
PKI trust models

Tim Moses

Abstract

The whitepaper discusses the need for trust models in order to build large-scale authentication infrastructures based on public-key cryptography. It describes three types of public-key authority and the trust mechanisms that operate between them. It shows how these authorities and mechanisms are assembled to form the two most common trust models: the hierarchy and the bridge. It describes how isolated hierarchy and bridge models adapt to a global public-key infrastructure containing a number of independent public-key infrastructures. It discusses path development and validation in both mainstream trust models and in the presence of certification authority key roll-over. The paper discusses the issue of certificate policy in the two mainstream models, and shows how certificate extensions are used to constrain the community of valid users in very large-scale authentication infrastructures.

1. Abbreviations

A number of important abbreviations are explained in this section.

AIA – Authority information access. A certificate extension that can convey the location of the certificates issued to the issuer of the certificate that contains the extension.

AKI – Authority key identifier. A certificate extension that contains an identifier that is unique (within the set of identifiers administered by the issuer of the certificate) to the issuer's key with which the certificate was signed.

ARL – Authority revocation list. A list of the serial numbers of current authority certificates that have been revoked.

ASN.1 – Abstract syntax notation number one. A syntax standard used to describe the contents of certificates.

CA – Certification authority. An entity that issues certificates.

CPS – Certification practice statement. A list of the safeguards operating in a PKI.

CRL – Certificate revocation list. A list of the serial numbers of current end-user certificates that have been revoked.

CSP – Cryptographic service provider. A hardware or software module that contains cryptographic functions and private keys.

CTL – Certificate trust list. A list of fingerprints of authority public keys signed by a trust list manager.

HTTP – Hypertext transfer protocol. The message protocol used in the Web.

IETF – Internet engineering task-force. The standards body that profiles X.509 for use in the Internet.

ISO – The International Organization for Standardization. The organization that (in collaboration with the ITU) standardizes X.509 PKI.

ITU – International Telecommunications Union. See ISO.

LDAP – Lightweight directory access protocol. The protocol used to access X.500 and similar repositories.

PKI – Public-key infrastructure. An authentication infrastructure based on public-key cryptography.

PKIX – Public-key infrastructure, X.509-based. The working group of the IETF that profiles X.509 for Internet use.

SKI - Subject key identifier. A certificate extension that contains an identifier that is unique (within the set of identifiers administered by the subject of the certificate) to the subject's key contained in the certificate.

TLM – Trust list manager. An authority that distributes authority public keys in the form of CTLs.

URL – Uniform resource locator. An address on the Web.

2. Public-key infrastructure

In its most basic form, a public-key infrastructure (PKI) consists of a community of principals, a community of verifiers and an authentication authority that is recognized by both. A verifier may also be a principal. But, for the purposes of clarity, we treat them separately throughout much of this paper. The purpose of the infrastructure is to allow a verifier to authenticate attributes (commonly the identity) of a principal when they communicate over an unsecured network¹. The situation is shown in Figure 1.

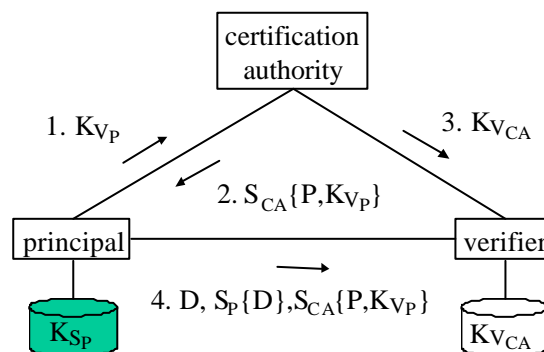


Figure 1- Single-domain PKI

¹ An unsecured network is any communications network that is not required to possess intrinsic integrity, authenticity or confidentiality properties. However, the availability of the authentication service described here depends directly on the availability of the communications network.

Authentication proceeds in the following steps:

1. The principal generates a public-private key pair $\{K_{V_P}, K_{S_P}\}$ and stores the private key (K_{S_P}) in local storage with *integrity* and *confidentiality* protection. It sends the public key (K_{V_P}) to the authority, using a trust mechanism (see below for a discussion of trust mechanisms).
2. The principal's public key and its identifying information (P) within a name-space sub-tree administered by the authority are signed by the authority, using its private key, $K_{S_{CA}}$. The resulting data structure, $S_{CA}\{P, K_{V_P}\}$, is called the principal's public-key certificate, and the authority is called a public-key certification authority or CA. The certificate is commonly returned to the principal.
3. The verifier obtains the authority's public key ($K_{V_{CA}}$) using a trust mechanism and stores it with *integrity* protection.
4. In order for the principal to authenticate itself to the verifier, it signs a data structure (D), attaches its certificate, and sends the data structure, the signature, $S_P\{D\}$, and its certificate to the verifier over the unsecured network. The verifier can then confirm that the data originated with the principal identified in the certificate.

The benefit of this authentication scheme over rival schemes rests on two factors:

- The verifier does not have to maintain any sensitive information about any principals that it may be called upon to authenticate; and
- The CA is not directly involved in transactions between the principal and the verifier.

This makes it possible for one CA to serve a very large community of principals and verifiers. And the security of the overall system can be protected in a highly cost-effective manner.

3. Large-scale PKI

In order for an authority to communicate with principals and verifiers in a cost-effective and reliable way, there must be an existing close relationship between them. Generally, authorities only have a suitable relationship with a limited community. And, commonly, PKIs are required to scale beyond such limits. Therefore, PKIs with more than one authority are required, and the authorities' keys must be shared amongst the participants in a secure manner. Trust mechanisms are used for this purpose.

3.1 Trust mechanisms

Four trust mechanisms will be considered:

Out-of-band mechanisms;

Certificate trust lists;
 Certificate request messages; and
 Cross-certification.

3.1.1 Out-of-band mechanisms

Out-of-band mechanisms include:

1. Publishing a fingerprint of an authority key² in a trustworthy location and passing the unprotected key itself over an unsecured communications network, the recipient can then verify the key using the trusted fingerprint;
2. Securing a protocol with a key derived from a random string of printable characters (the random string is shared by means of a trustworthy channel); and
3. Embedding a key in trusted software (the technique used by browsers and web servers).

3.1.2 Certificate trust list

A certificate trust list is an extension of the first out-of-band technique listed above. But, instead of publishing the fingerprint in a trustworthy location, it is secured by the signature of a trust list manager (TLM). See Figure 2.

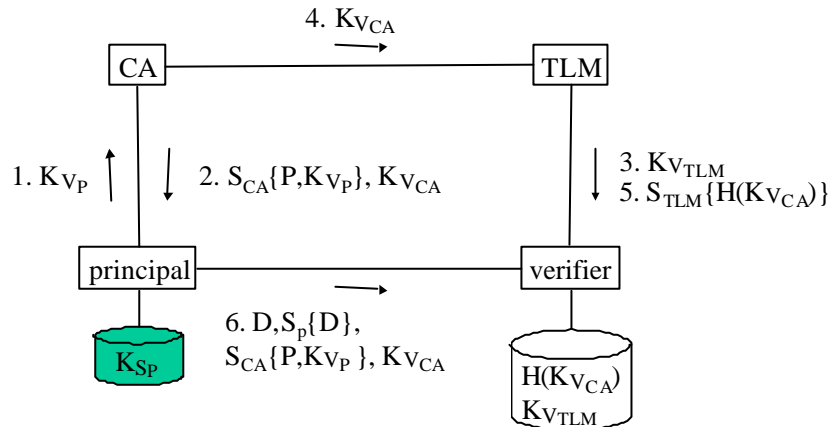


Figure 2 - Certificate trust list

The mechanism operates in this way:

1. The principal generates a public-private key pair and stores the private key in local storage with *integrity* and *confidentiality* protection. It sends the public key to the certification authority, using a trust mechanism.

² A fingerprint is a hash, or digest, of the key encoded in printable characters.

2. The principal's public key and its identifying information within a name-space sub-tree administered by the certification authority are signed by the authority and returned to the principal with the CA's public key, using a trust mechanism. The public key of the CA is commonly encoded as a self-signed certificate.
3. The verifier obtains the trust list manager's public key using a trust mechanism and stores it with *integrity* protection.
4. The trust list manager imports the certification authority's public verification key using a trust mechanism.
5. The trust list manager calculates a fingerprint of the certification authority's public key and places it in a list with other fingerprints. The resulting list is signed by the trust list manager and pushed to the verifier in the form of a certificate trust list.
6. In order for the principal to authenticate itself to the verifier, it signs a data structure, attaches its certificate and the public key of the certification authority. The verifier can verify the identity of the principal using the data structure, the signature, the certificate, the certificate trust list and the TLM's public key.

3.1.3 Certificate request message

The certificate request message is used by a registration authority (RA), as shown in Figure 3.

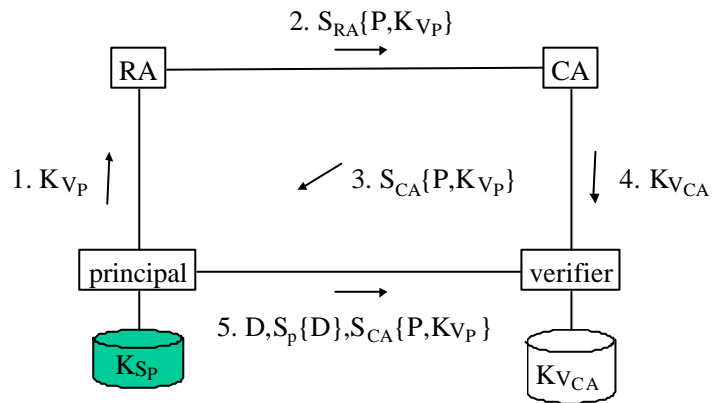


Figure 3 - Registration authority

In a preliminary step, the RA registers with the CA, as described in step one of Figure 1. The mechanism then proceeds as follows:

1. The principal generates a public-private key pair and stores the private key in local storage with *integrity* and *confidentiality* protection. It sends the public key to the registration authority, using a trust mechanism.
2. The principal's public key and its identifying information within a name-space sub-tree administered by the registration authority are signed by the registration

authority and sent to the certification authority. The signed message is called a certificate request message.

3. The certification authority verifies the signature of the registration authority. If it is valid, then it signs the public key and the identifying information of the principal, using its own private key. The resulting certificate is then commonly returned to the principal.

4. The verifier obtains the certification authority's public key using a trust mechanism and stores it with *integrity* protection.

5. In order for the principal to authenticate itself to the verifier, it signs a data structure and attaches its certificate. The verifier can confirm the identity using the data structure, the signature, the certificate and the CA's public key.

3.1.4 Cross-certification

The final mechanism, cross-certification, is shown in Figure 4.

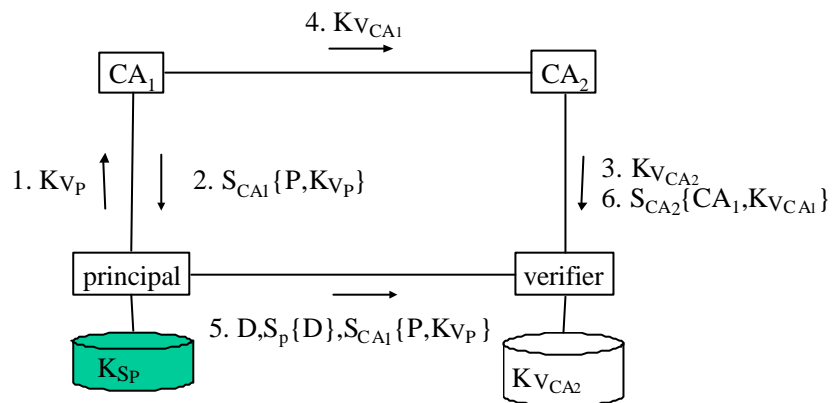


Figure 4 - Cross-certification

The mechanism works as follows:

1. The principal generates a public-private key pair and stores the private key in local storage with *integrity* and *confidentiality* protection. It sends the public key to CA₁, using a trust mechanism.

2. The principal's public key and its identifying information within a name-space sub-tree administered by CA₁ are signed by CA₁ and returned to the principal.

3. The verifier obtains CA₂'s public verification key using a trust mechanism and stores it with *integrity* protection.

4. CA₂ obtains CA₁'s public verification key using a trust mechanism. It then signs the key in a data structure called a cross-certificate.

5. In order for the principal to authenticate itself to the verifier, it signs a data structure and attaches its certificate.

6. The verifier obtains the cross-certificate from CA₂ over an unsecured network. The verifier can verify the identity of the principal using the data structure, the signature, the certificate, the cross-certificate and the public key of CA₂.

3.2 Comparison of trust mechanisms

Some of the differences between the cross-certification mechanism and the certificate trust list mechanism include:

- In cross-certification, CA₂ provides CA₁'s *key* to the verifier, not its *fingerprint*, as the TLM does. So, CA₁'s key does not have to be conveyed in the authentication protocol between the principal and the verifier.
- Each cross-certificate contains only one authority key, whereas a certificate trust list may contain fingerprints for many CA keys.
- In the cross-certification mechanism, a CA commonly *withdraws* another CA's key by issuing an authority revocation list (ARL). Whereas, in the certificate trust list mechanism, a TLM withdraws a CA's key by erasing its fingerprint from the verifier's store.
- The cross-certification approach is fully standardized by ISO, ITU and IETF and is interoperable among a number of vendors. Whereas, the certificate trust list has not been standardized.
- The role of the trust list manager is not recorded in the transaction evidence, which comprises the transaction data, the transaction signature, the certificates and revocation information. On the other hand, the roles of both authorities in the cross-certification mechanism *are* recorded in this transaction evidence.

Some of the differences between cross-certification and the registration authority include:

- While there are IETF standards for the registration authority protocols, they are not multi-vendor tested at the present time.
- As with the TLM, the role of the registration authority is not recorded in the transaction evidence.

3.2 Evolution of trust mechanisms

All of the mechanisms described above are in routine use today. However, not all interfaces are multi-vendor interoperable, even those that conform with industry standards. But, we should expect this situation to change over the next year or so.

We should expect implementations of these techniques to reach the limits of their ability to scale in certain applications over the next year or so. Figure 5 shows those

mechanisms that are in routine use today, others that are in experimental use and yet others that are primarily theoretical. The latter categories will be explained ore fully later in the paper. Bridged PKIs and multiple-rooted hierarchies are currently being evaluated in pilot settings and we should expect these approaches to move into production use in the near future. Connected bridges are also being used in pilots in a very limited way today. We can expect it to take longer for such approaches to become commonplace in production settings.

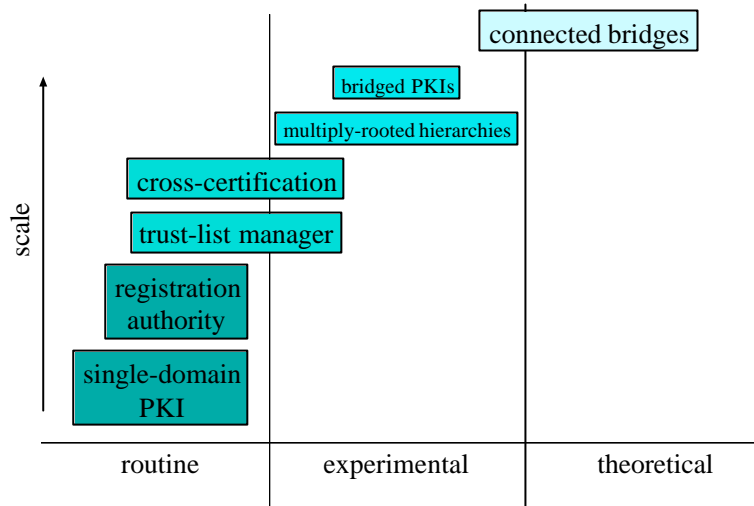


Figure 5 – Current status of large-scale PKIs

4. Trust models

Practical large-scale PKIs use a combination of trust mechanisms. But, all of them rely on one or more of the out-of-band mechanisms for initialization, including providing the public key of at least one authority to the verifier in a secure way. An authority key passed to a verifier in this way, for the purpose of validating its certificates, is called that verifier’s “trust point” or “trust anchor”.

We’ll look at two practical large-scale PKI trust models:

The hierarchical trust model; and

The bridge trust model.

4.1 Hierarchical trust model

As a matter of definition, a hierarchical trust model is one in which every key can be the subject of no more than one certificate or certificate request message. (This restriction is not usually applied to CTLs. That is to say, even in a hierarchy a key may be the subject of more than one CTL).

4.1.1 Trust mechanisms in an isolated hierarchical trust model

The mechanisms used in an isolated hierarchical trust model are shown in Figure 6. The symbol “A → B” means:

B's public key is passed to A so that entities that rely on A's public key may also rely on B's public key³;

A issues a certificate (or certificate request message or certificate trust list) for B;

or

“A trusts B”.

For the purposes of risk containment, trust is usually conditional. That is, A allows its key to be used as the basis of trust in B's key only for certain specified purposes and by a certain specified community of verifiers. We say that the key is approved for use in accordance with a specified certificate policy. It is common practice for the certificate issuer (A in this example) to dictate the terms of the certificate policy. This is because parties that rely on its public key have expectations concerning the quality or suitability of its certificates, and in order to preserve those expectations, it must impose conditions on its certificate subjects. But, at least in theory, the subject (B in the example) may dictate the policy, or the issuer and subject may negotiate a mutually agreeable policy. There will be further discussion of certificate policy later in the paper.

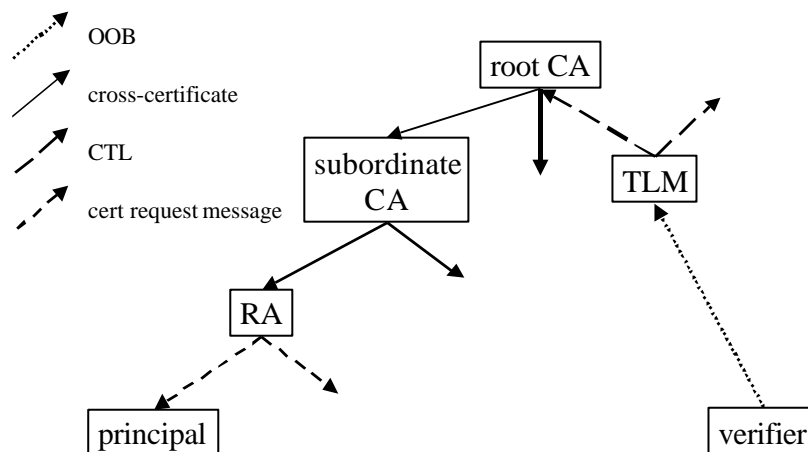


Figure 6 - Trust mechanisms in a hierarchical trust model

As can be seen from the diagram, the hierarchical trust model uses four different trust mechanisms. It uses an out-of-band mechanism between the verifier and the Trust list manager, a certificate trust list between the Trust list manager and the root CA, it uses cross-certificates between the root CA and the subordinate CA and between the subordinate CA and the registration authority and a certificate request message between the registration authority and the principal⁴.

³ To rely on an entity's public key means to validate its signatures, whether those signatures be on data or certificates.

⁴ It has been suggested that "trust is not transitive". I.e., just because A trusts B and B trusts C, it does not follow that A trusts C. But, we can see from this common example that it is common to rely on a chain of trust with at least five links. We can conclude that trust is transitive, at least under some circumstances.

It is a defining property of hierarchies that no entity key has more than one certificate issued for it. So, while the TLM may trust more than one root, each subordinate CA may be subordinate to only one root, and each principal may be subordinate to only one subordinate CA. For this reason, there is only one certification path between the principal and the root, and so the principal's certificate may be issued in the form of a chain, emanating from the corresponding root CA, and the whole certification path may be distributed in the interchange between the principal and the verifier. This would be a useful simplification if it were not for the need to distribute encryption certificates for securing store-and-forward communications. Because, in this case, the encryption certificate must be used prior to the communication, and because the communication is not bi-directional, the distribution of the certificates in the communication protocol itself is not helpful.

Of course, the hierarchical model may be extended to more levels than the two described here.

A note on terminology: to be consistent, we say that the TLM “issues a CTL for” the root and that the RA “issues a certificate request message for” the principal, just as we say that the root CA “issues a cross-certificate for” the subordinate CA. However, in all cases, the consumer of the data structure is not the subject of the CTL, certificate request message or cross-certificate.

4.1.2 Multiple hierarchies

A hierarchy operating in isolation is a somewhat artificial situation. A more realistic situation is one in which principals are required to operate subordinate to multiple roots, as shown in Figure 7. We discuss later the reasons why principals may have to operate subordinate to more than one root.

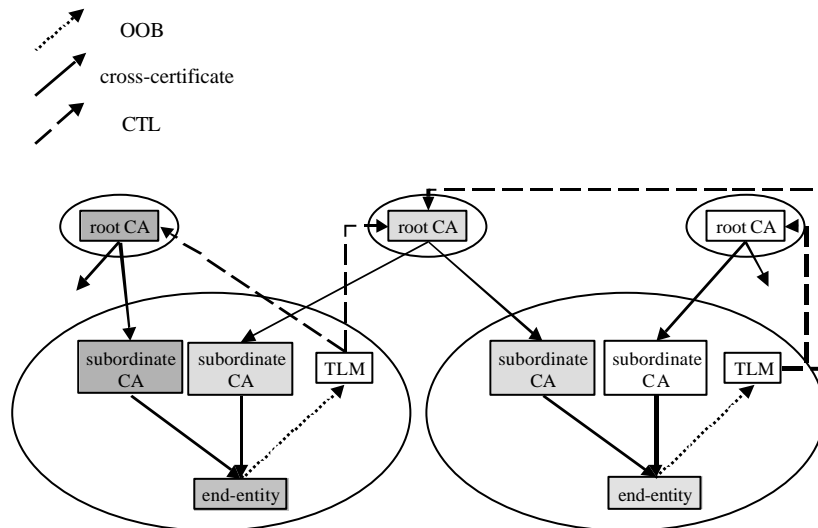


Figure 7 – Multiple rooted hierarchies

In the diagram, an end-entity is the combination of a principal and a verifier, and the shading indicates the policy under which the entity operates.

Unless the roots coordinate their policies, they will operate different policies and each will require its subordinates to operate in accordance with its policy. So, each subordinate CA operates a different policy, that of its root.

Authorities may coordinate their policies *by design* or under the command of an oversight body, such as a regulatory body. However, it is common for there to be no such coordination of policy between roots. Only the end-entity is required to operate in accordance with the policies of all roots. Subordinate CAs operate only in accordance with the policy of their single superior CA. We discuss what it means to operate in accordance with more than one policy later in the paper.

4.2 Bridge trust model

It is reasonable to ask why a subordinate CA cannot be certified by more than one root. The answer is that *it can*. But, the result is not a hierarchy, because then the subordinate CA would have more than one superior, which is inconsistent with the definition of a hierarchy. Instead, we call the result a bridge model.

4.2.1 Trust mechanisms in an isolated bridge trust model

The trust mechanisms used in the bridge model are shown in Figure 8. One can see that the bridge model is very similar to the hierarchy, shown in Figure 6. The only differences are that we have renamed the “root CA” to “bridge CA”, we have also renamed the “trust list manager” to “spoke CA” and we have also renamed the “subordinate CA” to “spoke CA”. The specific trust mechanisms used between the entities are also different: this model does not use the certificate trust list mechanism at all. In its place, it uses cross-certification.

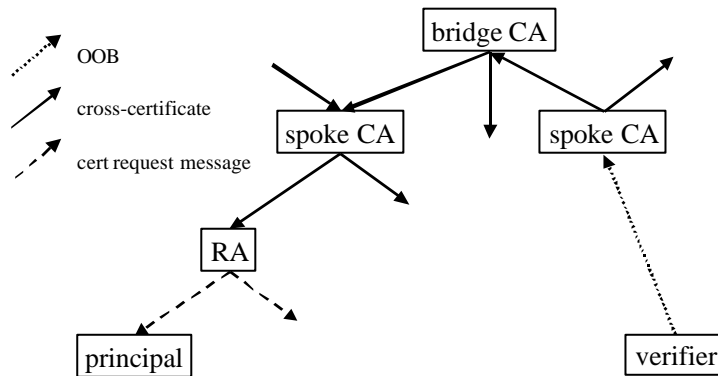


Figure 8 - Trust mechanisms in the bridge model

The essential difference, however, between the two models is that spoke CAs can be certified by more than one bridge, whereas, in a hierarchy, subordinate CAs can only be certified by one root. So, in a hierarchy, if a principal must be certified subordinate to more than one root, then it must have a separate key pair and there must be a separate subordinate CA for each root, whereas, in the bridge model, a

principal can operate “downstream” of many bridges with a single key pair and a single spoke CA⁵. Other differences between the two models are described below.

4.2.2 Multiple bridges

Just as in the case of the hierarchy, the isolated bridge model depicts a somewhat artificial situation. Figure 9 shows the more realistic situation of multiple bridged CAs.

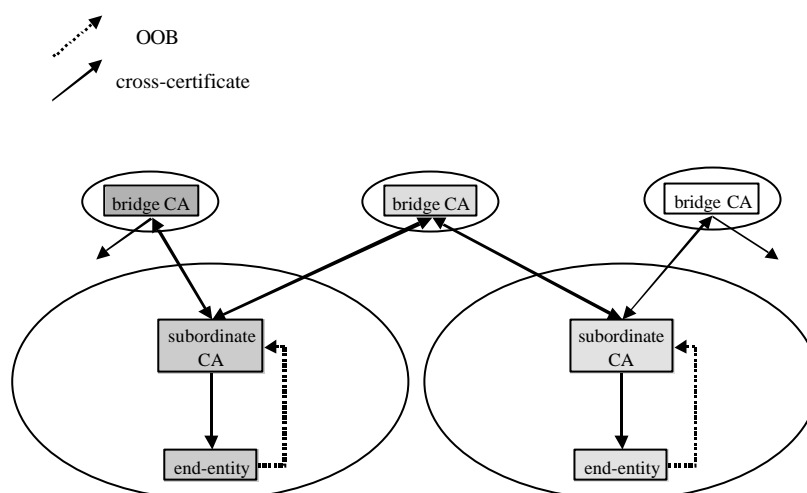


Figure 9 - Multiply-bridged PKIs

The striking difference between this architecture and the architecture with multiple *roots* is that there may be only one spoke CA per domain. Whereas, in the hierarchical model, there must be a subordinate CA for each root.

However, the single CA must operate in accordance with the policies of each upstream bridge. So, the spoke CA’s behaviour must be consistent with the policies of more than one upstream bridge.

4.3 Comparison between trust models

The main difference between the hierarchical trust model and the bridge trust model is in the way that they behave when there are multiple roots or bridges. See Figure 10.

If a principal has to operate subordinate to more than one root in a hierarchical trust model, then the part of the hierarchy that is subordinate to the root has to be duplicated for each root. This includes the principal’s certificate and private key. If, on the other hand, a principal has to operate subordinate to more than one bridge in a bridge trust model, then there is no need to duplicate anything.

As a result, if at some time after principal certificates have been issued, a new root must be introduced in the hierarchical model, then each principal has to obtain a new certificate, in the form of a chain emanating from the new root. But, if at some time

⁵ We say that one entity is downstream of another entity if the first entity’s public key is certified by the second one’s.

after principal certificates have been issued, a new bridge is introduced in the bridge model, then there is no impact on the principals. Operations that involve the cooperation of the principals are likely to be unreliable. So, this makes it difficult and costly to replace a root, possibly to the point of being impractical, whereas the cost of replacing a bridge is negligible by comparison.

Another major difference between the two models lies in the way they handle policy mapping. In the bridge model, the principal and spoke CA must operate under a single policy that is simultaneously acceptable to all of the bridge CAs. Whereas, in the hierarchical model, each subordinate CA must operate in accordance with the policy of its single superior root. Principals, on the other hand, must behave in accordance with the policy of the root under which the particular private key they have chosen is certified. An entity may act in accordance with different policies in one of two ways: it can adapt its behaviour over time to be in accordance with the appropriate policy for the time, or it can have a single set of behavior that is simultaneously consistent with all policies. The first strategy is commonly adopted for principals operating in a hierarchy and the second strategy is commonly adopted for spoke CAs in a bridge model.

In the case of the hierarchy, the verifier determines an acceptable policy for the transaction by choosing a suitable root CA from which to validate the principal's certificate. Therefore, the principal and the verifier must coordinate their choice of a root CA.

In the case of the bridge CA, the verifier determines the acceptable policy by defining the initial policy set in the path validation algorithm (see below for a discussion of path validation). Therefore, they do not need to coordinate their choice of a bridge CA. But they must ensure that they and the intervening CAs operate a policy that is suitable for the transaction.

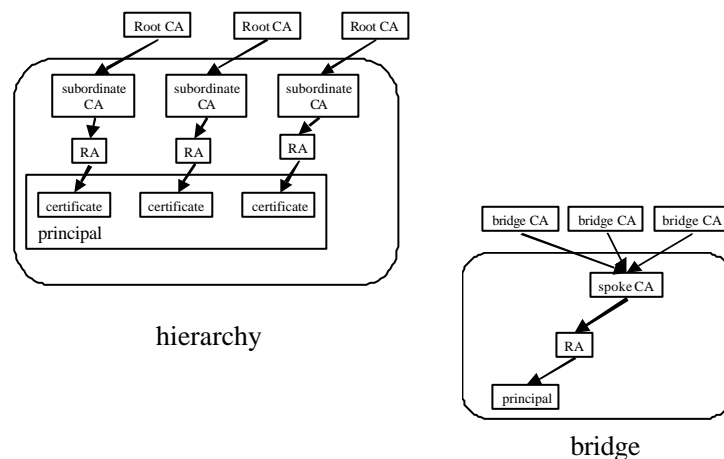


Figure 10 - Multiple nodes

Other differences lie in the approach the two models take to distribution of authority certificates and CRLs and procedures for path development and validation.

In the hierarchical trust model, each principal certificate is commonly issued in the form of a certificate chain emanating from the corresponding root. Whereas, in the bridge trust model, a repository (LDAP or HTTP) is required to distribute authority certificates and, commonly, ARLs and CRLs. In the hierarchical model, a repository is only needed to distribute principals' encryption certificates and CRLs.

The approaches to path development and validation adopted by the two models are discussed later.

4.3.1 Principals and verifiers in the same domain

While trust models were introduced in order for a verifier to authenticate a principal when they do *not* share a common authority, in most cases it remains a requirement to authenticate principals who *do* share a common authority with the verifier. So, we need to examine how the two mainstream models satisfy this requirement. See Figure 11.

When the principal and the verifier share a common certification authority, the root or bridge does not have to be present in the trust path. And, in fact, in common implementations of hierarchies, the root is not involved. This implies that the subordinate CA actually has more than one upstream authority (the root and the local TLM). But, in order to remain a “hierarchy”, only one upstream authority can be a *certification* authority. Additional upstream authorities must be TLMs.

In a *strict* hierarchy, the root is present in the certification path even when there is a common subordinate authority. However, there are few practical examples of hierarchies that operate this way, because reliance on the external root introduces additional cost and risk with no offsetting benefit.

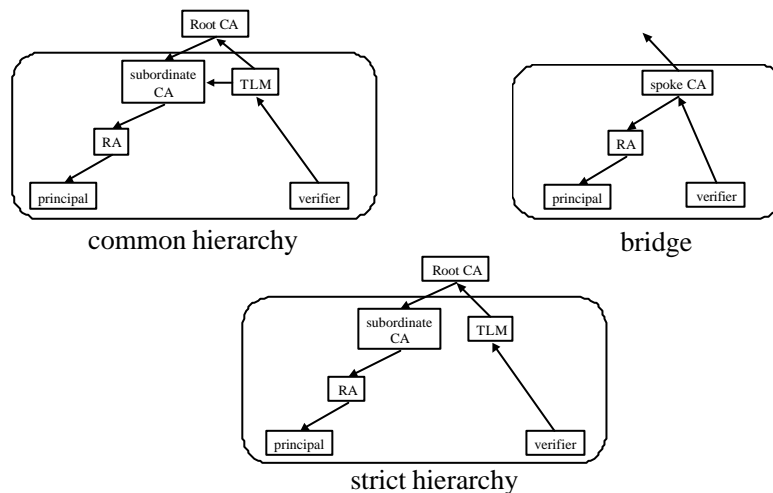


Figure 11 – Principal and verifier in the same domain

In the bridge model, the verifier directly trusts the spoke CA that issues the certificate to the local principal. So, no external authorities are involved in the trust path.

4.4 Path development and validation

There is also a marked difference in the way the two mainstream models commonly deal with the question of certification path development and validation.

4.4.1 Hierarchical model

In a hierarchy, a key can only have one superior CA that issues a certificate for it.

Because the principal is likely to have more than one certificate (one per root CA), it must choose one that can be verified by the verifier, i.e. one that is subordinate to a root whose policy is acceptable to the verifier for the particular transaction.

Path development, as shown in Figure 12, starts with the principal certificate corresponding to the private key that the principal chooses and works up the chain of certificates until one is discovered that is in the verifier's CTL. If none is found, then the principal must choose another private key. This process is repeated until a certificate corresponding to a fingerprint in the verifier's CTL is found (in which case a single complete path has been found), or there are no more principal private keys (in which case there are no complete paths between the verifier and the principal).

Following path development, path validation takes place. If the first complete path to be developed turns out not to be valid, then the process continues with the next principal private key.

We can see from this procedure that the number of paths to explore cannot exceed the number of certificates held by the principal.

Because the principal has more than one certificate, however, it has to decide which one to use, so that the verifier will accept it. Each principal certificate has a different policy, each policy is dictated by the corresponding root and if it is an assurance-style policy it may have different private key storage and activation and other requirements. So, the principal may have to modify its behaviour according to the private key that it is using. There is more discussion of certificate policy styles later in the paper.

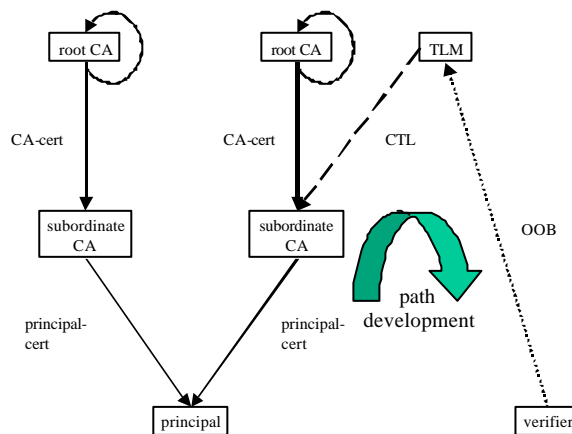


Figure 12 - Certification path development and validation in a hierarchy

A pseudo-code description of the algorithm is shown below.

```
branch:=0
REPEAT
  (
    distance:=0
    REPEAT
      (
        path[distance]:=get-cert(path,distance)
        distance++
      )
    UNTIL self-signed(path[distance]) OR on-ctl(path[distance])
    IF on-ctl(path[distance]) THEN
      valid:=validate(path)
    ELSE valid:=FALSE
    branch++
  )
UNTIL valid OR no-more-paths()
```

This pseudo-code does not only describe the function performed by the verifier. The outer “repeat” loop is actually performed by the principal. Only the function inside the outer loop is performed by the verifier. So, the pseudo-code describes the behaviour of both the principal and verifier and the negotiation that takes place between them.

4.4.2 Finding the next certificate in the path

If the certificate path is not distributed in the interchange between the principal and verifier, then the verifier needs a way to obtain the certificates in the path.

Once it has the principal’s certificate, and if the other certificates in the path are available in an X.500 or LDAP repository, then the issuer’s name from the certificate in-hand can be used as the entry name in a directory *read* operation to obtain the cross-certificate pairs attribute or CA certificate attribute in order to obtain the certificate whose subject has the same name and whose public key can verify the certificate in-hand. Then the required certificate can be found as the single value in the forward cross-certificate pairs attribute field or in the CA certificate attribute. Actually, considerations of key life-cycle management gives rise to the need for more than one certificate. But this will be discussed below.

If, on the other hand, the certificates are available on the “Web”, then the authority information access (AIA) extension from the certificate in-hand is used to perform an HTTP “get” operation. The result will be the single certificate whose subject has the same name and which contains the public key than can verify the certificate in-hand.

In a hierarchy, a certificate should be published by its subject.

4.4.3 Bridge model

Path development in the bridge model is shown in Figure 13.

As we saw earlier, in the hierarchical model, the maximum number of paths to explore is equal to the number of certificates held by the principal. By contrast, the

number of paths to explore in the bridge model is potentially unbounded. So, a procedure that relies on an exhaustive exploration of those paths is impractical and it becomes essential to evaluate and eliminate paths as development proceeds. In order to do this efficiently, it is important to place restrictions on cross-certificates so that invalid paths can be identified and eliminated early in the process. These restrictions take the form of certificate extensions, such as *basic constraints*, *name-constraints*, *certificate policy*, *policy constraints* and *inhibit any policy*, which are described later.

X.509 path *validation* must be performed in the forward direction, starting at the verifier and proceeding towards the principal. Therefore, in the bridge model, path development can be achieved most efficiently in the forward direction.

In the bridge model, principals do not need more than one certificate. In addition, self-signed certificates are not involved in the procedure.

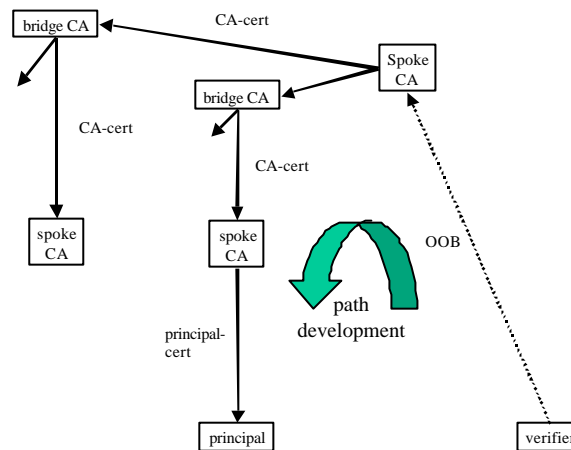


Figure 13 - Certification path development and validation in a bridged PKI

A pseudo-code description of the algorithm is shown below.

```

distance:=0
branch[0]:=0
REPEAT (
    valid:=false
    WHILE ( NOT valid AND branch-exists(distance,branch[distance]) )
        (
            path[distance]:=get-cert(path,branch[distance])
            valid:=validate-path(path)
            branch[distance]++
        )
    IF valid THEN
        (
            distance++
            branch[distance]:=0
        )
    IF NOT branch-exists(distance,branch[distance]) THEN distance--
)
UNTIL (valid AND have-principals-cert(path)) OR distance<0

```

Not shown are additional checks required to ensure that paths are not circular.

From an examination of the pseudo-code descriptions, one can see that the path development and validation algorithms for the hierarchy and bridge models are of approximately similar complexity. The complexity of the hierarchical algorithm springs from the fact that the principal must have a different certificate for each root and it has to choose a private key and certificate that will be acceptable to the verifier. It can only tell that it made the right choice through negotiation with the verifier. The complexity of the bridge model, on the other hand, springs from the fact that, even though the principal may have no more than one certificate, there may be many paths from the verifier's trust point to that certificate.

4.4.4 Finding the next certificate in the path

Given the principal's certificate, the verifier needs a way to obtain the certificates in the certification path.

If the authority certificates are available in an X.500 or LDAP repository, then the subject name from the certificate in-hand can be used as the entry name in a directory *read* operation to obtain the cross-certificate pairs attribute. Then the required certificate is one of those in the reverse cross-certificate field of that attribute. All the values can be retrieved and the acceptable ones extracted.

In the bridge model, a certificate should be published by its issuer.

4.4.5 Filtering

X.500 has a standard mechanism for specifying a subset of the values in an attribute for retrieval. This is most useful when the number of values in an attribute (e.g. cross-certificates in the *cross-certificate pairs* attribute) may be large, and the cost of downloading them all for processing by the verifier may be unacceptable. Then the repository can be instructed to return just the subset of values that meet the needs of the verifier. This facility is called filtering.

4.5 Key rollover

The fact that key pairs change over time, as the crypto-period of one key expires causing it to be replaced by another, complicates the trust model.

4.5.1 Key roll-over in a hierarchy

In a hierarchy, each CA at any given time commonly has two key pairs, each certified by its single superior CA, and those certificates have interleaved validity periods (see Figure 14). Each CA issues certificates using the key whose certificate has the latest expiry time, and the certificates it issues must be set to expire before the expiry of the certificate whose key was used to sign it. Consequently, all the superior certificates that were valid at the time of issuance of an end-user certificate will remain valid for the entire period of that certificate's validity.

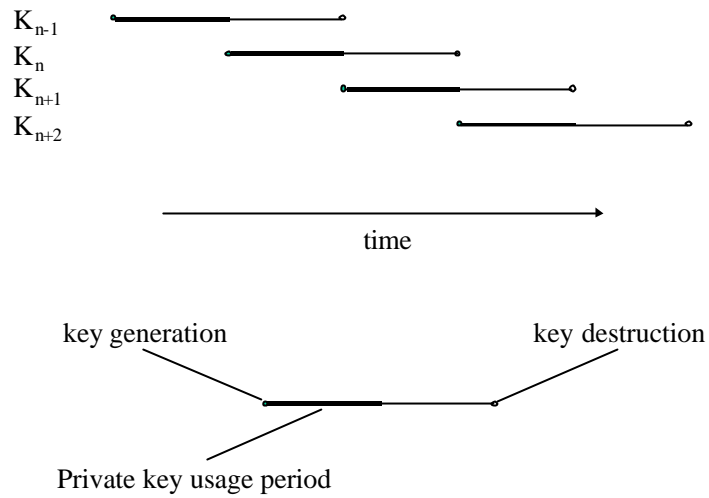


Figure 14 – Interleaved CA key validity periods

Notice that, at any given time, the CA has one valid private key and two valid public keys (and their corresponding certificates). Before a certificate reaches the end of its validity period (in fact, commonly half way through its validity period) the key holder can replace it by generating a new key and requesting that its superior issue a new certificate for that new key. The subject CA can then replace the expiring certificate in its publication mechanism with the new certificate.

In order to find a path from a principal's certificate to its corresponding root, the verifier can readily choose the next certificate in the path from the two currently published by the CA that issued the certificate in-hand by matching the authority key identifier (AKI) of the certificate in-hand with the subject key identifier (SKI) of the next certificate in the path, as shown in Figure 15. The AKI and SKI values are encoded in certificate extensions.

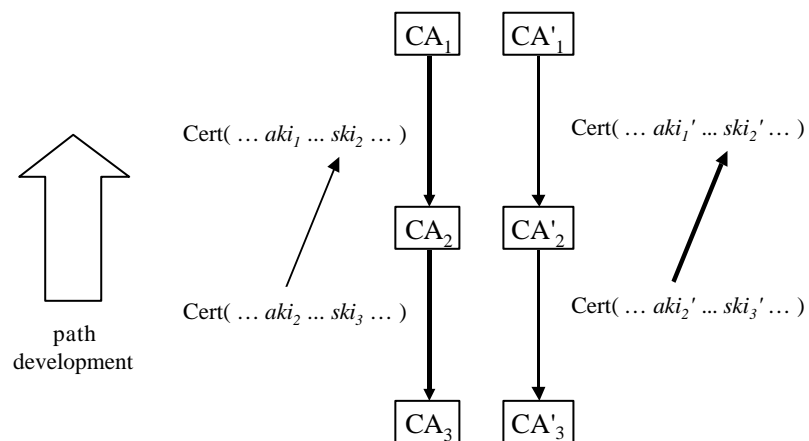


Figure 15 - Finding the next certificate

4.5.2 Key roll-over in the bridge model

In the bridge model, the situation is different because a CA's key may be certified by more than one issuer, and those certificates may expire asynchronously. Consequently, there is no guarantee that all the certificates in a path that were valid at the time a principal certificate was issued will remain valid for the entire period of that principal certificate's validity.

The procedure is shown in Figure 16. As each CA updates its key [step 2], it must have it certified and published by all the CAs that are immediately upstream, and it must use the new key to recertify and publish the existing keys of all CAs that are immediately downstream [step 3]. Finally, the upstream CAs delete from their publication mechanisms the certificates they issued for the CA's previous key and the CA deletes from its publication mechanism the certificates it previously issued for its downstream CAs' keys [step 4]. Because paths are developed in the forward direction, the downstream CA does not have to publish the resulting certificate. So, it does not have to participate in, or even be aware of, this procedure. However, it is common practice for the downstream CA to publish certificates for which it is the subject. So, it would, in fact, be involved in the key roll-over of its upstream CAs.

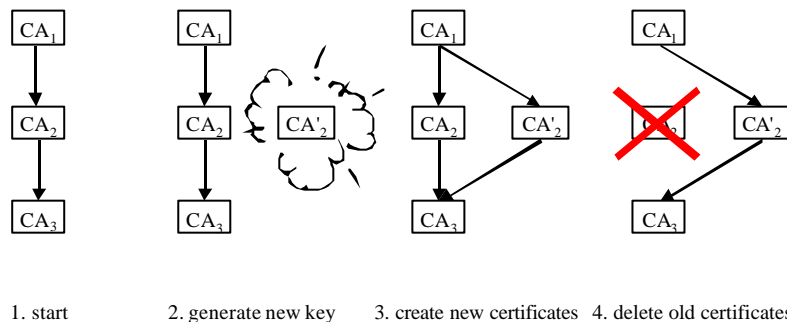


Figure 16 – CA key roll-over in the bridge model

Principals commonly store their own certificates as well as, or instead of, having them published by the CA immediately upstream. So, the approach described above would require them to participate in the key roll-over of their immediately superior CA. Because principals may not be continuously on-line, this solution is not practical. So a different approach is adopted when a CA that issues end-user certificates rolls its key over: it issues a certificate for its previous key, signed with its new key. See Figure 17.

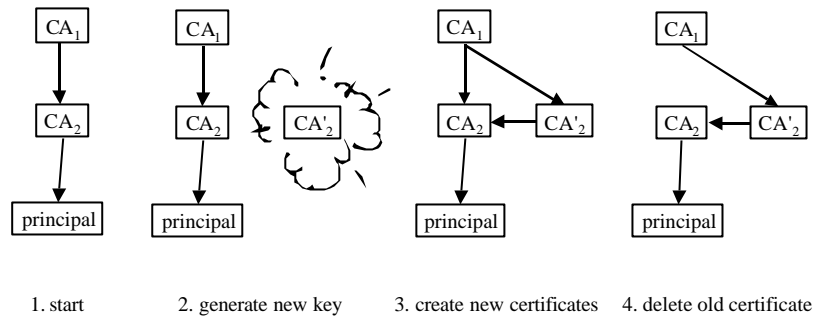


Figure 17 - First level CA key roll-over

Similarly, when a verifier's trust point changes its key, it cannot immediately provision its verifiers with the new key. So, it issues a certificate for its new key signed with its previous key. See Figure 18.

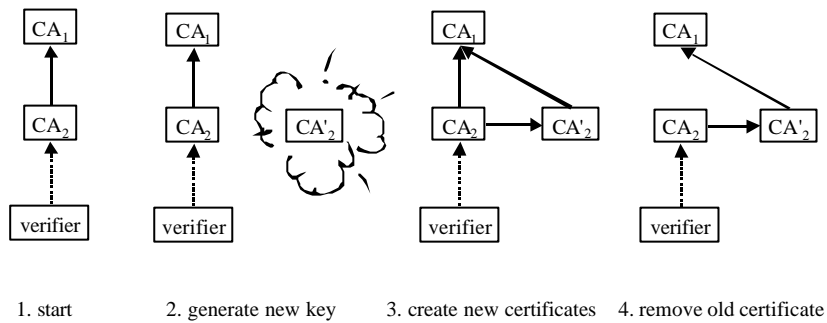


Figure 18 - First level CA key roll-over

4.6 Comparison summary

Comparison between the two mainstream models is summarized in Table 1, in which N is the number of roots or bridges under which the principal operates.

	Hierarchy	Bridge
Number of subordinate CAs at each level	N	1
Number of certificates per end-user	N	1
Policy convergence point	Principal	Spoke CA
Path development	Backwards	Forwards
Path validation	Following development	During development
Certificate publication	Subject	Issuer

Table 1 - Comparison summary of hierarchy and bridge

Finally, another consequence of the differing ways in which hierarchy and bridge trust models operate is that hierarchies may have shorter certification paths than bridges, leading to better performance.

4.7 Need to operate in more than one community

The foregoing comparison relies heavily on the behaviour of the models in the presence of multiple nodes. So, why would a principal have to operate subordinate

to more than one node? This is a requirement when principals have to authenticate to verifiers in different industry sectors e.g.:

Government

Financial services

Legal

Education

Supply chain

Whenever one CA's subscribers are required to authenticate to more than one community, there is a need to support multiple nodes.

4.8 Policy mapping

As we saw earlier, in the hierarchical model, only the end-entity has to operate in accordance with the policies of all the roots that certify it, as shown in Figure 19. This is commonly achieved by having separate cryptographic service providers (CSPs) for each hierarchy.

End-user policy mapping involves adapting end-user behaviour such as password requirements, obligations to report compromise, cryptographic algorithm requirements (CSPs), private key activation requirements (biometrics, etc.) to the policy defined by the root.

In the bridge model, the spoke CA has to operate a policy that is acceptable to more than one bridge.

A certificate policy comprises a set of policy elements, each of which can adopt one of a number of values. It should be an objective of policy writing to ensure that its elements are "orthogonal". This helps to facilitate mapping. When more than one element is directed at achieving the same objective there are many combinations that may be considered equivalent, and it becomes difficult to identify them.

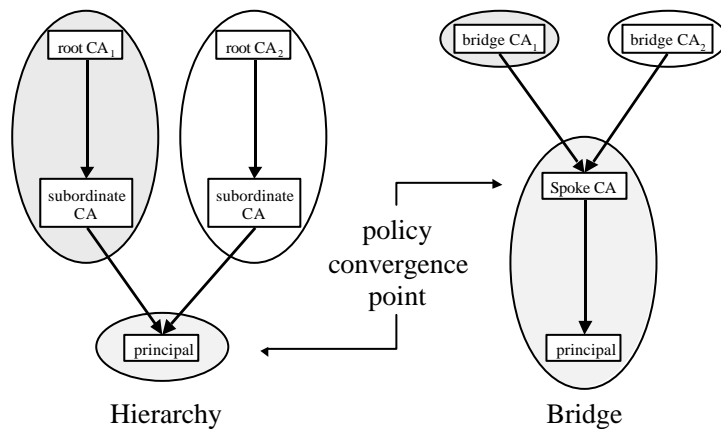


Figure 19 - Policy convergence

There are two primary documents that relate to policy mapping. These are the Certification Practice Statement (CPS) and the Certificate Policy (CP).

4.8.1 Certification practice statement

The certification practice statement is straight-forward. It is a record of the safeguards applied in each element of the PKI.

The elements of a CPS are not orthogonal, so they do not form a suitable basis for mapping.

Good guidance documents on the contents of a CPS are available. For instance, the IETF has published a framework for CPSs.

4.8.2 Certificate policy

There are two main types of certificate policy:

Usage policy; and

Assurance policy.

Usage policy defines the usage and community for which the certificates are suitable and approved. It lays out the warranties and obligations of the parties to one another. The parties must independently choose a set of safeguards for their own environment that limits their risk in light of their obligations under the policy to an acceptable level.

Assurance policies define the requirements for the safeguards in each of the party's operating environments. Then verifier application owners must evaluate the safeguards to determine whether they are adequate for their intended use.

When assurance-style policy is used, certificate validation merely answers the question, "*who* is the owner of the corresponding private key?" So, this must be followed by a process that maps that identity to eligibility in the context of the transaction. When usage policy is used, certificate validation in conformance with a particular certificate policy also partially answers the question of eligibility because it locates the principal in a defined community.

In order for two policies to be considered compatible, for purposes of policy mapping, the elements of each policy must be materially equivalent. This is easier to determine for usage policy than it is for assurance policy, unless the assurance policies are equivalent by design.

Usage policy should contain the following elements:

- Identification – The object identifier and user-friendly name of the policy.
- Community – The definition of the communities entitled to subscribe to and rely on the certificates.

- Applicability – The approved uses of the certificates, including limits of liability offered by the CA, and therefore, limits of reasonable reliance.
- Warranties – A statement of the warranties offered by the CA.
- Disclaimer – A statement to the effect that this policy statement defines the entire warranties offered by the CA, and that no other warranties, express or implied, are offered.
- Governing law – Identification of the law governing the operation of the CA.

The encoding of certificate policy in X.509 certificates is discussed later.

4.9 Hybrid architectures

Hybrid trust architectures combine the benefits of both mainstream models. See Figure 20. They provide the advantage of the shorter certification paths and more efficient path development offered by hierarchies, without having to operate multiple PKIs, as one normally would with a simple hierarchy.

Hybrid architectures require policy mapping between the policy of the local root and the policies of the various bridges. Then a single PKI in the local domain in which each principal has a single certificate and private key allows interoperation with many remote and independently-developed domains.

If the hierarchies weren't bridged, but were subordinate to roots, then there would have to be a separate local PKI for each of the roots.

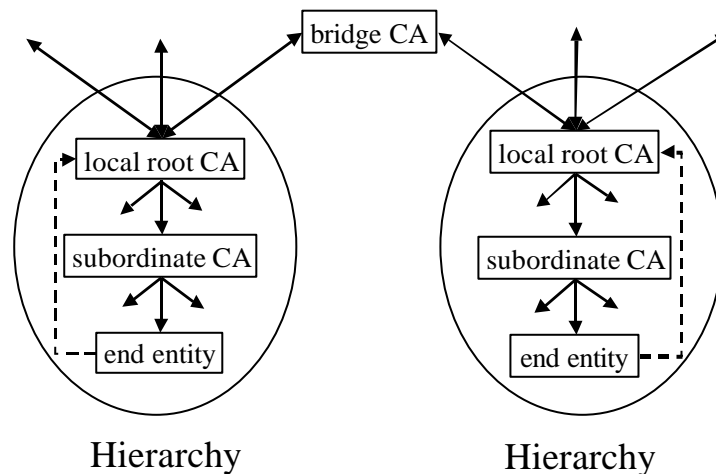


Figure 20 - Combination trust model - bridged hierarchies

The trust models in each of the local domain do not have to be the same, see Figure 21. This diagram shows a mesh and a hierarchy PKI interoperating through a bridge.

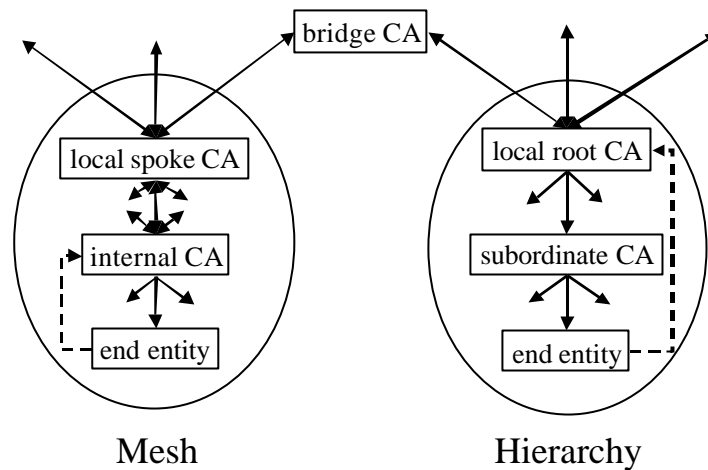


Figure 21 - Combination trust model - hierarchy and network

Path validation in combined architectures can be made more efficient if it can be recognized when the principal is operating in a local hierarchy. Then, it is optimal to develop the path backwards from the principal to the point at which there is more than one superior in the trust model, and then to develop the path forwards, validating as you go, from the verifier's trust point to the beginning of the path that resulted from the earlier backward development. Then the path validation can be completed.

5. Future trust models

All the preceding discussion related to trust models that are either in routine use or in experimental use today. In this section, we look at the ways in which these architectures may be extended to accommodate future requirements.

Earlier we saw how organizations can choose to subscribe to the bridges serving each of the communities of which they are members, as shown in Figure 22.

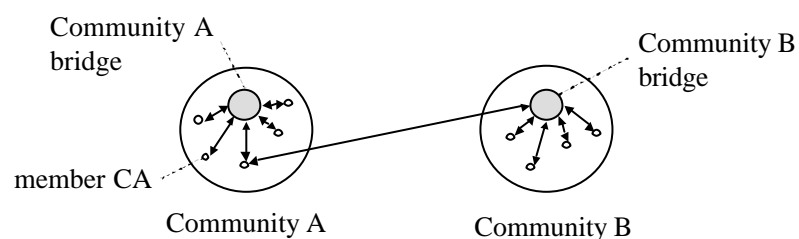


Figure 22 -Independent bridges

An alternative is that they operate within a single bridge which may “introduce” them to other bridges, as shown in Figure 23.

The utility of this approach is a business matter, not a technical one. It relies on an organization coming forward and offering the service of connecting its bridge to

other bridges. It remains to be seen which service providers can establish a business case for offering this service.

In the future, we may even see super-bridges that bridge communities by mutually cross-certifying with community bridge CAs that, in turn, directly serve communities of PKIs. But, it remains to be seen whether there is a business case for an organization to offer this service.

These possibilities are in the future. It is too early to tell what approaches will emerge into the mainstream.

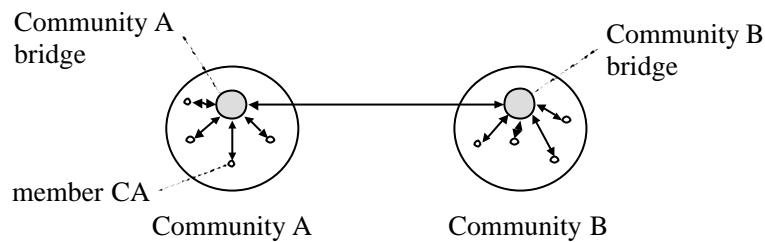


Figure 23 - Connected bridges

6. Unintended trust paths

A concern with the bridge model that is sometimes raised is that it opens up unintended trust paths, as shown in Figure 24. It is argued that if a spoke CA issues a certificate for a bridge CA and *it* issues a certificate for another bridge CA, and it issues a certificate for another spoke CA, etc., then pretty soon certification paths get unreasonably long and the assurance in the associated authentication is diluted to the point that it becomes worthless.

Fortunately, this is a misconception: there are control features in X.509 to prevent this from becoming a problem. These control features use the standard X.509 certificate extensions.

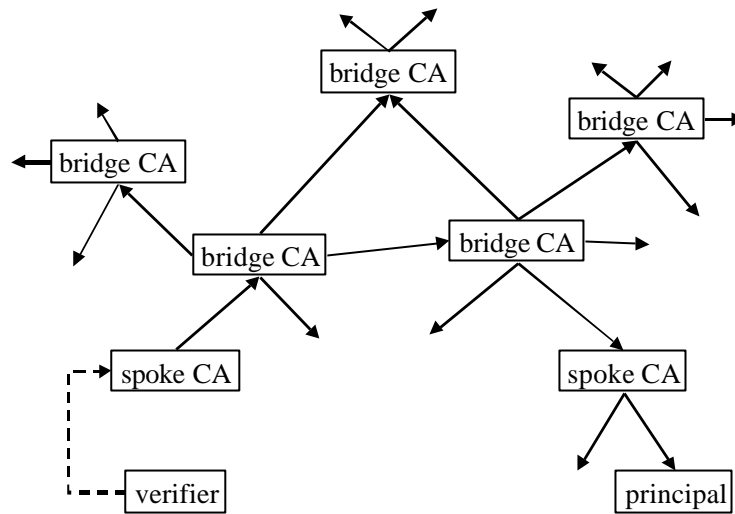


Figure 24 - Unintended trust paths

6.1 Certificate extensions

The X.509 certificate syntax standard specifies a method of adding information to certificates (using extensions) and a set of standard extensions. Certain of these standard extensions qualify the approved use of certification paths that include them. Of course, in order to be acceptable, several other conditions (such as validity, cryptographic integrity, revocation status, etc.) must also be verified.

These extensions allow the branching tree of certification paths to be pruned as the path is developed and to prevent unreasonable reliance on unqualified subscribers. Without these, it would be impractical to develop paths in a forward direction, in a richly interconnected trust model.

Extensions of relevance to this discussion are:

- Basic Constraints
- Certificate policies
- Policy mappings
- Policy constraints
- Inhibit any policy
- Name constraints

These are described in the remainder of this section.

6.2 Basic constraints

The basic constraints extension is primarily intended to distinguish between CA certificates and principal certificates, in order to prevent a principal from creating a certificate for another principal that would be acceptable to the verifier. The ASN.1 syntax of the basic constraints extension is:

```

BasicConstraintsSyntax ::= SEQUENCE {
    cA                BOOLEAN DEFAULT FALSE,
    pathLenConstraint INTEGER (0..MAX) OPTIONAL }

```

The “cA” field indicates that the subject of the certificate is a certification authority, and so the certification path does not have to terminate with this certificate. It must be set TRUE, unless the subject is an end-user. End-users (by definition) cannot issue certificates.

The “pathLenConstraint” field is intended to limit the number of subsequent certificates in the path. This is most useful in the cross-certificates issued *by* the bridge to a local hierarchy. Using this field requires the target domain to design its trust structure and to adhere to that design. This extension is limited in its usefulness because length is a poor indicator of a path’s acceptability.

6.2.1 Certificate policies

The certificate policies extension is primarily intended to ensure that certification paths are relied upon only for purposes for which they were issued and by the verifiers that are properly approved to rely on them. The ASN.1 syntax of the certificate policies extension is:

```

CertificatePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF
PolicyInformation

PolicyInformation ::= SEQUENCE {
    policyIdentifier  CertPolicyId,
    policyQualifiers SEQUENCE SIZE (1..MAX) OF
        PolicyQualifierInfo OPTIONAL }

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
    policyQualifierId  CERT-POLICY-QUALIFIER.&id
        ({SupportedPolicyQualifiers}),
    qualifier          CERT-POLICY-QUALIFIER.&Qualifier
        ({SupportedPolicyQualifiers}{@policyQualifierId})
        OPTIONAL }

```

A certificate policy is a plain-language document, but they are identified in this extension by an “object identifier”. An object identifier is a string of numbers, which, if properly administered, is globally unique. A certificate may contain none, one or more than one such identifier.

In a usage-style certificate policy, these values refer to the purposes and communities for which the certificate may be used. If an assurance-style policy is used, then they refer to the safeguards with which the certificates were issued. There is a standard special value, called “any policy”, which indicates that no limitations are placed by the issuer on the use of the certificate.

In the process of validating a certificate, the verifier can define the set of policies that are acceptable for its intended purpose. Then only certification paths made up of certificates that contain a common identifier and, furthermore, an identifier that is contained in the initial policy set, will be explored and validated.

The syntax also allows for *policy qualifiers*. The only qualifiers that are defined to date are a text string to be displayed to the verifier and a URL indicating how the plain-language text of the policy may be retrieved for display. Only qualifiers in the first certificate in the certification path are of interest to the verifier, because it is the responsibility of a CA when it issues a cross-certificate to ensure that the policy under which the subject CA issues certificates is consistent with its own.

This extension should be used in all certificates.

6.2.2 Policy mappings

The policy mappings extension is important when domains independently develop usage-style certificate policies that are materially equivalent but identified by different object identifiers. The ASN.1 syntax of the policy mappings extension is:

```
PolicyMappingsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
    issuerDomainPolicy CertPolicyId,
    subjectDomainPolicy CertPolicyId }
```

The operational authority for a CA can declare the policies equivalent by including the policy mappings extension in its cross-certificate. Then certificates issued under one policy can be used in a domain where the equivalent policy has a different identifier.

This extension may be used in certificates issued for and by the bridge CA.

6.2.3 Policy constraints

The policy constraints extension allows a CA to cause legacy certificates that don't contain a certificate policies extension to be rejected and to limit the number of policy mappings that occur in a certification path. The ASN.1 syntax of the policy constraints extension is:

```
PolicyConstraintsSyntax ::= SEQUENCE {
    requireExplicitPolicy [0] SkipCerts OPTIONAL,
    inhibitPolicyMapping [1] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)
```

These restrictions can be set to take effect further down the certification path.

One should expect both fields to be present in the certificate issued for a bridge by a spoke CA. In this case, explicit policies should be required in all subsequent certificates and policy mapping should be inhibited after the certificate issued by the bridge for the target spoke CA. While a single policy mapping step in a certification path will likely be acceptable in many applications, more than one is not likely to be.

6.2.5 Inhibit any policy

The inhibit any policy extension allows a CA to prevent those that rely on its public key from accepting certificates further down the path that are issued with no specified approved use. The ASN.1 syntax of the inhibit any policy extension is:

```
InhibitAnyPolicySyntax ::= SkipCerts
```

```
SkipCerts ::= INTEGER (0..MAX)
```

6.2.5 Name constraints

The name constraints extension is intended to help verifiers prune branching paths early in the process on the basis of names. The ASN.1 syntax of the name constraints extension is:

```
NameConstraintsSyntax ::= SEQUENCE {
    permittedSubtrees  [0]  GeneralSubtrees OPTIONAL,
    excludedSubtrees  [1]  GeneralSubtrees OPTIONAL }

GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

GeneralSubtree ::= SEQUENCE {
    base                GeneralName,
    minimum             [0]  BaseDistance DEFAULT 0,
    maximum             [1]  BaseDistance OPTIONAL }

BaseDistance ::= INTEGER (0..MAX)
```

The name constraints extension controls the name-spaces and their subtrees in which the subject CA can issue acceptable certificates.

It should be included in cross-certificates issued for bridge CAs by spoke CAs to indicate the subject domains of the bridge that are acceptable to the verifier community served by the spoke CA. Since bridge CAs may issue certificates to many domains which do not need to be trusted by the spoke's verifiers.

While the X.509 standard does not require name subordination, the name constraints extension works most effectively when name subordination is in effect.

6.3 Summary

Table 1 indicates where these extensions should be used.

	Certificate issued for the bridge	Certificate issued by the bridge
Basic constraints	✓	✓
Certificate policies	✓	✓
Policy mappings	✓	✓
Policy constraints	requireExplicitPolicy SkipCerts = 0 inhibitPolicyMapping SkipCerts = 1	
Inhibit any policy	skip certs = 0	

Name constraints	✓	
------------------	---	--

Table 2 - Extensions

7. Conclusions

The two mainstream trust models are the hierarchy and the bridge.

In a multi-policy, multi-node environment, the hierarchical approach leads to duplication of subordinate CAs and principals' private keys. The bridge CA approach allows a single spoke CA and single principal private key to be re-used in more than one bridge.

In the case of the bridge model, the style of certificate policy must be carefully chosen to facilitate mapping at the spoke CA level.