Chapter 4

Merging Hierarchical PKIs
without using Cross-Certification

4.1 Introduction

The Public Key Infrastructure technology is very important to support secure global electronic commerce and digital communications on networks. With the development of E-commerce, many enterprises have built their own Public Key Infrastructures (PKIs) to support the various web applications[139]. In electronic commerce, different PKIs need to be interoperated. Especially, they need to be merged for merger and acquisition of companies. Once the enterprise changes its collaborators, the multiple PKIs deployed by the enterprise and its new collaborators must be interoperated. The interoperability among these PKIs is temporary, which will dynamically change with the market requirements. Thus, the merging process needs to be low-cost\(^1\), easily constructed and flexible. Lot of research work has been carried out to provide solutions for merging PKIs. However, no solution is found to be suitable for faster certificate path discovery and cost effectiveness[5][6]. Considering this fact, a general method to unify Hierarchical PKIs is proposed in

\(^1\)Some part of this chapter appears in the Proceedings of International Conference on Open Source Computing, INCOSC-08, pp: 109-113
this research, that takes a different approach from cross-certification technique. The method is to unify the multiple CAs without using cross-certification. By using this method, the trust model with an efficient path processing can be built in comparison with the traditional merging methods with cross-certification. A certificate verifier should construct and validate the certification path. If there are cross-certifications, the path construction process is very complex. In such a case, path construction may involve the use of graph-theoretic path finding algorithm[140]. Moreover, the difference in security policies is a serious issue in PKI interoperation, since each company has its own security policies.

4.1.1 Cross-Certification

Cross-certification is a useful mechanism for binding together previously unrelated CAs so that secure communications between their respective subject communities can be enabled. The difference between certification and cross-certification is that, in the latter both the subject and the issuer of the resulting cross-certificate are CAs, but in the former, the subject is an end entity. When the distinction is important, the following terminology from RFC2510 can be used: If the two CAs belong to the same domain (for example, within an organization’s CA hierarchy, where a CA at one level is certifying a CA at the next level below), the process is referred to as intradomain cross-certification. If the two CAs belong to different domains (for example, when a CA in one company is certifying a CA in another company), the process is referred to as interdomain cross-certification.

Cross-certification can occur in one or two directions. If CA1 cross-certifies (that is, signs the identity and public key of) CA2 without CA2 cross-certifying CA1, it is called unilateral cross-certification. This results in a single cross-certificate. Alternatively, CA1 and CA2 can cross-certify each other. This phenomenon is called mutual cross-certification that results in two distinct cross-certificates and can be a more common occurrence, for example, between companies.
wanting to enable secure communications between their respective employees.

From the perspective of CA1, a cross-certificate issued for it (that is, with CA1 as the subject and some other CA as the issuer) is called a forward cross-certificate; one issued by it is called a reverse cross-certificate. The term forward can also be referred to as “issued to this CA” and reverse can be referred to as “issued by this CA”. If an X.500 Directory is used as the certificate repository, the appropriate “issued to this CA” and “issued by this CA” cross-certificates may be stored in a cross-certificate pair structure in the directory entry of each relevant CA. This structure can be helpful in facilitating certificate path construction. The mechanism of cross-certification can be used to extend trust between (or among) distinct relying party communities. In particular, cross-certification between two CAs is one way that a given CA can recognize that another CA is authorized to issue certificates in (typically a specified part of) a name space.

Thus, cross-certification allows otherwise disparate PKI domains to easily establish an interoperability path. An undesirable alternative would be to exchange root CA keys and to populate every end entity’s software or hardware tokens with the root CA key of the external domain. For example, assume that Alice has been certified by CA1 and holds a trusted copy of CA1’s public key and that Bob has been certified by CA2 and holds a trusted copy of CA2’s public key. Initially, Alice may trust only entities whose certificates have been signed by CA1 because these are the certificates she can verify. She cannot verify Bob’s certificate (because she does not hold a trusted copy of CA2’s public key); similarly, Bob cannot verify Alice’s certificate. After CA1 and CA2 have cross-certified, however, Alice’s trust can be extended to the subject community of CA2, including Bob, because she can verify CA2’s certificate, using her trusted copy of CA1’s public key, and then verify Bob’s certificate, using her trusted copy of CA2’s public key. CA1 may cross-certify CA2 but limit in some desired way the subject community of CA2 that the relying party community under CA1 will trust. Trust can be extended, on an organization-wide basis within the domain of CA1, only to certain individuals,
only to certain groups, only for specific purposes, and so on, in the domain of CA2. This kind of organizational control over trust extension, centrally determined by the CA1 administrator, is difficult or impossible to achieve with the Web model or with the user-centric trust model. It is also irrelevant in a strict CA hierarchy model because there is only one domain; there is no other domain to which trust can be extended.

If PKIs are to be implemented in any widespread fashion, cross-certification is an important factor to consider. Instead of using a single global CA, cross-certification allows end entities to use a CA based on their particular needs. It is possible that end entities under one CA may need to authenticate end entities under another CA; however, cross-certification supports this relatively straightforward process. Essentially, what occurs in a cross-certification is that one CA certifies another. As with the generation of an end entity’s digital certificate, a CA performs various due diligence tests on the CA it will cross certify. These tests are taken in accordance with the published Certificate Policy and Certificate Practice Statement of the certifying CA.

When a cross-certificate is issued, it extends the trust relationship of a CA. A relying entity, for example, may desire to validate the public key certificate of an end entity whose signing CA’s public key it is not aware of. Assuming that the relying entity trusts its own CA, when it sees a cross-certificate signed by that CA, it will then also trust that other CA, and subsequent certificates signed by it. The net effect of cross-certification is to allow many PKI deployments to be both extensible and scalable. As a result, any number of PKIs can participate in the cross-certification. In this way, this solution provides a means to achieve interoperability between PKIs.
4.2 Merging Hierarchical PKIs

In order to mitigate the certification path processing, a merging method of CAs without using cross-certification is designed and tested. The procedure is to set up a new PKI, without constructing new CAs and issue new certificates since it increases cost of merging. Also, if new CAs are introduced, the path verification time also increases. The aim of merging CAs is smooth migration using existing PKIs. Merging process is carried out to merge multiple CAs using CAs private key update operations[1].

In the proposed method, the term ‘Host’ PKI is used to refer to the PKI with which other PKIs will merge and the PKI which will merge with the Host PKI is referred to as ‘Guest’ PKI. The term 'entity' refers to either an end-entity or an intermediate CA. The merging of PKIs is possible only if the compatibility score of Certificate Policies of the PKIs to be merged are within the acceptance level which is explained in chapter 6.

4.2.1 Merging process

Let $PKI_1, PKI_2, \ldots, PKI_n$ represent the model set and they are to be merged into one trust model. Each $PKI_i$, $1 \leq i \leq n$, is the strict hierarchical model, respectively. $RCA_i$ denotes the most trusted CA (Root CA) in $PKI_i$. Because the trust model after merging CAs is a strict hierarchical model, the new Root CA after integration is determined. There are two methods of selecting new Root CA: (1) selecting from $RCA_i$, $1 \leq i \leq n$; and (2) constructing a new Root CA. Using the latter method, it is expensive to construct the new CA because Root CA is the most trusted authority, and therefore should be the most protected. All trust within the system hinges on the integrity of the private key and public key certificate belonging to Root CA. In order to minimize the cost incurred, the former method is used. $RCA_{new}$ denotes new Root CA after integration.
The merging process is described as follows:

First, the CA which will be the new Root CA ($RCA_{\text{new}}$) after the merger is decided. In the proposed method, Root CA of the 'Host' PKI is considered as $RCA_{\text{new}}$. If the compatibility score of certificate policies of the PKIs to be merged are within the acceptance level, then, $RCA_{\text{new}}$ announces its public key to entities of all the 'Guest' PKIs and issues certificates for their First level entities in the hierarchy. However, if the certificate policies of the PKIs to be merged are not within the acceptance level, merging is not possible.

$RCA_{\text{new}}$ just issues the certificate to all entities under 'Guest' PKIs, however, these entities cannot trust $RCA_{\text{new}}$. In other words, entities who do not trust $RCA_{\text{new}}$ cannot verify the certificate issued by $RCA_{\text{new}}$. In order to solve this problem, $RCA_i$, issues a certificate to $RCA_{\text{new}}$. This approach is the same as CA’s private key update operations[1].

Figure 4.1 depicts the overall process diagram of the proposed merging process.

In case of updating CA key pair, new CA key pairs are generated. In order to update CAs certificate smoothly, three kinds of certificates described as follows are utilized.

**Old with Old**

The certificate containing the old CA public key signed with the old CA private key.

**New with Old**

The certificate containing the new CA public key signed with the old CA private key.

**New with New**

The certificate containing the new CA public key signed with the new CA private key.

The Old with Old certificate is held by all entities who trust the CA. The new CA public key is certified by the old CA private key (New with Old certificate).
By verifying New with Old certificate, entities will be able to trust the New with New certificate.

In the method, the old CA keys mean $RCA_i$’s keys and the new CA keys mean $RCA_{new}$’s keys.

The entire merging process can be summarized as follows:

- The entities who trust $RCA_i$ have obtained the certificate containing the $RCA_i$’s public key signed with the $RCA_i$’s private key (Old with Old certificate).
- $RCA_{new}$ creates the new certificate signed by $RCA_{new}$’s private key (New
with New certificate) and distributes to all certificate users.

- $RCA_i$ issues the new certificate containing $RCA_{new}$’s public key signed by $RCA_i$’s private key (New with Old certificate). Entities who trust $RCA_i$ can trust $RCA_{new}$ by verifying this certificate.

4.2.2 The Algorithm

The following fundamental sets are defined:

- $\text{ENT}$ represents the set of all PKI entities. (For example, End-entities and CAs)

- $\text{CA}$ represents the set of all CAs and the subset of $\text{ENT}$

It is assumed that a certificate is the tuple $(x, y)$ where $x \in \text{CA}$, $y \in \text{ENT}$. In detail, $x$ is the issuer of certificate and $y$ is the owner of the certificate.

The following procedures are defined in the proposed algorithm for merging:

- $\text{Get\_Firstlevel\_Entities}(X)$: The subordinate CAs and end-entities in the next level of $X$ are determined

- $\text{Entities}(X)$: All the entities under $X$ are determined

- $\text{Announce\_Public\_Key}(X_1, \text{entities}(X_2))$: $X_1$ announces its public key to all the entities under $X_2$

- $\text{Issue\_Cert}(X_1, X_2)$: $X_1$ issues certificate to $X_2$

Steps

1. Let $n$ be the number of participating PKIs for merging

2. Let $RCA_1, RCA_2, \ldots, RCA_n$ be the Root CAs of these PKIs

3. Determine the New Root CA:

   $RCA_{new} \leftarrow RCA \text{ of the Host PKI}$
4. Determine the First level entities of Guest PKIs:

\[ \text{Entities}_{\text{first level}} \leftarrow \text{null} \]

\[ \forall \text{RCA}_i, \text{ where } 1 \leq i < n \quad /\text{except the } \text{RCA}_{\text{new}} \]

\[ \text{Entities}_{\text{first level}} \leftarrow \text{Entities}_{\text{first level}} \cup \text{Get}\_\text{Firstlevel}\_\text{Entities} (\text{RCA}_i) \]

5. Announce the public key of \( \text{RCA}_{\text{new}} \) for entities of Guest PKIs:

\[ \forall \text{RCA}_i, \text{ where } 1 \leq i < n \quad /\text{except the } \text{RCA}_{\text{new}} \]

\[ \text{Announce}\_\text{Public}\_\text{Key} (\text{RCA}_{\text{new}}, \text{Entities} (\text{RCA}_i)) \]

6. RCAs of Guest PKIs certify the \( \text{RCA}_{\text{new}} \):

\[ \forall \text{RCA}_i, \text{ where } 1 \leq i < n \quad /\text{except the } \text{RCA}_{\text{new}} \]

\[ \text{Issue}\_\text{Cert} (\text{RCA}_i, \text{RCA}_{\text{new}}) \]

7. \( \text{RCA}_{\text{new}} \) issues certificates to First level entities of the Guest PKIs:

\[ \forall \text{RCA}_i, \text{ where } 1 \leq i < n \quad /\text{except the } \text{RCA}_{\text{new}} \]

\[ \text{Issue}\_\text{Cert} (\text{RCA}_{\text{new}}, \text{Entities}_{\text{first level}}) \]

4.3 Implementation and Experimental Results

4.3.1 OpenSSL

OpenSSL is an open source implementation of the SSL and TLS protocols. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available. SSL is an acronym that stands for Secure Sockets Layer. It is the standard behind secure communication on the Internet, integrating data cryptography into the protocol.

OpenSSL is more than just SSL. It is capable of creating message digests, encryption and decryption of files, generation of digital certificates, digital signatures, and random numbers. OpenSSL is more than just the API, it is also a command-line tool. The command-line tool can do the same things as the API, but goes a
step further, allowing the ability to test SSL servers and clients. OpenSSL is used in the implementation of the proposed algorithm. It is used to generate the certificate and keys, signing certificates, creating Hierarchical PKIs etc. It is also used to store the digital certificates and retrieve particular field values from it. Private keys are also stored and retrieved using SSL. The private key of a CA is required to sign the certificate for the next level CA or the end user when the request is being made by the latter.

### 4.3.2 Certificate Generation

To generate a certificate, the CA performs the following steps. Note that although an RA may be used, it is excluded from the process here for brevity.

- Acquire a public key from the end entity.
- Verify the identity of the end entity.
- Determine the attributes needed for this certificate, if any.
- Format the certificate.
- Digitally sign the certificate data.

**Acquiring the Public Key**

Depending on the value proposition and business risks associated with the issuance of a public key certificate, the CA may take basic measures when obtaining the required identification and certificate information. For example, the CA may allow it to be sent electronically over the Internet, or the CA could use more sophisticated means such as mandating out of band manual methods (e.g., using bonded couriers). The type and integrity of the credentials requested by the CA during the enrollment process also depends on the intentions of the CA.
Verify the Identity of the End Entity

Again, depending on the business model in place and the amount of reliance on the public key certificates, the CA may take simple measures to authenticate the end entity. For example, they could simply take the end entity’s word that they are who they say they are, or the CA could institute more stringent measures, such as a detailed due diligence process as the business proposition merits. In any case, the most important factor in the binding process is to ensure that an entity’s identity is verified unambiguously.

Formatting the Certificate

Before the certificate is signed by the CA, all data to be placed into the certificate is collected and formatted. The specific data content and format of a public key certificate can vary depending on the needs of the PKI.

Signing the Certificate

Finally, the certificate is digitally signed using the private key of the CA used for signing certificates. Once signed it can be distributed and/or published. The Figure 4.2 represents the structure of a certificate.

4.3.3 Creation of a Hierarchical PKI

The Hierarchical PKI is created using Java with OpenSSL tool. The entire process of creating Hierarchical PKI consists of the following steps:

- Creation of root certificate
- Generation of key pair
- Creation of intermediate certificate authority
- Creation of end user certificate
Creation of the root certificate

In cryptography and computer security, a root certificate is either an unsigned public key certificate or a self-signed certificate that identifies the Root Certificate Authority (RCA). A root certificate is part of a public key infrastructure scheme. The most common commercial variety is based on the ITU-T X.509 standard, which normally includes a digital signature from a certificate authority (CA).

Digital certificates are verified using a chain of trust. The trust anchor for the digital certificate is the Root Certificate Authority. Since it is a self-signed certificate, the issuer and the subject name are the same.

Generation of Key pair

RSA algorithm has been used to generate key pairs for the certificates having key size of 1024 bits. The key pair generated using this algorithm is used to generate the certificates. It is the first algorithm known to be suitable for signing as well as
encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure, given sufficiently long keys and the use of up-to-date implementations. RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

**Creation of intermediate certificate authority**

An intermediate certificate is the certificate, or certificates, that are issued to the intermediate CAs by their superiors.

The intermediate certificate, or certificates, completes the chain to a root certificate trusted by the browser. Key pair for intermediate certificate authority is generated in the same way as for the root certificate authority. However the issuer and the subject name are different in intermediate CA as compared to root CA. The issuer name is the name of the certificate authority issuing the certificate (which may be the root CA) and the subject name is the name of the current intermediate CA being created. The private key of the parent CA is used to sign the certificate issued to the current CA. This intermediate CA can provide certificates to other CAs or end users signed by its own private key and which will be trusted by the whole infrastructure.

**Creation of end user certificate**

End user can request the intermediate or the root CA for the certificate which will be signed by the private key of the issuer CA. Key pair for end user certificate is generated in the same way as for the root certificate authority and intermediate CA. Figure 4.3 is the overall workflow diagram for creating Hierarchical PKIs.
4.3.4 Implementation of the proposed algorithm

A prototype for merging two or more Hierarchical PKIs based on the proposed method, is developed using Java with OpenSSL tool. As shown in Figure 4.4, PKI1, PKI2 and PKI3 are given as the inputs for the merging process. It is assumed that PKI2 is the 'Host' PKI and PKI1 and PKI3 are the 'Guest' PKIs. The result is the merging of these PKIs with RCA2 as the new Root CA ($RCA_{new}$). Root CAs of Guest PKIs, i.e., RCA1 and RCA3 certify RCA2 ($RCA_{new}$). After merging, RCA2 is the new Root CA ($RCA_{new}$).
4.3.4.1 Certificate Path Processing

As in traditional merging methods using cross certifications, the certification path processing is inefficient, since the path length becomes long and bidirectional. In the mesh model, the path length is dependent upon the number of PKI models. In the bridge CA model, the path length is independent of the number of PKI models. On the other hand, a strict hierarchical model is constructed by performing the proposed merging processes, so certification path processing is more efficient than the previous general methods.

Figure 4.4: Merging Hierarchical PKIs without using cross-certification
Table 4.1: Path verification time using cross-certification and the proposed method

<table>
<thead>
<tr>
<th>Depth of the tree</th>
<th>Time taken for Path verification in Merged PKIs with cross-certification (in milliseconds)</th>
<th>Time taken for Path verification in the proposed method (in milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>109</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>172</td>
<td>125</td>
</tr>
<tr>
<td>3</td>
<td>188</td>
<td>157</td>
</tr>
<tr>
<td>4</td>
<td>235</td>
<td>203</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>234</td>
</tr>
<tr>
<td>6</td>
<td>300</td>
<td>276</td>
</tr>
<tr>
<td>7</td>
<td>320</td>
<td>294</td>
</tr>
<tr>
<td>8</td>
<td>370</td>
<td>349</td>
</tr>
<tr>
<td>9</td>
<td>390</td>
<td>369</td>
</tr>
<tr>
<td>10</td>
<td>440</td>
<td>404</td>
</tr>
</tbody>
</table>

Table 4.1 shows the result of path verification time using the normal merging method, i.e., merging with cross-certification at the root and the proposed method by considering different depths of Hierarchical PKI trees. The graph in Figure 4.5 shows the comparison between the normal method (i.e., merging hierarchical PKIs using cross-certification at the root) and the proposed method for verification of end user certificates. From the graph it is seen that time taken for certificate path verification by merging PKIs using the proposed method is less than that of the normal method. As the depth of the Hierarchical PKI tree increases, the time taken for certificate path verification also increases.
4.3.4.2 Operational Cost

There are some factors to evaluate the operational cost. These factors are the employment cost (personal expense), the software and hardware that are needed to employ the issuance of the certificate and manage the CAs’ private keys. In a bridge CA model, the bridge CA should be constructed. While in the proposed method, the number of CAs is not increased after merging processes, so the employment cost of CAs is not affected. In exceptional case, the cost of protecting CAs’ private keys possibly becomes large. $RCA_{new}$’s private key must be stored very carefully. If $RCA_{new}$’s private key is compromised, all users are affected, since $RCA_{new}$ is the trust anchor for all users after merging.

4.3.4.3 Issuance of new certificate

In the mesh model, $n(n - 1)$ cross-certificates are required, where $n$ is the number of CAs. In the bridge CA model, the number of cross-certificates is $2n$. But in the proposed model, New with New and New with Old certificates need to be issued.
The total number of the new certificates to be issued is \( n \). Additionally, \( RCA_{\text{new}} \) issues the new certificate to entities who have been issued by the \( RCA_i \), \( 1 \leq i \leq n \). In order to trust \( RCA_{\text{new}} \), end users have to verify the New with Old and New with New certificates. The time requirement for issuing new certificates is \( O(k) \) where \( k=\text{total number of entities in the first level of all the ‘Guest’ PKIs and the number of RCAs} \). If there are \( n \) PKIs to be merged, in the proposed method, no cross-certification is required. However, Bridge and Mesh PKIs require \( O(n) \) and \( O(n^2) \) run time requirement respectively for cross-certifications. If the population of end entities becomes large, performing these processes will require much time and effort. Thus cost may increase.

### 4.4 Summary

A strict hierarchical model is constructed by performing this merging process, so certification path processing is more efficient than other methods. Certificate path length can be reduced which in turn reduces verification time. If the compatibility score of the certificate policies of the PKIs to be merged is within the limit, the PKIs can be merged.

In the next chapter, one more method to merge Hierarchical PKIs is explained. This method can be used during acquisition of companies. This method is slightly different from the method explained in this chapter.