Virtual Hierarchies - An Architecture for Building and Maintaining Efficient and Resilient Trust Chains

John Marchesini and Sean Smith *
{carlo,sws}@cs.dartmouth.edu

Department of Computer Science - Dartmouth College, Hanover, NH USA
www.cs.dartmouth.edu/~pkilab

Abstract. In *Public Key Infrastructure (PKI)*, the simple, monopolistic organizational model of a single certificate issuing entity works fine until we consider real-world issues. Then, issues such as scalability and mutually suspicious organizations create the need for a multiplicity of certificate issuing entities, which introduces the problem of how to organize them to balance resilience to compromise against efficiency of path discovery. Many solutions involve organizing the infrastructure to follow a natural organizational hierarchy, but in many cases, such a natural organizational hierarchy may not exist. In this paper, we use tools such as *secure coprocessing*, *threshold cryptography*, and *peer-to-peer networking* to address the former problem by overlaying a *virtual hierarchy* on a mesh architecture of peer certificate issuing entities, and achieving both resilience and efficiency.

1 Introduction

1.1 The Problem

Background By separating the privilege to decrypt or sign a message from the privilege to encrypt or verify, *public-key cryptography* enables forms of trusted communication between parties who do not share secrets *a priori*. Eliminating the need for shared secrets has multiple advantages. On a global level, it potentially enables extending trusted communication across organizational boundaries, between parties who have never met. But it can also reduce overhead in managing communication between parties even on a local level, within one organization: the number of needed keys goes from $\Omega(n^2)$ to $O(n)$, where n is the number of entities.

PKI has many definitions; the most commonly accepted definition refers to how one participating party learns what the public key is for another party. Typically, approaches to PKI begin by condensing trust: rather than *a priori* knowing the public key of each party in the population, the relying party instead knows the public key of a designated special party, who in turn issues signed statements (e.g., certificates and CRLs) about members of the population.

* This work was supported in part by Internet2/AT&T, by IBM Research, by the Mellon Foundation, and by the U.S. Department of Justice, contract 2000-DT-CX-K001. However, the views and conclusions do not necessarily represent those of the sponsors. A preliminary version appears as TR2002-416.

This designated party is typically called the *certificate authority (CA)*. Some approaches separate the process of issuing certificates from the process of identifying and authorizing keyholders to receive the certificates; in these approaches, the latter tasks become the responsibility of the *registration authority (RA)*. Since the CA must hold and wield a private key of considerable value, implementations apply various protections to that private key, such as housing it in a hardened cryptographic module that is kept offline. Some ambiguity thus results regarding what the term "CA" refers to: the entity (typically online) that issues and manages certificates; or this entity's specific machine that houses the private key. In this paper, we use the first implication.

There are numerous models which describe how a CA and RA should be related, and who should operate them. One notion is to have an independent third party operate the CA, and let the organization with actual end-users only operate the RA. Another popular model is to have the CA and RA run by the same organization and have this enterprise PKI blessed by some higher level third party organization (e.g. Verisign).

We make the latter assumption that the CA and RA are managed by the same domain (i.e. as an *enterprise PKI*). This arrangement is discussed in RFC 2459 and in the current literature (e.g. [1–5]). Real-world deployers of this methodology include: the U.S. Treasury and State Departments, the Federal Deposit Insurance Corporation, Verisign, and many colleges and universities, including Dartmouth College.

More than One This simple PKI model of one CA servicing a user population suffers from some inherent limitations. A certificate is the CA's assertion that a keyholder possesses certain properties, such as a particular identity. In order to accept a certificate, a relying party needs to trust the CA to do two tasks: (1) to ensure that its currently valid certificates only assert bindings the CA judges to be true; and (2) to judge a binding is true only when the keyholder really has those properties. If the relying party does not know the CA—or if the CA is not in a position to verify the identity of the keyholders according to some uniform policy—then the relying party cannot reasonably accept the certificate.

These tasks can lead to conflicting constraints. Task (2) requires a relationship between the CA and the keyholders—which can lead to the enterprise PKI model [1] of multiple CA/keyholder sets. However, task (1) requires a relationship between the CA and the relying party, which creates the need to organize these PKIs so that public key operations can take place across these sets.

PKIs within an organization are becoming a common occurrence. Such systems have been well studied, and are often built from commercially available components. Within an organization which has a PKI, the certificates generated by the organization's CA are meaningful. Outside of such an organization, however, those same certificates are meaningless unless some agreement between a number of organizations is in place.

The question thus arises of how to organize multiple CAs. The basic literature (e.g. [2]) gives a serious examination of this question. Readers unfamiliar with this literature may be tempted to assert that the only natural solution here is to use a *name-constraint*

[1] Some older approaches to PKI attempt to address this conflict by isolating the CA—task (1)—but delegating registration to local RAs—task (2). However, this creates its own trust complications [5].

hierarchy: group CAs into sets that have some natural relationship; for each group, establish a new CA that certifies the CAs in that group; and continue this process upward, so that the we result in a tree with a single trust root. For example, one might follow the DNS hierarchy, and assume that a global root certifies a `edu` CA, which certifies a `dartmouth.edu` CA, which certifies a `cs.dartmouth.edu` CA, which certifies each member of our department.

Although apparently natural, this hierarchical approach has many drawbacks.

- Hierarchies are not always *scalable*, in that they cannot permit the participating fraction of the population to grow gradually. Suppose the natural social hierarchy has four levels, and two unrelated leaves want to establish a trust relationship. They can only do this if all the interior CAs—from the first leaf, up to the root, then down to the second leaf—are already participating in the PKI.
- Hierarchies are not always *usable*. The globally unique names determined by the natural hierarchy may not necessarily be usable by the humans who need to use them to make trust judgments. A colleague reports that his `foo.com` domain has 100K machines whose names are of the form `bar.foo.com`. Not only is this namespace crowded—a typo will likely give the user the wrong machine instead of an error—but it also changes dynamically: a sysadmin the user never meets changes machine names without notification. Thus, using the natural hierarchy as a basis for trust (via a hierarchical PKI) is problematic.
- Hierarchies do not always *exist.* The relationship between users and their immediate CAs is usually (but not always) natural. However, the upper regions get murky. With mobile devices [6, 7] or collections of universities or government departments, one typically encounters federations of peers with no clear natural organization.[2] Indeed, except for DNS and perhaps the Roman Catholic church, it is hard to find a natural hierarchy whose upper regions are well-defined. Which U.S. military service[3] should be the root? Which DOE laboratory?[4] Why should CREN or USPS or NIH be the root over academia or U.S. citizens at large?

PKI as a system Consequently, we are going to take the unorthodox view of looking at "PKI" as a *system* that has various properties, instead of an automatic mirror of a social arrangement that may not necessarily be appropriate, even if it exists.

We might start by thinking of conventional CAs (from the simple model above) as nodes, and trying to decide how to link them together—perhaps by creating new CAs—in a directed graph, where edges go from a CA to each entity that it certifies.

Two desirable properties of any PKI are *resilience* and *efficiency*.

- By *resilience*, we mean the ability of the system to tolerate the discovery that any given key pair has been compromised. What trust judgments become impossible? How many key pairs must be revoked and reissued?

[2] The fact that "everyone gets to be King" has been cited as motivation for some U.S. government bridge projects.

[3] The first author would assert that it's the branch to which he belonged.

[4] The second author would assert that it's the DOE laboratory for which he used to work.

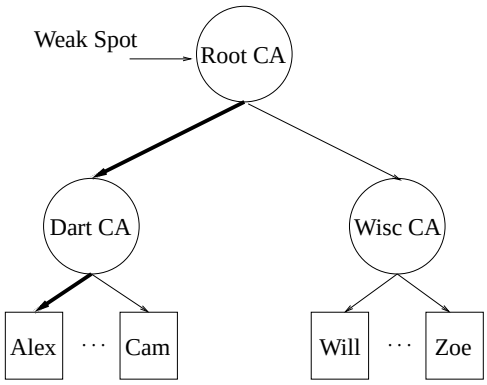

Fig. 1. In a hierarchy, when Zoe receives a certificate from Alex, she verifies that the Dartmouth CA certificate is signed by the Root CA. She then verifies that Alex's certificate is signed by the Dartmouth CA. As the number of CAs grows, it takes $O(logV)$ time to verify the path from the Root CA to the user which needs to be verified. If the Root CA is compromised in a hierarchy, the system goes down until all certificates are revoked and new certificates are issued.

– To verify a certificate, a relying party needs to find a path from a *trust root* (a node it *a priori* trusts) to the certificate in question. By *efficiency*, we mean the running time of the algorithm to discover this path.

Resilience and efficiency are typically competing goals.

Structured Centralization. Many current architectures impose a rigid structure on the organization of CAs, which means that path construction and validation can be deterministic and efficient. Although this structure permits path algorithms to traverse the topology within some efficient time constraints, it also results in a large amount of authority residing in a single place (e.g. the root CA). This centralization of authority directly decreases resilience: if the root CA is compromised, the entire PKI is unusable until it can recover.

Hierarchies are the canonical example of this structured approach. Traditionally, hierarchies achieve $O(logV)$ (where V is the number of CAs) verification time, because paths in a tree are well-defined and easy to find (Figure 1). We noted many drawbacks above; another drawback is that hierarchies place increasing amounts of value on the private keys of interior nodes. If an adversary were to compromise an upper-level CA or even the root CA, the entire PKI must suspend operation until a recovery can occur (i.e. all certificates issued by that CA are revoked, and new ones are reissued with the CA's new private key) (Figure 1).

Thus, hierarchies obtain efficiency at the cost of resilience.

Unstructured Decentralization. An opposing view involves organizing CAs in a more decentralized way in an effort to increase resilience by not placing so much authority in one centralized place. However, decentralization implies that path validation algorithms must now do more work and must often use non-determinism to decide if a

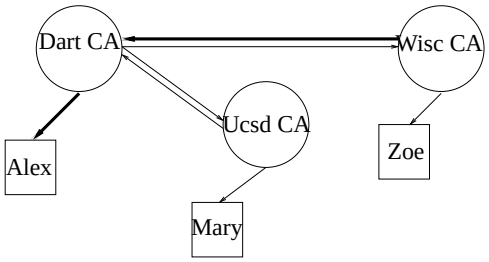

Fig. 2. Meshes offer increased resilience. If the Wisconsin CA's private key is disclosed, the other CAs can continue to operate. Only Wisconsin is affected. When Wisconsin gets back online, it may rejoin the CA network. Meshes also offer decreased efficiency. When Zoe receives a certificate from Alex, she verifies that that a trust path exists from Alex's CA to a point which she trusts by examining certificates stored in the CA directories. In this example, the Dartmouth CA and Wisconsin CA are cross-certified, so she believes Alex's certificate is good. As the number of CAs grows, it takes $O(V)$ time to verify a certificate chain.

received trust chain is valid. These properties translate into a decrease in efficiency and an increase in complexity on the part of the verifier.

Meshes are the canonical example of the unstructured approaches. Mesh PKI architectures have been developed in part to avoid this single point of failure (Figure 2). However, the non-deterministic nature of peer-to-peer organization increases the complexity of the path verification algorithm significantly (Figure 2). Due to the fact that not all possible choices lead to a trusted CA, coupled with trial-and-error construction of the trust path (a path to a trusted CA), verification time in these schemes is usually high. Further, mesh architectures make no guarantee to avoid cycles, leading to the existence of choices in the path construction algorithm which may never terminate.

Thus, meshes obtain resilience at the cost of efficiency.

Other Approaches Finding algorithms which increase the efficiency of path construction in decentralized organizations is an emerging area of research. Algorithms which use *certificate extensions* (such as name constraints and policy extensions), as well as loop elimination techniques have been developed to enhance efficiency [1]. Yet another possibility could be to use machine addressing and routing techniques, as the problems appear to be potentially isomorphic. Our concern however, is the underlying organization of CAs, and how they may be arranged to achieve efficiency and resilience.

Other common architecture schemes are more hybrid.

Extended Trust Lists are used to give users the ability to maintain lists of CAs which they choose to trust. Each entry in this list may represent a single CA or an entire PKI, which itself could be a Hierarchy or a Mesh. This scheme poses new challenges for validation algorithms, as the starting point for these algorithms could be any node in the list. The implication is that a path may have be to be constructed using every entry in the list as a starting point.

Bridge CAs provide another alternative to the common approach of cross-certifying enterprise PKIs through peer-to-peer relationships. Cross-certification without a Bridge

CA results in $(n^2 - n)/2$ relationships for n enterprises (in graph theory, this graph is known as a complete graph on n vertices, and is named K_n [8]).

The Bridge allows each of the enterprise PKIs to cross-certify to it, resulting in a star topology between enterprises and reducing the number of relationships to n. Bridges are also used for translating between different policy and legal domains. In these situations, the PKIs themselves are usually complex.

While this is an attractive solution if all of the enterprises are Hierarchies, the Bridge architecture does not solve path validation issues in general, as each of the enterprises themselves may be Meshes [2, 3].

The *Bottom Up With Name Constraints* model [4] is one which shares our goal of allowing organizations to construct their own PKI and then connect it to other organizations' PKIs. The model assumes a hierarchical namespace and that CAs are certified in both directions, down (from parent to child) and up (from child to parent). The model also allows for CAs to cross certify directly.

Path validation in this model is quite efficient due to the presence of certification in both directions. The validation algorithm begins by starting at a trust anchor and looking first for a cross-certified CA which is either an ancestor of the target or the target itself. If this fails, the algorithm proceeds up to the parent CA and searches through its cross-certified CAs. This terminates when a cross certificate is found or when a common ancestor is found.

This model also has impressive resilience properties. If a key is compromised, the compromised CA can issue new certificates to all of the CAs which are certified (up, down, and cross). The communities belonging to each certified CA will automatically be bound to the new key, without having to make any changes.

The major difference between this model and ours is that we relax the assumption of a hierarchical namespace. As mentioned earlier, hierarchies are not always usable, scalable, or present.

1.2 Our Solution

We believe that both properties—efficiency and resilience—are important to most PKI systems. We thus propose an architecture and are developing a prototype which aims to bridge the gap between these seemingly competing goals. We feel this is novel as most current architectures fail to provide both goals.

Our objective is to devise an architecture which allows for CAs to organize themselves in such a way as to maintain the following two invariants:

1. *Efficiency.* Trust chains produced by any of the entities may be verified in an efficient manner; trust chains are loop-free. This is common in hierarchy schemes.
2. *Resilience.* The secrets (private keys) do not exist in any one place; for upper-level CAs, private-key computations require collaboration. The parties are fairly randomly distributed throughout the topology.

This architecture is useful when the relative authority of the CAs in the real world is not easily represented by a strict hierarchy and when certificates need to be used frequently outside of the issuing namespace.

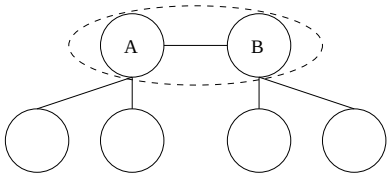

Fig. 3. A single collective. All of the nodes are CAs. The nodes inside of the oval (A and B) are maintaining a portion of the private key privilege (e.g. via threshold cryptography), which is acting as the Root CA key. The other nodes are directly connected to one of the nodes which collaborate in cryptographic computations for the collective and may "use" the key (i.e. make a broadcast to have a message signed by the nodes in the oval). It should be noted that it is possible to put all of the nodes inside the oval, meaning that each member of the collective would maintain a portion of the private key privilege.

Overview The mechanism we propose to accomplish this task is a *virtual hierarchy*—a logical hierarchy formed in a peer-to-peer network. As with a standard hierarchy, we can model a virtual hierarchy as a tree with nodes and directed edges. Leaves can represent bottom-level users; their parents represent their natural CA.

However, the remaining nodes are *virtual* CAs. Although each such node is a logical entity in the virtual hierarchy, it represents the *collective* action of a set of conventional CAs. Consequently, we use the term *collective* for this set. (See Figure 3.) In the virtual hierarchy, such collectives of conventional CAs comprise all the upper-level CAs.

We obtain this collective action via cryptography. There are a number of cryptographic schemes which require collaboration in order to perform cryptographic computations. The broad category for such methods is known as *threshold cryptography*. Some examples include *multi-party signature schemes* [9] and *Multi-Party RSA* [10], which we will discuss in Section 3.4. Even *secret sharing* [11] might qualify as a primitive threshold scheme.

To simplify exposition, we frame our prototype in terms of secret splitting, where the parties in a subset of a collective each hold a fragment of the collective's private key. To wield this key, the parties reconstitute it—leading to the significant drawback that a single party might retain the key. More advanced threshold schemes do not share this weakness; Section 3.2 discusses these issues further.

We have developed (and are prototyping) algorithms that allow natural CAs to form collectives in an ad hoc manner. They then organize collectives into a hierarchy (where a virtual node can itself become certified by another collective) in order to maintain a good tree structure. (See Figure 4.)

This approach thus obtains the goals we desired:

1. *Efficiency.* By maintaining the structure of a hierarchy, we retain an expected $O(logV)$ trust chain verification cost, with no loops.
2. *Resilience.* By distributing the higher-level CA private key privileges among multiple parties, we retain the resilience of decentralized approaches.

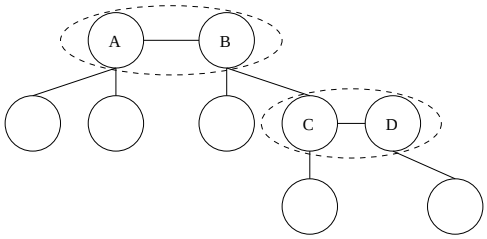

Fig. 4. A hierarchy is emerging. Here there are two collectives, linked by the CA denoted as "C". C is a member of the original collective shown in Figure 3, and is a member collaborating in cryptographic computations (along with D) of the second level collective. The ovals contain nodes which share private key privileges.

Application Layer	CAs reside here.
Virtual Hierarchy Layer (VHL)	Logical hierarchy layer.
Trusted Peer Access Layer (TPA)	Peer-to-peer layer.
TCP/IP Protocol Stack	The TPA uses TCP.

Fig. 5. The protocol stack.

Physical Layer The physical layer is a peer-to-peer network of secure coprocessors [12] (we use the IBM 4758 [5]). The secure coprocessor is not strictly necessary to make the virtual hierarchy layer work. However, since nodes in this layer are CAs, they must all have a cryptographic module; however, tamper-resistance and remote attestation/outbound authentication adds security against insider attack at any one node. Practically speaking, part of our decision to use secure coprocessors came from the fact that we already had some devices, we had some familiarity with the programming environment, and the modules we had are validated to FIPS 140-1 Level 4.

2 Overall Structure

We approached the problem in two stages, the first was to implement a peer access layer which allows secure coprocessor to communicate securely, and the second was to implement the virtual hierarchy algorithms on top of that layer. The resulting protocol stack is depicted in Figure 5.

[5] Recently, security vulnerabilities [13, 14] have been demonstrated in an application (IBM's CCA) which runs on the 4758. It should be noted that these vulnerabilities belong to the application, and not the 4758 platform. At the time of writing, the 4758 has no known vulnerabilities.

Our prototype implements the Virtual Hierarchy Layer (VHL) and the Trusted Peer Access Layer (TPA). The prototype version of the VHL contains a command line interface so that we do not need to integrate with a CA at this stage in development. The peer access layer running inside of the IBM 4758 is depicted as the TPA layer, and the algorithms which construct and maintain the logical hierarchy are shown as the VHL. The two layers are implemented as separate processes, with the output of the VHL being piped into the TPA using standard UNIX pipes.

Before we discuss the layers in detail, a simple example will be useful in understanding the high level operation and what we are trying to achieve. In the example shown in Figures 6 through 10, two machines A and B will connect, negotiate a private key, and store half of the private key privilege. This operation forms the root collective. Four more machines will join the collective, and are able to "use" the private key, whose privilege is maintained by A and B. The result is similar to Figure 3. Then a new collective will be formed by C and D, and a new node will be placed in tree (i.e. the virtual hierarchy, shown in the "VHL" column of the diagrams). This is a simple example, building a virtual hierarchy with only two nodes, but will serve to familiarize the reader with the basic concept. As more CAs make connection requests in the TPA, the tree in the VHL will continue to grow downward.

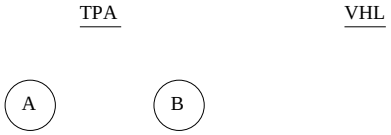

Fig. 6. Step 1: The two machines are not connected, and neither of them share the private key privilege for a virtual CA.

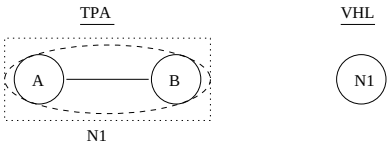

Fig. 7. Step 2: A connects to B and they establish a virtual CA, resulting in the two parties now sharing a portion of the private key privilege (as denoted by the oval). A collective is formed and a node $N1$ is established in the virtual hierarchy.

3 Virtual Hierarchy Layer

From the highest level, the virtual hierarchy (i.e. the logical hierarchy in the peer-to-peer network) is constructed by an algorithm that allows peer CAs to establish a secure

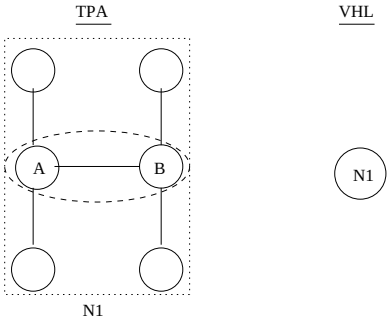

Fig. 8. Step 3: More nodes join the collective by connecting to one of the machines already in the collective (A or B). None of the new nodes are required to negotiate keys, as the maximum size of a collective for this example is six. Since the four new CAs are allowed to sign statements with the key held whose privilege is shared by A and B (as in Figure 3), the virtual hierarchy remains unchanged.

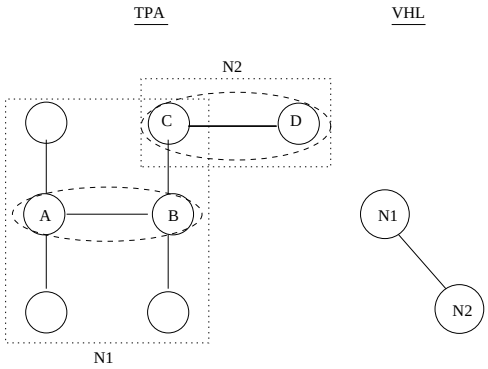

Fig. 9. Step 4: Machine D makes a connection to machine C. Because D is at a distance greater than one from one of machines acting as a virtual CA (machines A and B in this example), C and D must establish a new virtual CA and share a portion of the private key privilege (as in step 2 above). This operation forms a new collective and a new node $N2$ in the virtual hierarchy. Note that CA C is now in both collectives.

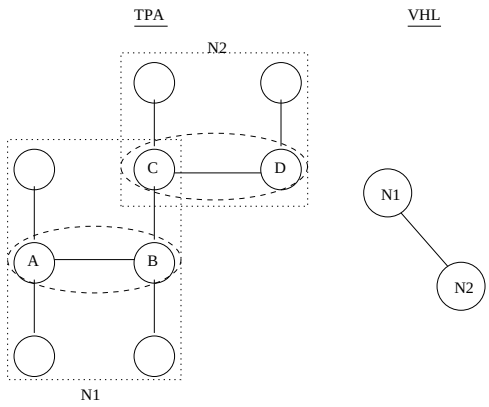

Fig. 10. Step 5: Two CAs make connections to members of the new collective (C or D), and join the new collective. Since a private key has already been established for that collective (whose privilege is shared by C and D), and the maximum size is six, the new members do not need to negotiate a key (as in Step 3). The virtual hierarchy remains unchanged.

connection and negotiate a secret which each of their communities may use as an end-point in their trust chain. Pieces of the trust root's privilege are then stored among the peers who negotiate it.

This leads us to make the following two claims:

Increased Resilience The result of this negotiation produces a root "entity" whose privilege is distributed among the n parties who are at a distance one (i.e. directly connected) to one of the actors in the negotiation. This group of n parties is a collective and all act as though the "entity" is their root CA. The result of requiring collaboration to perform cryptographic computation among a group of peers alleviates the single point of failure problem. While the number of targets increases, the payoff for successfully compromising a target decreases. Minimally, an attacker must successfully compromise at least as many targets as are sharing the private key privilege to get the private key. As with pro-active security [15], this makes the scheme more resilient to compromise.

Increased Efficiency The result of this negotiation produces a root "entity" whose role is to act as a trust point for the n parties who are at a distance one to one of the actors in the negotiation. We argue that this maintains a hierarchical trust structure similar to one which would be found in a physical hierarchy of CAs. Maintaining this hierarchy allows trust calculations to be performed at an average of $O(logV)$ time (again where V is the number of CAs participating in the network).

It is important to realize that our solution, like other current schemes, would require each community to perform some amount of work to reflect the new topology. Time and space complexity analysis of this task is an area for future work.

3.1 Registration

In order for a CA to join a collective, it must offer a statement of its policy and practices to the collective for review. If and only if the collective reaches a consensus to allow the CA to join, will the CA be admitted. The same process holds for collectives which join other collectives.

There are a number of good questions that arise here. How is this policy expressed? How is it evaluated? How does a collective reach a consensus? While these questions are all important, they are all orthogonal to the idea we are presenting. We have provided a framework in which to explore a range of possible solutions. In our prototype, we rely on the presence of a 4758 running our software. As noted earlier, the 4758's outbound authentication facilities [16], or the remote attestation features of Trusted Computing Platform Alliance (TCPA) or Palladium could provide more assurance, in that they could verify things such as code load and CA software.

3.2 Simplifying Invariants

Our algorithm follows several rules that constrain and simplify the problem.

In order to maintain the property that verification may be done in $O(logV)$, we have designed our algorithm to maintain the invariant that there are no cycles in the connection graph produced by the connection network of CAs. These connections are accomplished using the protocol TPA Layer.

We maintain this invariant because if we were to allow cycles at this layer, we would break the hierarchical structure by transforming it from a tree into a less-structured graph. Breaking the hierarchical structure would have the following two implications:

First, to perform an efficient path verification algorithm in this graph, the algorithm would need to locate the shortest correct (i.e. matching the certificate chain) path. This would take longer than $O(logV)$ in the average case.

Second, any such algorithm would require state to be maintained so that the shortest correct path may be calculated without having to account for the time it takes to discover the topology in real time. This could be accomplished by implementing a "smarter" routing algorithm in the TPA Layer (e.g. reverse path forwarding [17]). Because we maintain our no-cycle invariant, we can instead use simple broadcasting. Alternative schemes that relax this invariant are an area for future work.

In addition, our algorithm maintains simplifying restrictions on the communication that occurs between two collectives.

First, at least one collective must be a *root collective* (the root node in some virtual hierarchy). Without this restriction, intra-collective connections would break the tree topology by introducing cycles. This can be seen by envisioning two unconnected trees. If a leaf node in one tree makes a connection to a leaf node in another tree, no cycle is introduced. But if their parents also connect, a cycle is formed. By forcing at least one node in the connection to be the root node, we alleviate the formation of cycles. If neither collective is a root collective, the protocol will not allow the connection.

In addition to breaking the topology, violating this restriction is a potential attack, whereby an attacker tries to join a number of collectives in an attempt to gather pieces of the private key privilege. As noted, one possible defense is to utilize the IBM 4758's

outbound authentication scheme as a basis for authentication. Other secure hardware solutions, such as the TCPA, may have similar solutions, but are perhaps more easily compromised than a FIPS 140-1 Level 4 validated device, such as the IBM 4758.

Second, nodes which collaborate in cryptographic computations for one collective may not collaborate for another. Allowing nodes to collaborate for two collectives simultaneously forces that node to hold two separate trust chains and breaks the hierarchical constraints.

We considered having the algorithm maintain a "balance invariant" on the hierarchy (e.g., each operation would maintain some balance property in the tree). However, this approach could result in large changes in the topology when a single node joins the network. We do make an assumption, however, that the nodes join and leave the network in a random fashion, resulting in a randomly built tree. It can be shown that randomly built trees have a height of $O(logV)$, and a worse case height of V. The possibilities of enforcing a balance invariant is an area for future work.

Cryptography In order to meet our claim of increased resilience, it is necessary for our scheme to require collaboration in order to perform cryptographic computations.

We take the approach that pieces of the private key privilege are scattered throughout the collective, as noted earlier. Many cryptographic techniques can enable this behavior, such as secret sharing and cooperative signature schemes [18]. These schemes are secure, but complex to implement. We discuss implications of some of these techniques in Section 3.4, and discuss Multi-Party RSA, as we plan to eventually implement it.

For simplicity, we considered the naive method of secret splitting [19]. When a request is made to sign something, the key is discovered by the host which received the request by broadcasting to the collective and ordering the pieces of the key. This is a *transient* operation in that the key is not stored at the host. Once the operation has been performed, the host forgets the key. It should be stated that this operation is *dangerous, as it exposes the assembled key*. In reality, assembling the key inside secure hardware—that could be trusted to protect it from malicious insiders—could provide some level of protection, as could using a better threshold scheme which does not require the key to be assembled at all.

3.3 The Algorithms

Our prototype maintains the invariants put forth in Section 1.2. The client and server actions guarantee the first invariant by eliminating cycles in the topology. We argue that the elimination of cycles is key to allow for efficient validation. The algorithms maintain the second invariant by enforcing that parties which negotiate a secret only store a fraction of the privilege. This implies that the secrets are distributed among members of the collective.

The server action is responsible for accepting connection requests, authenticating them, and deciding whether the two parties need to negotiate a secret. This decision is based on whether one of the parties has a portion of the private key privilege. The presence of such an entity in either party implies that at least one of the parties belongs to a collective and the other one is joining. The absence of this portion of the privilege

implies that a new secret must be negotiated, which in turn, means that a new collective is being formed.

The client action is called from an outside entity (i.e. user code), and is essentially making the same decision as above. The added burdens of avoiding cycles and enforcing assumptions about communication between collectives belongs to this action.

The validation procedure's sole responsibility is to determine whether some trust chain it receives is valid. This is done by traversing the list from the front (trust point) to the rear, and validating each node. The validation for any node is done by the Verify call. If Verify is successful for every node in the chain, then the validation procedure will return true.

Pseudocode for the algorithms can be found in the appendix of our preliminary report [20].

3.4 Analysis

In order to meet our claims of increased resilience and efficiency, we need to establish the following:

Structural Correctness The client and server actions maintain the negotiated secrets in a hierarchical, acyclic fashion. This is necessary to get $O(logV)$ average running times for the Validate procedure.

Privilege Distribution The functions maintain the property that the private key privilege and collaborating parties for each collective are distributed throughout the collective.

Structural Correctness The notion of structural correctness is used to show that the client and server actions maintain the private key privileges in a hierarchical, acyclic topology.

The hierarchy is maintained in two ways. First, the original two parties to connect form the root collective. As additional nodes join one of these two nodes, they are integrated into the collective as they are at a distance of one from one of parties maintaining the private key privilege for the collective.

As nodes make connections with collective members which are not maintaining the private key privilege, new collectives are formed. It is worth mentioning that a node which does not belong to a collective is the root of a virtual hierarchy which contains only itself.

Second, if there is a connection established between collectives, at least one of the collectives must be a root collective. If this were not the case, it would be possible for two leaf or internal collectives to join, resulting in every node in both trees to be reachable from two different trust roots, forming a cycle. This is exactly what we are trying to avoid, as this is the type of situation which leads to validation algorithms having to try multiple paths from an end point to a trust point.

The algorithms maintain a topology which avoids cycles. Each node in every collective maintains a *my_root* variable which is set to the root collective. This variable is

managed to always contain the node's root collective. As nodes attempt to make connections, they check this so as ensure that they do not attempt to make connections with nodes which already belong to the same tree.

Secret Distribution Secret distribution is the principle means by which we meet our claim of increased resilience. As noted earlier, threshold cryptography provides many tools. We discuss two.

Secret Splitting The scheme we implemented in our initial prototype is perhaps the simplest, and the weakest. The client and server actions distribute the keys across the collective in such a way that they can be correctly reassembled, and used to sign statements from the collective.

Splitting the private key into x pieces and re-assembling them when the collective needs to sign a statement does not invalidate the key. This technique is referred to as *Secret Splitting* [19], and for our prototype, we let $x = 2$.

One problem with this scheme is that we do not mandate redundancy of the key fragments. If Alice and Bob each hold a fragment and Alice has a power outage, the collective can no longer sign statements, at least until a new key can be established (which invalidates all the outstanding signed statements), or Alice powers up again.

The second problem with this scheme is that the key must be reassembled to be used. The only justification for this (albeit a weak one) is the fact that we have secure hardware. Without such machinery, this would totally expose the private key for a short time.

Secret sharing [11] would eliminate the first problem but not the second. Multi-Party RSA would eliminate both, which is why we plan to implement it in our next prototype.

Multi-Party RSA In opposition to a transient reassembling of the key and letting the result sign some statement, we envision a scheme which sends the statement around to each node holding a key fragment, and a portion of the signature being applied at that node. We plan to eventually use some instance of Multi-Party RSA to employ this technique in our system [18, 10].

Due to the algebraic properties of RSA, the algorithm lends itself to collaborative signature schemes quite naturally (an idea first proposed by Boyd [9]). Since then, the cryptographic community has generated a number of methods and protocols which utilize these properties. Samples of some such protocols and proofs of their security are discussed by Bellare and Sandhu [10] and Tsudik et. al. [21]. (This is by no means a complete list.)

Practically, using Multi-Party RSA in our system would allow a subset of members in the collective to possess key fragments, but would relax the assumption that they key is reassembled. The message is instead broadcast to the collective and comes back signed by the keyholders.

4 The Trusted Peer Access Layer

The TPA implements a protocol for trusted peers which allows them to communicate in a secure fashion. By secure, we mean that all parties mutually authenticate one another,

and that all traffic is encrypted by the secure coprocessor in such a way that an adversary could not discover the plain-text of the message — *not even if the adversary is the host* (i.e., the computer which houses the coprocessor). The protocol need only provide a decentralized means to locate items stored among those participating in the network (e.g. Gnutella) [22].

Loosely, the TPA Layer is a peer access layer running in secure hardware (the IBM 4758 Secure Coprocessor). The protocol is implemented across two communicating programs, one running on the host and the other residing in the card.

The host code is responsible for 1) implementing a command line interface which allows users (or other programs) to issue commands, 2) connection management between nodes over standard sockets, and 3) handing the TCP payloads to the card for processing and putting response packets from the card onto a socket.

The card code is where the protocol's packet processing logic resides, as well as the routing tables and secrets. The idea is that the card manufactures outgoing packets, encrypts them using secrets negotiated by it and another coprocessor in the network, and sends a chunk of ciphertext along with a socket number to the host so that it may place the ciphertext into a TCP payload and fire it to the intended recipient. When a packet arrives, the host program pulls the ciphertext out of the TCP packet and sends it to the card for processing.

The following is a brief discussion of the three major phases of development that drove our prototype implementation.

Peer-to-Peer. Our first task was to evaluate existing true peer-to-peer protocols that allowed for distributed location without the aid of a central server (like Napster). Gnutella was immediately appealing due to its simplicity, community, and availability of documentation and open source implementations.

It is important to understand what exactly Gnutella is and what it is not. Gnutella is a protocol and nothing more. In v0.4 (the base specification), Gnutella defines five packet types (called descriptors), a format for headers, and six rules for routing. Gnutella is only used to locate files across a network, transfers are done out of band (usually over HTTP).

However, Gnutella is not an implementation of this protocol. There are several implementations in existence, some of which add to the basic protocol, but they implement at least the core functionality described above [23].

We chose to use the core protocol as well as it seemed to fit our needs (actually, the "Push" descriptor type exceeds our needs, so we eliminated it), and could help reduce our time to prototype.

Secure Hardware. The next task was to find a fairly mature code base that implemented an open source Gnutella *servent* (SERVer + cliENT). Our constraints was that it should run on Linux, and be command line driven in order that we may pipe commands to it (something GUI based schemes lack).

We chose Gnut v0.4.25 because it met our requirements, was well documented, and professionally coded [24].

We then undertook the task of finding which pieces of Gnut stayed on the host and which went to the 4758. As stated above, the socket management code remained on the

host, and the packet logic and routing tables were ported to CP/Q++ (the native OS of the 4758).

At the end of this phase, we were able to observe 4758-enabled machines store strings and using the command line interface, were able to let other nodes locate them.

Adding Armor. In order to meet our definition of resilience, we had to implement a protocol for authentication and encryption, using the native cryptographic services provided by the 4758.

First, we consider authentication. The first element of our definition of resilience is that nodes must have a way to mutually authenticate one another. Bird et al. [25] explain that nonce based protocols are most secure, and since the 4758 provides a random number generator, we decided to go this way. We ended up implementing FIPS 196, which is essentially the core of most authentication schemes used in practice (e.g. Secure Sockets Layer) [26].

Second, we consider encryption. Once nodes have authenticated, the initiator sends four 3DES keys generated by its 4758 to be used for further encryption of all traffic between the two parties. Two of the keys are for encrypting messages and the other two are used for constructing a keyed Message Authentication Code for each message. We chose DES/3DES because it is fast.

5 Summary and Future Work

We are currently in the process of implementing the prototype, and once complete, we intend to make it available for public download. The TPA is lacking encryption support for all traffic. However, we do currently support authentication and are able to locate strings (which would represent cryptographic keys) across machines in the lab. The VHL is currently being implemented.

As it turns out, the result of this work has led to many more questions. In its current state, we plan to show proof of concept. As future work on this project progresses, we plan to address some of the questions that have been raised in order to evolve the system past being just a proof of concept. We are considering many ways to enhance the architecture.

One direction is to examine data structures other than trees. Balanced trees (e.g. AVL or Red-Black trees), and directed acyclic graphs could possibly lead to better solutions.

Another direction is to examine different routing protocols in the TPA. Specifically, reverse path forwarding or some other protocol which is a little smarter than just broadcasting could be interesting.

Our current architecture uses secret splitting, but as mentioned, we plan to extend the prototype to use Multi-Party RSA, allowing the message to travel around the collective to be operated on instead of the key being reassembled at one machine.

We plan to make much use of the virtual hierarchy technique in our current NSF-funded Marianas project [22], which explores using peer-to-peer techniques and secure hardware to build survivable trusted third parties.

References

1. Elley, Y., Anderson, A., Hanna, S., Mullan, S., Perlman, R., Proctor, S.: Building certification paths: Forward vs. reverse. In: Network and Distributed System Symposium Conference Proceedings. (2001)
2. Housley, R., Polk, T.: Planning for PKI. Wiley (2001)
3. Polk, T., Hastings, N.: Bridge certification authorities: Connecting b2b public key infrastructures. In: PKI Forum Meeting Proceedings. (2000)
4. Kaufman, C., Perlman, R., Speciner, M.: Chapter 15. In: Network Security - Private Communication in a Public World. 2nd edn. Prentice Hall (2002)
5. Ellison, C.: Improvements on conventional pki wisdom. In: 1st Annual PKI Research Workshop. (2002) 165–175
6. Hubaux, J., Buttyan, L., Capkun, S.: The quest for security in mobile ad hoc networks. In: Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing. (2001)
7. Zhou, L., Haas, Z.: Securing ad hoc networks. IEEE Network (1999) 24–30
8. Wilson, R.: Introduction to Graph Theory. Addison Wesley (1997)
9. Boyd, C.: Digital multisignatures. In: Cryptography and Coding. Oxford University Press (1989) 241–246
10. Bellare, M., Sandhu, R.: The security of a family of two-party rsa signature schemes. `citeseer.nj.nec.com/bellare01security.html` (2001)
11. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979)
12. Smith, S., Weingart, S.: Building a high-performance, programmable secure coprocessor. Computer Networks **31** (1999) 831–860 Special Issue on Computer Network Security.
13. Anderson, R., Bond, M.: Api-level attacks on embedded systems. Computer (2001)
14. Clulow: (2002) Personal Communication.
15. Rabin, T.: A simplified approach to threshold and proactive rsa. In: CRYPTO. (1998)
16. Smith, S.: Outbound authentication for programmable secure coprocessors. In: 7th European Symposium on Research in Computer Science. (2002)
17. Tanenbaum, A.: Computer Networks. Third edn. Prentice Hall (1996)
18. Simmons, G.: An introduction to shared secret and/or shared control schemes and their application. Contemporary Cryptology: The Science of Information Integrity (1992) 615–630
19. Feistel, H.: Cryptographic coding for data-bank privacy. Technical Report RC 2827, IBM Research (1970)
20. Marchesini, J., Smith, S.: Virtual hierarchies - an architecture for building and maintaining efficient and resilient trust chains. Technical report, Dartmouth College (2002) Available at `www.cs.dartmouth.edu/tr/ncstrl.dartmouthcs/TR2002-416/`.
21. Boneh, D., Ding, X., Tsudik, G., Wong, C.: A method for fast revocation of public key certificates and security capabilities. In: 10th USENIX Security Symposium, USENIX (2001) 297–308
22. Nicol, D., Smith, S., Hawblitzel, C., Feustel, E., Marchesini, J., Yee, B.: Survivable trust for critical infrastructure. In: Internet2 Collaborative Computing in Higher Education: Peer-to-Peer and Beyond. (2002)
23. Clip2: The gnutella protocol specification v0.4. (`www.clip2.com`)
24. Pieper, J., Munafo, R.: Gnut documentation. (`www.gnutelliums.com`)
25. Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., Yung, M.: Systematic design of a family of attack-resistant authentication protocols (1992)
26. U.S. Dept. of Commerce / National Institute of Standards and Technology: Entity Authentication Using Public Key Cryptography. (1997) FIPS PUB 196.