An Overview of PKI Trust Models

Radia Perlman
Sun Microsystems

Abstract

If Alice and Bob each know their own private key and the other's public key, they can communicate securely, through any number of public-key based protocols such as IPSec [1], PGP [2], S/MIME [3], or SSL [4]. However, how do they know each other's public keys? The goal of a public key infrastructure (PKI) is to enable secure, convenient, and efficient discovery of public keys. It should be applicable within as well as between organizations, and scalable to support the Internet. There are various types of PKI that are widely deployed or have been proposed. They differ in the configuration information required, trust rules, and flexibility. There are standards such as X.509 [5] and PKIX [6], but these are sufficiently flexible so that almost any model of PKI can be supported. In this article we describe several types of PKI and discuss the advantages and disadvantages of each. We argue against several popular and widely deployed models as being insecure, unscalable, or overly inconvenient. We also recommend a particular model.

The term principal is commonly used in the literature for any client of the public key infrastructure (PKI), whether it be human, server, client machine, or something else. A principal is anything that needs to authenticate itself or verify the identity of another principal. Also, for clarity, sometimes we use names such as Alice or Bob for principals. This enables the use of pronouns to make explanations shorter and cleaner. How do principals learn each other's public keys? Various methods can be insecure or so inefficient as to be impractical in a large network:

- Configure each principal with the public key of every other principal, in case they need to communicate. This is clearly unworkable in a large network.
- Widely publish public keys, such as providing a Web site or newspaper on which principals can advertise their keys. This is insecure. Although public keys do not need to be kept secret, and in fact wide knowledge of a principal's public key is desirable, the security problem is that Bob must know for certain that a particular key really does belong to Alice. If Bob can be tricked into thinking that Trudy's public key is really Alice's, Trudy can impersonate Alice to Bob.

The accepted solution is to have trusted nodes known as certification authorities (CAs) digitally sign data structures known as certificates that state the mapping between names and public keys. If Bob trusts a particular CA and knows that CA's public key, he can securely know Alice's public key if he can obtain a certificate signed by that CA certifying Alice's public key as belonging to the name Alice.

In many cases it is unrealistic to expect Bob to be configured with the public key of the CA that certified Alice's key. Therefore, it is necessary for Bob to obtain a chain of certificates. For example, suppose Bob knows the public key of CA1 and trusts CA1. Bob might obtain the following set of certificates:

- [CA2's key is P2] signed by CA1
- [CA3's key is P3] signed by CA2
- [Alice's key is P4] signed by CA3

Mathematically this is straightforward, once Bob has this set of certificates. However, how does Bob know CA1's key? How does he obtain the particular set of certificates that so neatly create a secure path to Alice's public key? Should all chains be trusted if they mathematically create an unbroken path to the target name? In this article we explore various methods. Principles we advocate are:

- Security is not "one size fits all." The security involved in certifying keys of NSA employees is likely to need to be higher than that involved in certifying keys for users of the local public library.
- There is no perfect solution. It is better for the world to deploy something that will work in most cases than delay deploying anything until a perfect solution is found.
- Security needs to be convenient or users will circumvent it.

A Single-CA Model

This model consists of a single CA for the world. Every piece of equipment would come configured with knowledge of that CA's public key, or have that CA's public key embedded in the hardware. All certificates would need to be obtained from the organization which runs that CA. This is certainly the simplest strategy.

What's Wrong with It?

- There is no organization universally trusted by all countries, companies, universities, political organizations, and so on.
• It is inconvenient, insecure, and expensive to obtain a certificate from a distant and unrelated organization. A common mechanism for doing so is e-mail. Not only is e-mail (before the certificates are in place) insecure, but it is difficult for the CA to ensure that the user requesting the certificate is really the name in the requested certificate.
• It is good security practice to change keys periodically. This might be because advances in mathematics (such as factoring) make it advisable to migrate to a larger key size, or it might be good to periodically change the CA's key to limit damage in case the key were compromised without anyone's knowledge. Or it might be necessary to change the key in case it is suspected that the CA's key has been compromised. However, with the one-CA model, if that CA's key were ever changed, the entire world would need to be reconfigured. Hardware with the CA's public key embedded would need to be replaced.
• If one organization is given a monopoly on granting certificates, there is the danger that once the world becomes dependent on the technology, the organization with the monopoly will charge excessive fees for certificates.

Despite these shortcomings, this model of PKI has been advocated and implemented, usually by organizations that intend to base their future revenue on collecting fees for granting certificates.

A Single-CA Plus RAs Model

This model, like the one above, consists of a single CA and all principals configured with its public key; all certificates are signed by that CA. However, there are also multiple registration authorities (RAs) trusted by that CA to verify the mapping between a name and a key and send a signed request to the CA. A user need not directly request a certificate from the CA. Instead, the user can interact with an RA, which authenticates the user and the user's key and sends a signed request to the CA for the CA to create a certificate for that user with that key. If the CA receives a validly signed request from the RA, it grants a certificate. To principals verifying certificates, this looks just like the single-CA model.

This is a partial solution to the problem in the single-CA model that everyone needs to get certificates from one organization. With multiple RAs, there would presumably be an RA more conveniently accessible. It still has the nice property that principals only need to know the single CA key. Only the CA is aware of the RA keys.

Revocation of an RA (when its key has been compromised, or the operator is no longer trusted) is very simple. The single CA is merely told to stop honoring requests from that RA.

The other disadvantages of the single-CA scheme still exist in this model, however.

An Oligarchy of CAs

This model is similar to the single-CA model, except that instead of being configured with the public key of a single CA, everything is configured with a set of keys, so there are perhaps dozens of organizations from which one can obtain certificates.

The advantage of this over the single-CA model is that competition among trusted CAs should prevent abusive pricing for obtaining certificates.

However, this model is less secure than the single-CA model. In the single-CA model, if the one key were compromised, everything would become insecure because whoever stole the CA's private key could generate bogus certificates allowing anyone to impersonate anyone. The security of the oligarchy model depends on all of the configured keys remaining secure. Compromise of any of the dozens of keys is as serious as compromise of the single key in the single-CA model.

In practice it is not necessary to compromise one of the legitimate CAs (e.g., by obtaining its private key, gaining physical access, or bribing the CA operator) in order to compromise security. It is common for browsers to be shipped with a set of CA public keys, and allow users to configure that set by adding or deleting CA public keys. Naive people might assume that a publicly available workstation, such as one in an airport lounge or hotel room, would be secure. However, it is easy to configure it with bogus CA keys and impersonate the rest of the world to the naive user of the machine. Changing an obscure piece of configuration on a user's machine makes it attackable over the network. This configuration can be done with physical access to the machine, or tricking an impatient and naive user into agreeing to add a new CA key to the list of trusted CAs after visiting a site with a certificate signed by a CA key unknown to that browser.

In theory this issue is no different from installing malicious code on a publicly accessible machine, and there really is very little one can do to protect oneself against that. But in practice it is much more difficult to install new code than to modify the configuration, especially when the configuration is designed to be something anyone typing at the machine can change.

Configured Plus Delegated CAs

This model, implemented in current browsers, is similar to the models in earlier sections, the only difference being that the CAs whose keys have been configured into the users' workstations can sign certificates authorizing other CAs to grant certificates. We call the CAs whose keys have been configured the configured CAs, and the CAs who have been authorized by those CAs to act as CAs delegated CAs. Both the configured and delegated CAs are completely trusted, and any certificate from any of those CAs (including the chain which links that CA to a configured CA) will work. Some implementations place a restriction on the number of CAs in the chain, usually by having the configured CA mark a path length restriction in the certificates it issues to delegated CAs.

This model enables users to obtain certificates from more places, since they no longer need to get a certificate directly

Figure 1. A single-CA model.

Figure 2. A single CA plus RAs.

Figure 3. An oligarchy of CAs.
from one of the configured CAs. Instead, they can obtain a certificate from a delegated CA.

This model is more convenient and somewhat more secure for users to obtain certificates, because given that there are now a lot more CAs, it is likely that user Alice can find a CA sufficiently nearby that she can physically visit it, making it easy to securely give the CA Alice's public key. Also, Alice can be accompanied by a person the CA operator knows and trusts who can vouch for Alice's identity.

However, other than that advantage, it still has the shortcomings of the other schemes. Any compromise of any CA completely compromises security. If the compromised CA is one of the delegated CAs, in theory the certificate authorizing that CA can be revoked. But the compromise may go unnoticed for a significant period of time, and it is unfortunately not uncommon for PKIs to ignore the revocation issue. So this model can be even less secure because there are more CAs, and theft of any of their keys can enable the thief to impersonate anyone to anyone.

The introduction of delegated CAs has some of the same advantages as the introduction of RAs described earlier. In both cases there are more places from which to obtain certificates. In both cases principals do not need to be configured with the public keys of all places from which certificates might be obtained (RAs or delegated CAs).

With RAs it takes longer to obtain a usable certificate, since the RA must communicate with the CA, and it is the CA that actually issues the certificate. With delegated CAs the certificate chain is longer, and verification is therefore less efficient.

Anarchy

This is the model used by the original, public domain PGP. The idea is that each user starts off by configuring public keys they have securely learned out of band. Then they obtain certificates through a number of other means, such as e-mail or downloading certificates from public databases. Massachusetts Institute of Technology has such a database of certificates. It is common whenever there are large gatherings of computer-oriented people to have PGP key signing parties with elaborate rituals involving people in the room reading off digests of their public keys, previously exchanged via e-mail, to ensure that the e-mail arrived intact. If you know the person, you sign a certificate for them.

This approach does not scale beyond a relatively small community of trusted individuals. Imagine if it were the PKI of choice for the Internet. How big would the database of certificates have to be? Assume a conservative estimate of 100 million Internet users, each having signed 10 certificates. That would be a database of a billion certificates. It would be very difficult to search such a database for a path.

It can be done. It is very similar to network routing, and Dijkstra’s algorithm [7] can be used to search for all certificates reachable from the starting trusted set. But the algorithm involves, in the worst case, examining every certificate.

This is already unworkable, but there’s a second fatal problem. Even if by some miracle you were able to find a path that mathematically created a chain from a key you trust a priori to the target name, how would you know whether you could trust that chain? The application using this model could put up a little box on your screen informing you that the chain goes from your friend George to Amy to William to Carol to David. You’ve never met William or Carol. Should you trust them to sign certificates? Well, if you trust your friend George, you might assume he’d never sign a certificate for someone who wasn’t completely trustworthy, not only to sign certificates, but to only sign certificates for trustworthy people. But in practice you cannot safely make such assumptions. Ordinary people are not trained to be that careful, and with a large community of users, there will be some percentage of people who will purposely muddy the waters with bogus certificates.

Also, it makes sense to allow trusted users to sign certificates for untrustworthy people. Why shouldn’t Charles Manson have a public key, and why shouldn’t the world be assured of the mapping between his name and his key? But that does not mean you should trust a certificate signed by Charles Manson which asserts the key of the U.S. President. In PGP the certificate only states that the signer verifies the identity of the subject. Trust is considered a local matter. You can configure your workstation with knowledge of some keys and how sure you are of the name to key mapping, and how trustworthy you think the individual is. But there is no way for you to judge the trustworthiness of someone several links down in a chain whom you’ve never met. The trust information in the public domain PGP only applies to the first link in the chain.

One difference between this model and the one above is that in the anarchy model there is no predefined core set of configured CAs. Instead, each individual starts with a personally configured set of trusted public keys. Another difference is that in the anarchy model, arbitrarily long chains are usually allowed. In the configured plus delegated CA model as commonly deployed, chains are limited in length, commonly to 3.

Top-Down

In this model, as in the one above, there is (usually exactly one) configured root CA key, and that CA can delegate to other CAs, which can delegate to others. The difference is that in this model there is assumed to be a hierarchical namespace,
and a CA is only trusted to certify name-to-key mappings for names
in the subtree of the namespace with root being that
CA. For example, there could be a CA on the premises of the
local university, say MIT, managed by playful undergraduates.
It should be trusted for certificates with names of the form
foo@mit.edu, but not for names of the form president@white-
house.gov. This rule of trusting a CA only for a portion of the
namespace is known as name subordination and makes this
model much more workable than the previous models in
which CAs were completely trusted to certify any names.

In the top-down model, each user starts out knowing the
Root key, and retrieves all the certificates from the Root
down to their own key. Alice can authenticate to Bob by send-
ing him all the certificates from the Root down to herself.
Since Bob also starts out knowing and trusting the same Root
key, he has a path to Alice's key.

This model eliminates a lot of the weaknesses of the previ-
ous model. It is computationally simple to find a path. Indeed,
there is usually only a single path to a target name. It does
not place complete trust in any CAs except the Root CA. All
other CAs are trusted only for a portion of the namespace.

This model was specified for the PEM electronic mail stan-
dard [8]. The PKI model was specified in [9]. The original
(pre-RFC) PEM PKI hierarchy recommended a single Root
key, which everyone would know a priori and would presum-
ably never change. To act as a CA in this hierarchy, you'd
have to follow a set of policies defined by someone other than
you. In other words, security would be one size fits all. Every
organization would have to be equally careful. The hierarchy
might necessitate tamper-proof hardware for CAs, and you
could not exist in the hierarchy unless you went along with the
mandated policies and procedures.

PEM soft-ended one size fits all, and instead adopted several
sizes fits all. There was a single Root, but it certified various
policy CAs, each of which would be the Root of a different
hierarchy. To be certified in a hierarchy you had to abide by the
policies as defined by the policy CA which was the Root
of that hierarchy. One potential policy might be undefined,
which would allow anyone in that hierarchy to have any poli-
cies they wanted.

This rigid top-down model, with predefined policies and
procedures for operating CAs, prevented PEM from achieving
significant deployment.

DNSSEC [10] has a similar hierarchy, although without the
mandated policies and procedures. It is a top-down model
using name subordination. Top-down models have the disad-
vantange that the entire PKI depends on the security of the
single Root key. If it were compromised, the entire world could
be impersonated. And to change the Root key would require
massive reconfiguration of everyone.

Up-Cross-Down

This model was proposed in [11]. It assumes a hierarchical
namespace with a directory structured so that any name can be
found. It also assumes the name subordination rule. Each
node in the namespace is represented by a CA. It contains
three types of certificates:

- Down: a parent certifies the key of the child
- Up: a child certifies the key of the parent
- Cross: any node certifies the key of any other

The rule is, you start at the bottom, with your own key.
You look in the directory for cross-certificates to an ancestor
of the target name (where ancestor is the least common ances-
tor, i.e., the common prefix of your name and the target
name, or closest in the hierarchy to the target name than the
least common ancestor). If there are no crosslinks to an
ancestor, you follow the parent certificate to go up a level.
You check there for cross certificates to an ancestor of the
target name; if none, you follow the up certificate, and con-
tinue until you find a cross-certificate to an ancestor of the two
names, or reach the least common ancestor of the target
name. In either case, at that point you start down. Thus, the
rule is, you go up as many times as necessary to reach a cross-
certificate or the least common ancestor; then you go across
at most once, and then down to the target name.

For example, if Fig. 6, if Radia.East.Sun wants to discover
Steve.East.Sun's key, she starts with her own key, finds the up
certificate to the parent East.Sun, notes that she has now
reached an ancestor of Steve.East.Sun's name, and then pro-
ceeds downward from there. To reach Jeff.MIT, she'd con-
figure from East.Sun, following the up certificate to the
parent (Sun), and then finding the cross-certificate to MIT,
and from there down to the name Jeff.MIT. This model has
many advantages over the models described in the previous
sections:

- Security can be deployed within an organization without the
  need to obtain certificates from any outside organization.
- No single compromised key causes massive reconfiguration.
- There is no preordained Root organization. The entire PKI
can consist of independent intra-organizational trees loosely
  coupled with cross-certificates.
- Security within your own organization is completely within
  your own hands. If you manage your own CAs well, com-
  promised keys outside your organization will not affect
  what are presumably the most security-sensitive operations,
  namely authentication between users of your own organiza-
  tion, because the path between users in an organization
does not go outside the organization. For example, in Fig. 6
the path between Radia.East.Sun and Steve.East.Sun does
not depend on any CA other than East.Sun. Some argue
that the CAs higher in the hierarchy will be more carefully
managed and therefore more trustworthy, but no matter
how trustworthy they are, given that you still need to trust
all the CAs in the path between the least common ancestor
(or target of the cross-link) and the target name, it cannot
be more trustworthy to additionally have to trust the CAs
between the Root and the least common ancestor.

For example, in Fig. 7, in the top-down model you need to
trust Root, A, least common ancestor (LCA), and C, whereas
with the up-cross-down you only need to trust B, LCA, and C.
It might be argued that B is not in the path in the top-down
model, but if B is untrustworthy it means that you can be
impersonated to the rest of the world, which is a worse
problem than your not finding a trustworthy path to the target
name. Also, with a cross-link, say from you to C, the chain to
the target only includes C.

Flexible Bottom-Up

This model is similar in spirit to up-cross-down, but allows
more flexibility. It makes use of a PKIX extension known as
name constraints, which certify a CA, but only for the explic-
itly named portion of the namespace. The name constraints
field can include permitted names, excluded names, or both.

Although PKIX offers the name constraints in the extension, little has been written about how it would be used, or what sort of flexibility might be useful beyond up-cross-down. We describe that in this section. Again, we assume a hierarchical namespace and hierarchical directory in which any name can be looked up. But instead of a strict up-cross-down rule, you allow any hierarchy by use of the name constraints. To enforce the strict up-cross-down rule, the name constraints field would be set as follows:

- A down certificate from B.A to C.B.A would include the name constraint permitted names C.B.A, meaning all names in the subtree rooted at C.B.A.
- An up certificate from C.B.A to B.A would include the name constraint permitted names * excluded names C.B.A, which for brevity we write as permitted = * - C.B.A, meaning that B.A is trusted for the entire namespace except for C.B.A and below.
- A cross-certificate to X.Y would include the name constraint permitted names X.Y.

This model can also build the anarchy hierarchy by always using the * name constraint. When looking for paths, you don't simply follow up-cross-down, but follow every link for which the name constraints still include the target name. Although the anarchy hierarchy can be built with this technology, it will have the same path-searching problem as the anarchy model. Every certificate link must be explored because it might lead to the target name.

The flexible bottom-up model has the disadvantage that it allows unscaleable structures to be created, but it has the advantage of certain flexibility not accommodated in the up-cross-down model. We would recommend that the default name constraints used be those for the up-cross-down model, and that the flexibility be explicitly configured only when necessary.

One example where more flexibility might be desirable is to support a mesh of root CAs rather than a single root CA.

To support this structure, the usual up-cross-down default name constraints would be set in all certificates except those ones in the mesh of roots. The up certificate from Sun would contain the name constraints *= Sun (all names except Sun and below). The certificates between all the Roots (e.g., between Re and Rc) would contain * in the name constraints, meaning that each root CA trusts each other root CA for all names.

Note that the name constraints in a chain cannot allow additional names. The name constraint field on an additional certificate in the chain can only narrow the choices. So for instance, since the certificate from Sun to Ra contained the name constraints *= Sun, even though the interroot certificates contain the constraint permitted names * , a certificate for a name in the Sun namespace issued outside the Sun namespace would not be accepted by anyone in the Sun namespace.

Another example where additional flexibility beyond up-cross-down might be desirable is again illustrated in Fig. 9. Suppose Radia did not want to maintain an uplink to East.Sun, perhaps because it changed frequently and she did not want to watch for the memos announcing the changes. Instead, she depends on her friend Joe.West.Sun. He keeps careful track of the key changes for East.Sun.

This is enforced by having Radia.East.Sun issue a certificate for Joe with the name constraint *=Radia.East.Sun. Joe issues a certificate to East.Sun with name constraint *, or perhaps *-=West.Sun to prevent potential children of Joe.West.Sun from following the link to East when they really want Joe's parent. For Radia.East.Sun's purposes, the two links from Radia to Joe and from Joe to East serve the same function as the up certificate from Radia to East in the strict up-cross-down model.

Yet another example of useful flexibility is illustrated in Fig. 9. Suppose Sun does not want to choose a Root organization, or go to the effort of maintaining a lot of cross-certificates. Suppose it wishes to trust MIT, for maintaining a lot of cross-certificates and perhaps choosing a Root as well. The certificate from Sun to MIT would contain the same name constraint in this case as if it were certifying a Root, namely *=Sun.

This model has all the advantages of the up-cross-down model, provided it is not abused so much as to make searching for paths intractable, and it allows more flexible trust rules than strict up-cross-down.

Relative Names

Relative names was an idea in [11] that seems to have been abandoned by the industry, possibly due to the fact that it was not supported in earlier versions of X.509. Rather than putting a full name such as Radia.East.Sun into a certificate, the certificate would contain only the portion of the name beyond the parent's name. So the certificate signed by East.Sun would contain just the name "Radia." and the full name would be inferred by appending the relative name to the parent's name.

The advantage of relative names is that it allows mapping an entire subtree to a different portion of the namespace with minimal disruption, for example, when a company is bought by
Another idea that goes along with relative names is the ability for "aliases," where a node or an entire subtree has multiple parents. This might happen because an individual has an office in two places and two names, such as radia.east.sun and radia.west.sun. Or the multiple names might be a temporary situation to enable the old name to continue to work until all the access control information that lists the old name can be updated, after an organization has been moved in the hierarchy, say from east.sun to labs.sun.

There are challenges created by relative names, but they are probably not insurmountable. For example, if name constraints are used, should the name constraints contain full names or relative names? If full names are used in the name constraints field, there is no advantage in using relative names in the certificates because the name constraints, which are also included in the certificate, would require reissuing the certificate if the name changed.

If relative names are used in up certificates, a parent would be known simply as up or ... If a node had multiple names (multiple parents), and certified its parent's key as K1, to which parent name does the up certificate apply? If the parent's full name is in the certificate, you lose the main advantage of relative names. One potential solution to this is to put only as much of the parent's name into the certificate as necessary. So, for instance, if there are two parents whose name differs in the last element, only that final element would need to be in the certificate. Then, only if the new name separated a node from its parent would the up certificate need to be reissued.

Summary

In this article we present several widely deployed PKI models and discuss the strengths and weaknesses of each. We advocate a flexible bottom-up model which is possible to build with the PKIX standard. We also present the advantages and disadvantages of relative names, which unfortunately cannot be supported by the current standards.

Acknowledgments

I would like to thank Charlie Kaufman, Steve Hanna, Phil Rosenzweig, Anne Anderson, Yassir Elley, and Sean Mullan for reviewing and contributing many of the ideas in this document.

References


Additional Reading


Biography

RUDIA PERLMAN (Radia.Perlman@Sun.COM) is a Distinguished Engineer at Sun Microsystems. She is known for her contributions to bridging (spawning free algorithms) and routing (link state routing) as well as security (public-key-proof networks). She is the author of Interconnections: Bridges and Routers, and co-author of Network Security: Private Communication in a Public World, two of the top 10 networking reference books according to Network Magazine. She is one of the networking industry's 25 most influential people, according to Data Communications. She has about 20 issued patents in the fields of routing and security. She has a Ph.D. in computer science and degrees in mathematics from MIT.