2.

3 The Domain Name System

The Domain Name System (DNS) [8] is a global distributed database. It was originally created to map Internet host names to IP addresses and vice versa, distributing the namespace and control to individual organizations. It has proven to be very efficient and versatile, and has become a critical part of the Internet infrastructure.

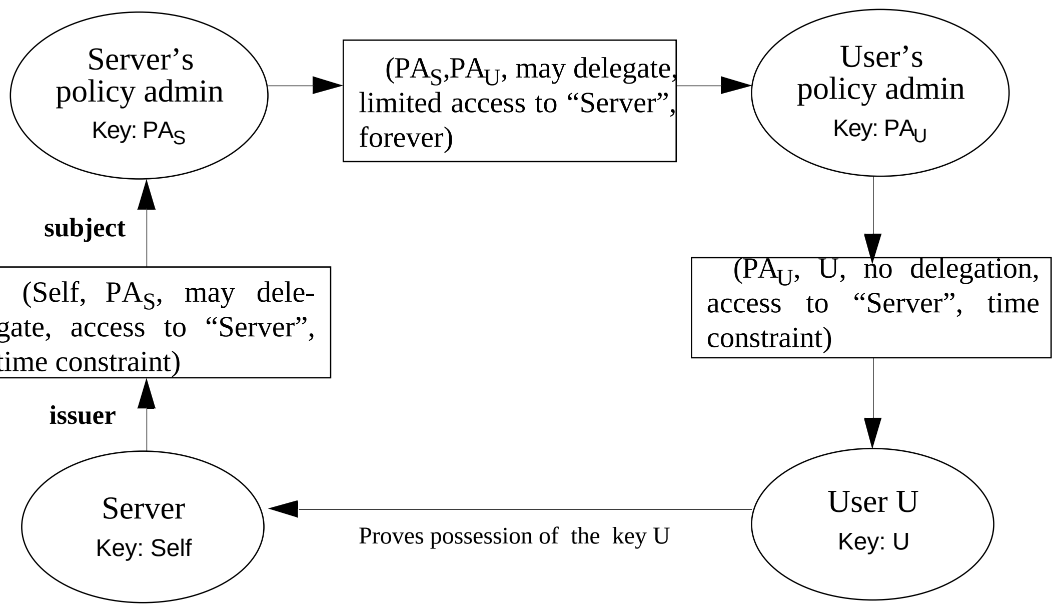
Server's policy admin
Key: PA_S
(PA_S,PA_U, may delegate, limited access to "Server", forever)
User's policy admin
Key: PA_U
subject
(Self, PA_S, may delegate, access to "Server", time constraint)
(PA_U, U, no delegation, access to "Server", time constraint)
issuer
Server
Key: Self
Proves possession of the key U
User U
Key: U

Figure 2: Basic authorization certificate loop

In Section 3.1 we have a brief look into the basic DNS naming structure, the following section explores ways to name services and users. Section 3.3 introduces work in progress to define a certificate resource record type.

3.1 Overview

The DNS naming space is a classical tree structure consisting of arcs and nodes. Nodes have a label, which can be considered as a text string for our purposes. The null label is reserved for the root node. Sibling nodes cannot have the same label. The domain name of a node is the list of the labels on the path from the node to the root of the tree. In the textual notation a dot "." is used as the separator between node labels (or can be thought as an arc label).

Nodes contain data, which is arranged as typed resource records (RR). Resource record types define what kind of data can be stored in the DNS, for example, IP-addresses, free text etc. Creating new resource record types requires IETF standardization action.

By the realization of DNS being a critical component of the Internet and it lacking any form of data origin authentication, DNS security extensions were created by the IETF DNSSEC working group. The DNS security (DNSSEC) standard [5] specifies three new security related resource record types, of which the public key (KEY) record type is relevant to this paper.

The KEY is a general purpose public key resource record type. In our schema it is used to attach public keys to keyholders, i.e entities who need keys in the Internet (hosts, services and users).

3.2 Naming Non-Host Entities

Services. The DNS system is focused on naming physical hosts and storing their attributes (IP-address, mail exchange information etc.). Hosts in turn implement services. To be able to store public key information for a service, we need to name it in the DNS. Fortunately, this is standard practice today; most well managed domains use service aliases like `www.acme.com` or `mail.acme.com` to point to the real host or hosts implementing the service. This indirection enables managers to change the actual host without reconfiguring a score of client programs.

Users. Representing users is a more difficult problem. There is no user naming per se in the domain naming scheme. Users do not translate easily into hosts. However, there is a way to specify a user by way of his or hers mailbox address by replacing the @-separator with a dot, for example `user@acme.org` would be `user.acme.com` in the DNS mailbox name syntax [8]. This type of user naming is used by the DNSSEC standard if mapping users to DNS is needed.

However, a major consideration with users is the privacy issue. The storage system should not reveal any more information on the user than is required by the authentication or authorization process. Since SPKI certificate based authorization does not need to reveal the user name to the service, neither should the DNS. Thus the mailbox type of naming is not a good solution for SPKI certificates. For X.509 certificates it may suffice.

To store public keys and SPKI certificates we need the user to DNS name mapping to be one way only. Therefore we could use an one way hash function (for example MD5) to create a hash string from the user name and/or other account information to be used as a DNS name component instead of the mailbox. For example the MD5 algorithm hashes "Some User" to "12e472e68a416-9fb904d41ac30dbd1f4". The corresponding domain name would be `12e472e-68a4169fb904d41ac30dd1f4.acme.com`.

The hash algorithm should be selected keeping in mind that the maximum length of a single DNS label is 63 bytes, i.e 252 bits using hexadecimal encoding. If more bits are needed, a more effective encoding can be chosen, e.g., BASE64. The DNS standard itself allows almost all byte values to be used in the labels.

To prevent this technique being subject to input guessing attack, we can use keyed hash. The probability of a duplicate hash is algorithm dependent but generally sufficiently small.

3.3 The Certificate Resource Record Type

To store certificates in the DNS structure we need a new resource record type defined. This work is currently in progress [4]. The aim is to create a single

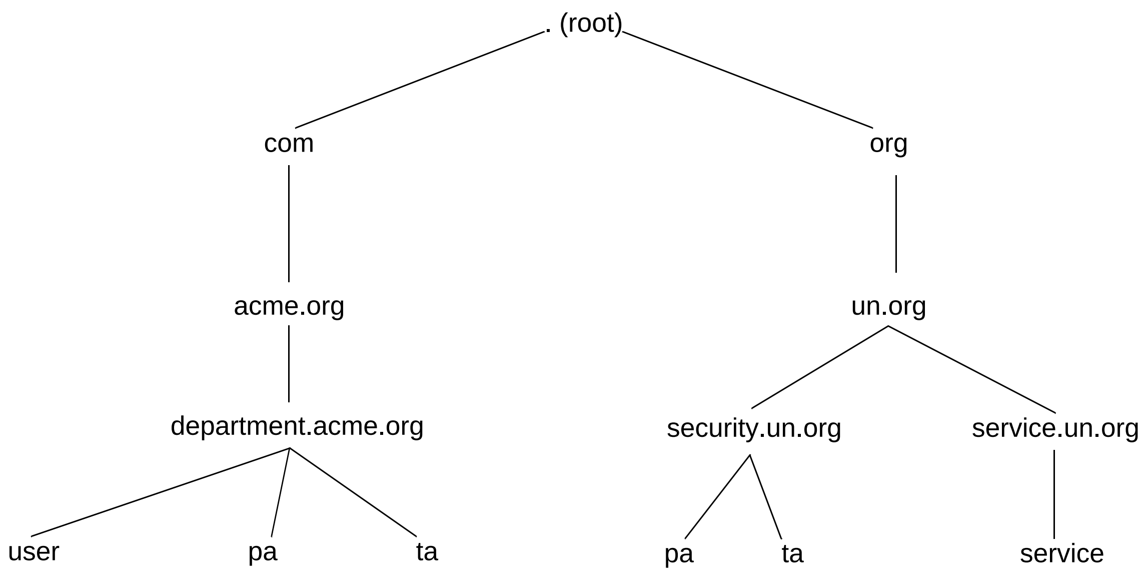

Figure 3: Example DNS tree

certificate record type which is able to contain any kind of certificate (e.g X.509, SPKI, PGP or some other yet undefined).

The CERT record format currently consists of four elements: type, key tag, algorithm number and the certificate or certificate revocation list itself. The type element specifies the certificate type. The key tag is a 16 bit hash of the subject's key. The tag is used to quickly determine which KEY and CERT records belong together. The algorithm numbers are assigned by IANA, currently only one number (1 for MD5/RSA) has been assigned. The certificate or CRL itself is stored in a BASE64 encoded string or strings. Thus the internal structure of the certificate is not visible in the DNS.

4 DNS as the SPKI Certificate Storage

Given the SPKI certificate semantics and the DNS certificate resource record, we now propose an effective and simple way to store and retrieve SPKI certificates using the DNS. Basically, we store copies of the relevant SPKI certificates at suitable locations within the DNS hierarchy so that on-line creation and verification of SPKI certificate sequences and loops becomes relatively straightforward. Currently we are implementing a prototype of the certificate and trust management system based on this proposal.

In Section 4.1, we describe the general idea of storing SPKI certificates in DNS resource records. In addition to that, we show how to organize the certificates in a meaningful way. In the next part, Section 4.2, we show how this organization can be used to effectively build certificate sequences. The sequences, in turn, may be used to check authorization as outlined in Section 2.3. Finally, we discuss the issues pertaining to adding and removing certificate resource records.

4.1 Storing SPKI Certificates into the DNS Nodes

To simplify the representation we will consider an example of a partial DNS hierarchy that consists of two organizations. One organization has a server to be accessed, and the organization's trust and policy administrators. The other organization has a user that wishes to access the server, and corresponding trust and policy administrators. This example is shown in Figure 3. The structure may be easily generalized into a more complex situation involving several organizations and multiple delegations.

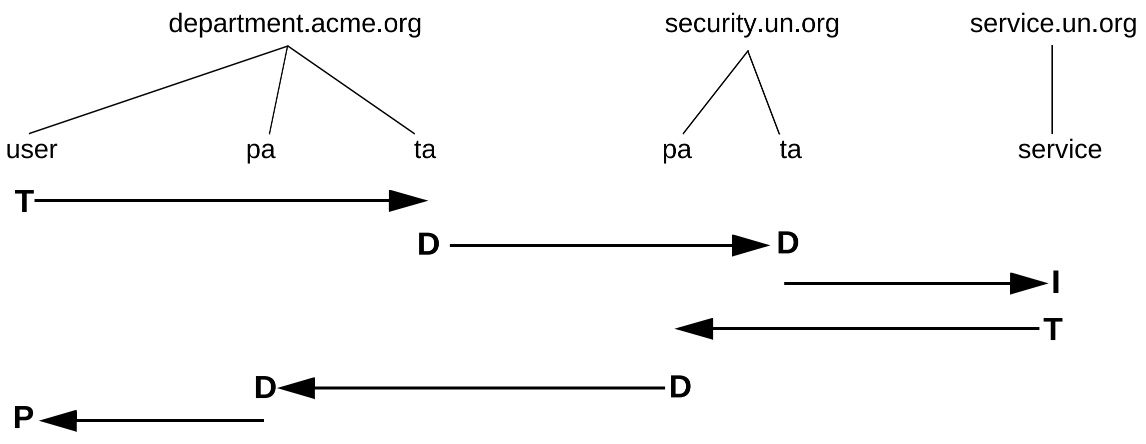

Figure 4: Storage and location of certificates

The basic idea of our schema is to have DNS nodes that carry resource records pertaining to a specific SPKI principal, i.e. a SPKI key. The binding between a DNS domain node and a SPKI key need not be secure; DNS is just a convenient place to search for certificates that have that particular key as their issuer or subject. Technically, the domain name of this node is given as the optional issuer-location and/or subject-location fields of each certificate.

Where to store a given certificate, at its issuer DNS node, subject DNS node, or both, depends on the certificate type. Trust certificates are stored only at the issuer node. Delegation certificates are stored both at the issuer and at the subject. Permission and identity certificates are stored only at the subject node.

Let us first consider the trust certificates (T). Trust certificates are basically only needed by their issuer, when verifying the validity of a certificate sequence. Each verifier naturally knows its own DNS name. Storing the trust certificates under this DNS name allows the verifier to fetch trust certificates on demand, allowing them to be used even in networked devices with very little memory. Furthermore, if the issuer of the next certificate in the chain is known, it is easy to filter out the only trust certificate needed for this particular verification[*].

What comes to permission certificates (P), it is natural to associate them with their subject. First, each active subject knows its domain name. This allows it to fetch all its own permission certificates from the DNS. Second, the certificates can be considered as properties of the subject; hence, it is natural to store them under the subject's name.

With respect to storage location, identity certificates (I) are similar to permission certificates. They should be stored in the DNS node corresponding to their subject. Behind this is the assumption that the DNS name of a service is *known beforehand*, i.e., before any security checks are made. This allows the verifying party to fetch any identity certificates that apply to the target directly from the DNS directory.

Delegation certificates (D) seem to be most problematic. In one respect, delegation expresses trust on the delegator's behalf. Conversely, they can be considered as properties of the delegate, expressing trust placed on them. Furthermore, as we will shortly show, the search algorithm sometimes needs to traverse delegation paths in either direction. Therefore we propose that the delegation certificates are stored both on the delegating party and at the delegate.

The proposed organization of various certificates and their storage points is depicted in Figure 4.

[*] Due to the possibility of having Certificate Reduction Certificates (CRC certificates), this is quite an important possibility from performance point of view.

4.2 Search algorithm

Our search algorithm extends the one presented by Aura [1]. In his paper Aura describes and analyses three algorithms: forward search, backward search, and two-way search. Our algorithm is based on the two-way search variant, adding a number of heuristics to cut back average search cost.

Basically, all the certificates (stored in the DNS tree) form a directed delegation network. The nodes of the network are the issuers and subjects of the certificates (i.e., the keys). The nodes are physically represented as DNS nodes. The arcs of the network represented by the certificates themselves, each certificate representing an arc from the issuer to the subject. The search problem to solve is to find a path from the verifier to the final subject, thereby creating a certificate chain. Furthermore, the path must be such that the permission or identity being checked is transferred on each arc belonging to the path. If there are any delegation restrictions or other details breaking the transitivity of the trust, they must be considered in the arc selection phase of the algorithm.

Definition of a delegation network. Formally, a delegation network $DN = (K, C)$ is a set of keys K, forming the nodes of the network, and a set of arcs (certificates) C, $C \quad K \times K \times D \times A \times V \times N \times N$, where $D = \{ false, true \}$ is the delegation bit, $A \quad 2^P$ is the set of authorizations, represented as sets of permissions $p \quad P$, and V is the set of verifications (ignored), and N is the set of DNS names, denoting the issuer and subject locations.

Search problem. Given the formal definition above, the search problem can be formulated. Given a verifier $k_v \quad K$ with a corresponding DNS name $n_v \quad N$, a final subject $k_s \quad K$ with a corresponding DNS name $n_s \quad N$, and a permission $p \quad P$, the problem is to find a sequence of certificates $C_S = \quad c_0, \quad , c_n$ such that the issuer of c_0 is k_v, the subject of c_n is k_s, $c \quad C_S, p \quad \text{authorization}(c)$, $c \quad \{ c_0, \quad , c_{n-1} \}, \text{delegation}(c) = true$, and all the certificates are also otherwise relevant.

Relevant certificates. Before describing the algorithm itself, we define the concept of a *relevant certificate*. A certificate c is *relevant* with respect to a search problem (k_v, n_v, k_s, n_s, p) iff

1. The authorization field of the certificate denotes p, i.e., $p \quad \text{authorization}(c)$
2. Either the delegation bit of the certificate is true, or the certificate is the last one in the chain, i.e., $\text{delegation}(c) = true \quad \text{subject}(c) = k_s$
3. The certificate is not otherwise unsuitable to be included in the chain. For example, some earlier certificate may have limited the maximum length of the chain, or there may be restrictions expressed in the authorization field that mark the certificate bad with respect to the chain being formed. In general, these and other such details are beyond the scope of this paper, and assumed to be taken care of in the actual implementation.

Algorithm. Now we are ready to present the actual algorithm. The algorithm consists of two main steps. First, a forward search is performed. The forward search is terminated as soon as the forming search tree starts to branch. Second, a backward search is done. The algorithm is terminated when a satisfying certificate sequence is found, or the search is deemed failed by the heuristics.

Forward search

1. Set the current DNS name n_v,the current key k_v, and an empty chain.
2. Fetch all certificates using the current DNS name.
3. Filter out all certificates that are not issued by the current key.
4. Filter out all certificates that are not relevant to the current search problem.
5. Filter out all certificates whose subject is already present in the chain.
6. For all certificates whose subject is the final subject k_s, check the signature. If the check succeeds, add the certificate to the end of the chain, terminate,and indicate success. Otherwise, filter out the certificate (and issue a warning).
7. If there are no more certificates left, terminate and indicate failure.
8. If there is only one certificate left, check that it is correctly signed by the issuer, add it to the chain, set its subject location and subject as the current DNS name and key, and continue from step 2.
9. The search tree branches. Set up backward search target as all the certificates in the chain, and the remaining certificates in the current fetch set. Mark those taken from the sequence as checked and rest as unchecked.

Backward search

1. Start with the target set formed by the forward search. Set the current DNS name as n_s and the corresponding key as k_s, and an empty backward tree.
2. Fetch all certificates using the current DNS name.
3. Filter out all certificates whose subject is not the current key.
4. Filter out all certificates that are not relevant to the current search problem.
5. Filter out all certificates whose issuer is already present in the backward tree.
6. For all certificates whose issuer is present in the backward search target, check the signature. If the check succeeds, check the signature of the found target if marked unchecked. If both checks succeed, terminate and indicate success. If either of the checks fails, filter out the certificate (and issue a warning).
7. If there are no more certificates left, terminate and indicate failure.
8. Add the remaining certificates as leaves to the backward tree.
9. Using the heuristics (see below), either terminate indicating probable failure, or select one of the leaves of the backward tree, and continue from step 2.

Heuristics. Basically, we have added two different sets of heuristics: termination heuristics and backward tree selection heuristics. However, these are somewhat mixed, as we shall shortly see.

According to Aura's analysis, a typical successful search terminates in relatively few steps, while an unsuccessful search may require several magnitudes more steps. Thus, from a practical point of view it is useful to terminate the search relatively fast in the face of a probable unsuccessful termination instead of performing an exhaustive search. Our heuristics fullfill this requirement.

The second background issue behind our new heuristics lies in the structure of the DNS tree. In typical cases the certificate chains will flow either directly from the verifiers DNS domain to the subjects DNS domain, or through at most one intermediate domain trusted by the verifier. Saying this, we consider the DNS domains to consists of the actual subdomain plus any superdomains up to one or two levels below the top level domain. That is, the domain of the node `some.department.acme.org` is considered to cover the node itself, `department.acme.org` and `acme.org`.

Given these preliminaries, we can now describe the heuristics.

Leaf selection and termination heuristics.

1. Consider the verifier's domain (with its superdomains as discussed above), the subject's domain (with superdomains) and any other domains present in the backward target set as relevant domains.
2. When selecting a leaf certificate to follow, consider the certificate's issuer location. The close the issuer location is to the verifier's domain, other relevant domain, or the subject's domain, the better the leaf is. For example, if the verifier's domain is `service.un.org` and leaf's issuer location is `pa.security.un.org`, the leaf is quite good because it belongs to an immediate subdomain of the verifier's superdomain `un.org`.
3. In addition to the leaf's issuer location's closeness to the relevant domains, the leaf's depth from the root of the backward tree (i.e. the final subject) is also important. We believe that in most practical settings the certificate paths will be short, i.e. probably 3–6 certificates long, and certainly shorter than 10 certificates long. (If this is not the case, the situation can be administratively fixed by creating suitable CRC certificates.) Therefore, we suggest that an upper limit is set to the check depth.
4. The termination can be based on both leaf relevance and depth. When all remaining leaves exceed some combined irrelevance and depth level, the search should terminate.

4.3 Administering certificates

From an administrative point of view, certificates are created, stored into DNS for retrieval, and removed from DNS once they have become obsolete (revocation is not covered by this paper). Depending on the certificate, certificate creation can happen in several possible ways, including both off-line and on-line creation. However, once they have been created, they have to be stored in the DNS tree at appropriate locations.

Adding or removing certificates is not very different from other DNS administration. Adding and removing hosts and services is a routine operation. Thus, the only relevant problem seems to be to make sure that any changes needed are *appropriate* from the DNS administrator's point of view.

Trust and identity on the server side. Servers tend to be relative stable. Services are probably changed more often, but still not too frequently. Similarly, the service identity and the administrative keys a server is programmed to trust are probably rather stable. The service identities and initial trust relations are typically administered within an organization. Thus, the host and service administration patterns seem to correspond well with the security administration requirements.

Trust and permissions on the user side. Eventually, the users should be allowed to decide by themselves whom to trust. Therefore, it would be beneficial if the user's could administer their own trust certificates. However, these certificates are only needed when the verifier is the user itself. Therefore, in many cases these certificates need not actually be stored in the DNS, but they can be permanently cached in the user's workstation or smart card. Thus, their final storage seems to depend heavily on the actual application and terminal equipment used.

Permissions are clearly a different issue. Typically, a user is granted permissions by a security administrator. This administrator may belong to the user's organization or not. In the case of an administrator within the same organization, he or she is probably closely connected with the user's DNS administrator (they may even be one and single person), and there does not seem to be any conflicts of interest. However, if some security administrator from some other organization wants to grant permissions just to a certain person, these permissions are not necessarily relevant at all from the user's DNS organizations point of view. However, in such a case the permission certificates can be permanently cached just like the trust certificates.

Thus, if the user, for a reason or another, does want to store the trust and cross-organizational permissions within the DNS database, the user probably should have his or her own zone. In this case the user may have two DNS names, one within the organizational domain, and one personal name. In this case, the personal name is the one that belongs to the user specific zone. If, on the other hand, permanent certificate caching is enough, no such a zone is needed.

Delegations. Delegations seem to be the most problematic. The nature of the two-way search algorithm requires that they are stored on both at the issuer and at the subject locations. Typically, these locations belong to different organizations. However, the issuer of the certificate has usually a close connection with the issuer location's DNS domain; therefore, storing the delegation at the issuer end should pose no administrative problems. Luckily, the subject of the delegation typically really needs the delegation; at least when delegating further, if not earlier. Thus, it seems plausible to assume that given good enough tools, the administrator in the subject end of a delegation certificate is motivated enough to fetch and store the certificate also at the subject location.

Removing expired certificates. Removal of expired certificates should probably be done by some automatic means. It is quite easy to write a program that traverses the DNS tree, looks for expired certificates, and removes them. Removal is not relevant from a security point of view, and need not be necessarily performed on the case of revocation. However, integrating removal with revocation is probably a good idea since it may improve overall performance.

5 Example

As an example, let's consider the Acme Inc. extranet WWW pages, to which access is granted to employees of "friendly" organizations according to the company security policy. Now, in our first phase, Some User from the United Nations, being naturally a friendly party, wants to gain access to the extranet pages. At the latter stage in Section 5.2 we see the user accessing the pages.

5.1 Granting Access

The system or WWW administrator of the extranet service has two choices, either to directly grant access to Some User, or move the decisions to a higher level in the organization, for example the security policy administrator.

If the extranet administrator grants certificates himself and the certificates are stored with the service or server data, the situation is similar to the normal ACL usage, only differing in the storage system (DNS) and the ACL entry format (certificate). Therefore we do not peruse this direction further.

Delegating control. If the extranet service administrator has trusted the local policy authority for access permissions and created a trust certificate ($K_{extranet.acme.com}$, $K_{pa.acme.com}$, true, everything, V), the Some User's request for access may be handled by the policy authority.

If the user access request is according to the company policy, the policy authority either creates the access certificate directly to the user ($K_{pa.acme.com}$, $K_{userhash.un.org}$, false, read_http://extranet.acme.org/, V) , or creates a delegation certificate for the user's organization's policy authority ($K_{pa.acme.com}$, $K_{pa.un.org}$, true, read_http://extranet.acme.org/, V). This policy authority in turn can create an access certificate for the extranet ($K_{pa.un.org}$, $K_{userhash.un.org}$, false, read_http://extranet.acme.org/, V) , which it is able to store in the DNS with the other Some User's certificates.

5.2 Accessing the Service

Now, Some User wants to access the Acme Inc extranet for information on a thingamanjig they manufacture. When the User clicks the Acme Inc extranet link in the WWW browser, he or she must prove the possession of the private key to the remote service using a conventional public key authentication protocol.

When the service has been convinced that behind the HTTP connection is someone knowing the private key for $K_{userhash.un.org}$ it can start executing the algorithm presented in Section 4.2.

In the forward search phase the service searches all certificates created by its access granting key $K_{extranet.acme.com}$. In our example, the service has created a trust certificate for $K_{pa.acme.com}$ with that key.

The search jumps to the DNS node `pa.acme.com` and continues. The forward phase usually terminates there, since policy authorities generally have created several delegation certificates. Target set contains the nodes `extranet.acme.com`, `pa.acme.com` and all nodes for which $K_{pa.acme.com}$ has created a delegation certificate for the current operation, including in our example `pa.un.org`.

The backward search phase starts with the user DNS node. If the certificate records for `userhash.un.org` include a certificate issued by any of the target set keys $K_{extranet.acme.com}$, $K_{pa.acme.com}$ or $K_{pa.un.org}$ and the authority and signature matches, the certificate loop is thus closed and WWW pages opened.

This certificate loop can be resolved by the client, too. In that case the client executes the algorithm and passes all certificates in the chain for the service to verify.

6 Conclusions

The recent proliferation of non-hierarchical certificate systems such as PGP and SPKI create new needs for certificate disrtibution and retrieval. The X.500 directory structure, on which the X.509 certificate storage and retrieval is based on, seems less than ideal for other certificate formats and semantics. The Internet Domain Name System (DNS) provides a hierarchical, distributed, fault-tolerant and flexible name space, where certificates with differing semantics can be easily stored.

In this paper we have defined a way to store SPKI certificates within the DNS name space. We have shown that using this organization the certificates can be effectively retrieved and managed.

We have also given an algorithm that allows certificate sequences and loops to be looked up on demand. To reduce the average certificate sequence lookup time we have added a number of improvement heuristics. Finally, we have analyzed the adminstrative implications of our suggested scheme showing that it addresses the basic administrative requirements particularly well.

All in all, we have shown that it is feasible to create a technically sound infrastructure for policy based certificates in the Internet.

References

[1] Aura, T. “Comparison of Graph-Search Algorithms for Authorization Verification in Delegation Networks”, In Proceedings of 2nd Nordic Workshop on Secure Computer Systems, 1997.

[2] Aura, T. “On the Structure of Delegation Networks“, Licenciate’s thesis, Helsinki University of Technology, 1997.

[3] Blaze, M., Feigenbaum, J., Lacy, J., “Decentralized Trust Management”, In Proceedings of the IEEE Symposium on Security and Privacy, 1996

[4] Eastlake 3rd, D., Gudmundsson, O. “Storing Certificates in the Domain Name System”, Internet Draft, draft-ietf-dnssec-certs-01.txt, 1997.

[5] Eastlake 3rd, D., Kaufman, C., “Domain Name System Security Extensions”, Request For Comments 2065, 1997.

[6] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T., “Simple Public Key Certificate”, Internet Draft, draft-ietf-spki-cert-structure-04.txt, 1997.

[7] Lehti, I., Nikander, P. , “Certifying Trust”, to appear in Proceedings of the Practice and Theory in Public Key Cryptography, 1998.

[8] Mockapetris, P. V., “Domain names -- concepts and facilities”, Request For Comments 1034, 1987.

[9] Rivest, R., Lampson, B., “SDSI - A Simple Distributed Security Infrastructure”, Technical Report, 1996.

[10] International Telegraph and Telephone Consultative Committee (CCITT), “Recommendation X.509, The Directory - Authentication Framework”, CCITT Blue Book, Vol VIII.8, pp. 48-81, 1988.

[11] Zimmermann, P., “The Official PGP Users Guide”, MIT Press, 1995.