WHITE PAPER

Understanding Certification Path Construction

September 2002

Certification path construction is a complex process and is subject to some degree of trial and error. The certification path construction process has not been standardized, and there is very little published information available to help implementers and product evaluators understand the complexities involved. The purpose of this white paper is to clarify issues associated with the certification path construction process and to make recommendations where appropriate.

Acknowledgements

This White Paper is a deliverable from the PKI Forum's Technical Group (TWG). Several member organizations and individuals have contributed by providing content, editorial assistance and editorial reviews.

Author:

Steve Lloyd
PKI Forum

Contributors:

Andrew Nash
Russ Housley
John Linn
Magnus Nystrom
RSA Security

Helen Mullenger
Baltimore Technologies

Jeff Stapleton
KPMG LLP

Contents

Introduction

Public Key Infrastructure (PKI) supports a number of security-related services, including data confidentiality, data integrity, and end-entity authentication. Fundamentally, these services are based on the proper use of public/private key pairs. The public component of this key pair is issued in the form of a public key certificate and, in association with the appropriate algorithm(s), it may be used to verify a digital signature, encrypt data, or both[1].

Before a certificate can be used, it must be validated. In order to validate such a certificate, a chain of certificates or a *certification path* between the certificate and an established point of trust must be established, and every certificate within that path must be checked. This process is referred to as *certification path processing*[2].

In general, certification path processing consists of two phases: 1) *path construction* and 2) *path validation* described as follows:

1) *Path construction* involves "building" one or more candidate certification paths. Note that we use "candidate" here to indicate that although the certificates may chain together properly, the path itself may not be valid for other reasons such as path length, name, or certificate policy constraints/restrictions.

2) *Path validation* includes making sure that each certificate in the path is within its established validity period, has not been revoked, has integrity, et cetera; and any constraints levied on part or all of the certification path are honored (e.g., path length constraints, name constraints, policy constraints). However, some aspects that might be associated with path validation are sometimes taken into consideration during the path construction process in order to maximize the chances of finding an acceptable certification path sooner rather than later. We will revisit these concepts later in this paper.

[1] It should be recognized that there are several reasons that separate key pairs should be used for digital signature and confidentiality, including differing requirements associated with key backup/recovery and long-term handling of keying material, and the ability to use different algorithms for each (e.g., DSA could be used for the digital signature and RSA could be used for symmetric key exchange).

[2] This is sometimes called "certificate path processing." We use "certification path processing" in order to maintain consistency with the terminology found in X.509.

PKI forum

The 4th Edition of X.509 [Ref1] and the Internet Certificate and Certificate Revocation List Profile as defined in RFC3280 [Ref2] provide the most recent implementation guidelines for certification path validation. However, both sources are silent when it comes to the certification path construction process.[3] In fact, there is very little published literature that addresses the certification path construction issue. Notable exceptions include [Ref3] and [Ref4].

Purpose, Scope and Assumptions

The purpose of this paper is to clarify terminology and review issues associated with the certification path construction process and to make recommendations where appropriate.

The path construction process can be complex and subject to some degree of trial and error, and there is very little existing literature that discusses issues associated with the certification path construction process. Therefore, the purpose of this paper is to clarify terminology and review issues associated with the certification path construction process and to make recommendations where appropriate. However, this paper is not meant to dictate how vendors MUST perform path construction, nor are we attempting to describe a complete path construction algorithm. Vendors are free to implement their own path construction logic as they see fit. Nonetheless, this paper provides useful information that should be taken into consideration when evaluating or implementing certification path construction algorithms.

Although this paper discusses issues related to certification path validation, it is not the purpose of this paper to address certification path validation per se. The 4th Edition of X.509 and RFC3280 should be consulted for additional information regarding the certification path validation process.

This paper focuses on Version 3 public key certificates as defined in the 4th Edition of the X.509 Recommendation [Ref1]. We recognize that other forms of certificates exist, including the Version 1 and Version 2 public key certificates defined in previous Editions of X.509. However, we concentrate on Version 3 public key certificates since they are the most common form of certificate found in most enterprise and government deployments, and many of the extensions that are only supported with Version 3 are essential in order to control business relationships within and across PKI domains. This paper discusses certain Version 3 certificate extensions that can be used to help facilitate the certification path construction process. It should be noted that any discussion associated with these extensions is not applicable in the context of Version 1 or Version 2 public key certificates.

This paper focuses on Version 3 public key certificates as defined in the 4th Edition of the X.509 Recommendation [Ref1]... We concentrate on Version 3 public key certificates since they are the most common form of certificate found in most enterprise and government deployments, and many of the extensions that are only supported with Version 3 are essential in order to control business relationships within and across PKI domains.

It is recognized that path construction software can be implemented locally (i.e., coupled with the client system), remotely (i.e., delegated to an external trusted 3rd party), or a combination of both. However, where path construction is performed is irrelevant for the purposes of this paper.

Basic Concepts

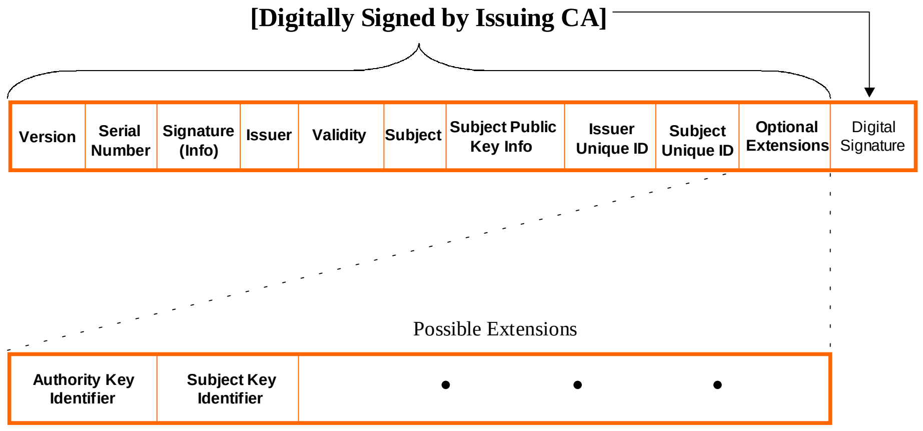
The basic structure of the Version 3 public key certificate is illustrated in Figure 1. Certificates are issued by Certification Authorities (CAs) to other CAs or to end-entities (e.g., end-users, devices, Web servers, processes). CAs may also "self-issue" certificates to themselves (this is discussed in more detail later).

Certificates issued to CAs are known as CA certificates, and certificates issued to end-entities are referred to as end-entity certificates. The Basic Constraints certificate extension is used to distinguish between CA certificates and end-entity certificates.

The 4th Edition of X.509 defines a relying party as "a user or agent that relies on the data in a certificate in making decisions." Stated another way, a relying party "uses" end-entity certificates for some express purpose. For example, a relying party may authenticate a message originator and verify the integrity of the message by "using" the message originator's corresponding certificate to verify a digital signature associated with the message. A trust anchor is a CA certificate (or more precisely, the public verification key of a CA) used by a relying party as the starting point for path validation (which may or may not be the same starting point for path construction as discussed further below). A relying party may have one or more trust anchors, and these trust anchors can be derived from a number of sources. For example, a trust anchor may be the public key of a root CA or it may be the public key of the CA that issues one or more certificates directly to the relying party.

[3] This is simply a statement of fact and is not intended to be a criticism. It is generally agreed within the industry that path construction algorithms should not be subject to standardization. The purpose of this paper is to shed light on path construction issues that are not immediately obvious from the existing standards.

Figure 1: X.509 Version 3 Public Key Certificate



A certificate that one CA issues to another CA is referred to as a *cross-certificate.* A superior CA may issue a cross-certificate to a subordinate CA as commonly found in hierarchical trust models. This is referred to as *unidirectional* or *unilateral cross-certification*. When CAs issue certificates to each other it is known as *mutual* or *bilateral cross-certification.* This is commonly found in distributed trust models. We note that there are many examples in the industry where the term cross-certification is meant to denote the bilateral case only. However, from a technical perspective cross-certification can also be unilateral. This is reflected in the 4th Edition of X.509 where a cross-certificate is defined as follows:

> *Cross certificate - This is a certificate where the issuer and the subject are different CAs. CAs issue certificates to other CAs either as a mechanism to authorize the subject CA's existence (e.g. in a strict hierarchy) or to recognize the existence of the subject CA (e.g. in a distributed trust model). The cross-certificate structure is used for both of these.*

A certificate that one CA issues to another CA is referred to as a cross-certificate. Cross-certification can be unilateral or bilateral.

Certification path construction involves discovery of a "chain of certificates" between the end-entity certificate and a recognized trust anchor. Certification paths can be constructed in the forward direction (i.e., from the end-entity certificate to a recognized trust anchor) or they can be constructed in the reverse direction (i.e., from a recognized trust anchor to the end-entity certificate). Which of these methods is best has been the source of some debate over the past few years. We assert that forward direction path construction is best suited for hierarchical trust models and reverse direction path construction is best suited for distributed trust models, and this paper will demonstrate that a robust path construction algorithm must be capable of building paths in both directions. In fact, it may be appropriate to build portions of a certification path using one method and other portions of the certification path using the other.

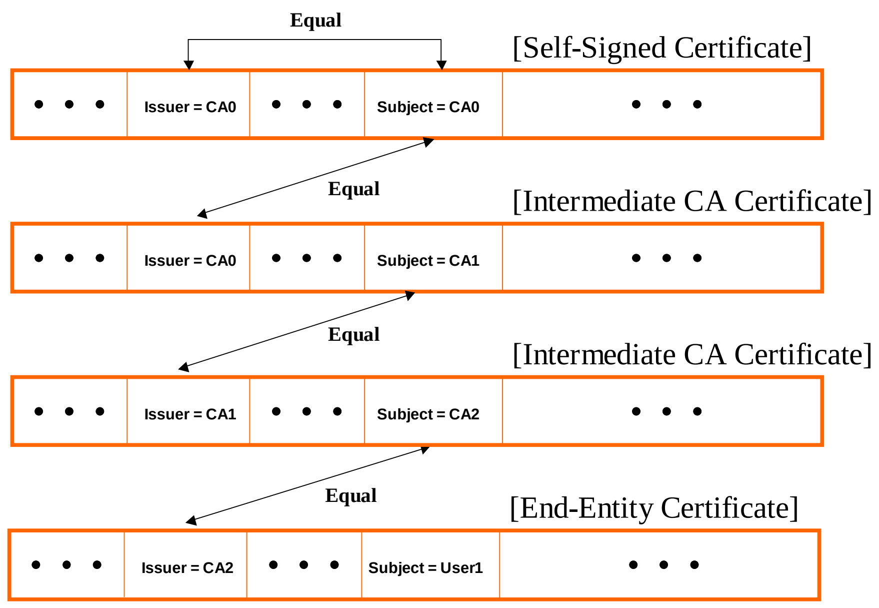
We assert that forward direction path construction is best suited for hierarchical trust models and reverse direction path construction is best suited for distributed trust models, and this paper will demonstrate that a robust path construction algorithm must be capable of building paths in both directions.

Before we begin to look at the issues surrounding the certification path construction process, let's first explore some fundamentals behind the notion of "certificate chaining".

Name Chaining

At the most basic level, a candidate certification path must "name chain" between the recognized trust anchor and the target certificate (i.e., the end-entity certificate). Working from the trust anchor to the target certificate, this means that the Subject Name in one certificate must be the Issuer Name in the next certificate in the path, and so on. Figure 2 helps to illustrate this concept. In this example, the path begins with a self-signed certificate that contains the public key of the trust anchor. The path ends with the end-entity certificate. All other certificates within the path are referred to as intermediate CA certificates. Note that every certificate in the chain except for the last one is a CA certificate.

Figure 2: Name Chaining Illustration



Name chaining alone may not be sufficient to determine if the certification path is a legitimate candidate that should be submitted to the certification path validation process.

Although we will discuss this in more detail later, let's quickly review how a certification path can be constructed using name chaining. (In order to keep things simple, it is assumed that the Issuer DN's match the corresponding directory entries associated with each CA.) When building certification paths in the forward direction, we can use the Issuer DN in the end-entity certificate to retrieve the certificate(s) that have been issued to the CA that issued the end-entity certificate. As illustrated in Figure 2, Issuer=CA2 in the end-entity certificate will lead to CA2's certificate. Once we have retrieved CA2's certificate, we can use the Issuer DN in CA2's certificate to retrieve CA1's certificate. Finally, the Issuer DN in CA1's certificate leads us to CA0's certificate. The same logic applies when we build certification paths in the reverse direction (with the order reversed, of course). This is discussed in more detail later.

If Certification Authorities (CAs) were guaranteed to have only one public/private signing key pair active at any given time, satisfying the name chaining requirement would be all that is required to construct a candidate certification path. However, it is possible (even likely) that CAs will have more than one valid signing key pair at the same time (e.g., to support CA key rollover). This means that name chaining alone may not be sufficient to determine if the certification path is a legitimate candidate that should be submitted to the certification path validation process. This leads us to the notion of "key identifier chaining" as discussed below.

AKIDs are used to distinguish one public key from another when a given Certification Authority (CA) has multiple signing keys, and SKIDs provide a means to identify certificates that contain a specific public key.

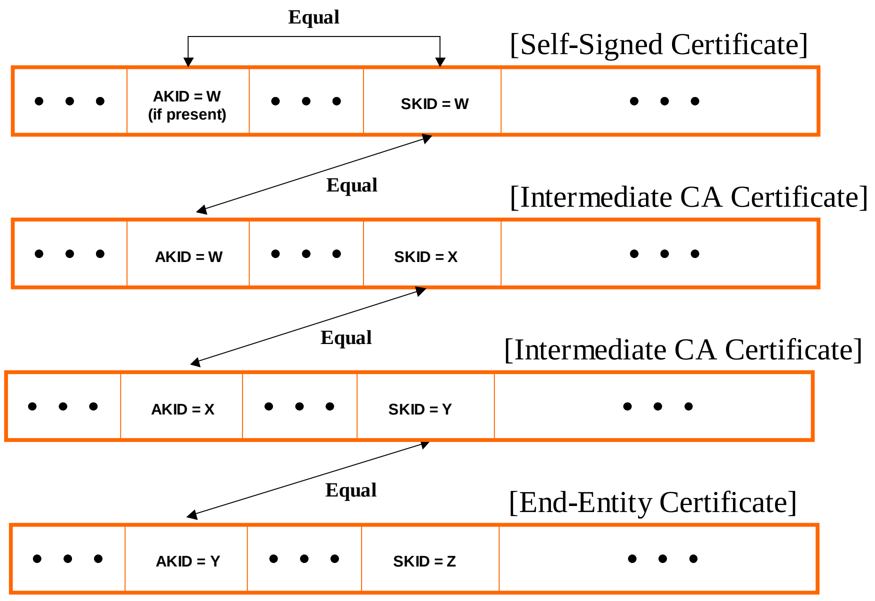
Key Identifier Chaining

The Authority Key Identifier (AKID) and Subject Key Identifier (SKID) are certificate extensions that can be used to help facilitate the certification path construction process. As discussed in X.509 and the Internet Certificate and CRL Profile (RFC3280), AKIDs are used to distinguish one public key from another when a given Certification Authority (CA) has multiple signing keys, and SKIDs provide a means to identify certificates that contain a specific public key.[4]

Similar to "name chaining" between a trust anchor and an end-entity certificate, the SKID of the first certificate should be the AKID of the next certificate in the path, and so on. Figure 3 helps to illustrate this concept. Note that since the AKID and SKID are certificate extensions, the concept of key identifier chaining applies only to Version 3 public key certificates.

[4] The reader may be interested in a recently published PKI Forum Implementation Guideline regarding AKIDs/SKIDs in the context of cross-certification. This guideline can be found at http://www.pkiforum.org/resources.html.

Figure 3: Key Identifier Chaining Illustration



AKID/SKID Structure and Use

In accordance with X.509, the AKID can be represented using a **keyIdentifier**, an **authorityCertIssuer**, **authoritySerialNumber** pair, or both. However, X.509 also states that:

> *The **keyIdentifier** form can be used to select CA certificates during path construction. The **authorityCertIssuer, authoritySerialNumber** pair can only be used to provide preference to one certificate over others during path construction.*

In addition, the Internet Certificate and CRL Profile (see RFC3280) states that the **keyIdentifier** field in the AKID must be included in all certificates generated by conforming CAs (with the exception of self-signed certificates, in which case the **keyIdentifier** would be included within the SKID and, optionally, within the AKID). Further, the Internet Certificate and CRL Profile states that:

> *The **keyIdentifier** field of the Authority Key Identifier extension MUST be included in all certificates generated by conforming CAs to facilitate chain building.*

In accordance with X.509, the SKID can only be represented using the **keyIdentifier** (i.e., the SKID syntax does not include the **authorityCertIssuer, authoritySerialNumber** pair). Further, the Internet Certificate and CRL Profile states that all CA certificates must include the SKID extension and:

> *The value of the Subject Key Identifier MUST be the value placed in the key identifier field of the Authority Key Identifier extension … of certificates issued by the subject of this certificate.*

Thus, a CA conforming to the Internet Certificate and CRL profile must populate the AKID of all certificates that it issues with the same **keyIdentifier** that is populated in the SKID of the certificate used to verify the digital signature on those issued certificates.

The Internet Certificate and CRL Profile states that: the keyIdentifier field of the Authority Key Identifier extension MUST be included in all certificates generated by conforming CAs to facilitate chain building and the value of the Subject Key Identifier MUST be the value placed in the keyIdentifier field of the Authority Key Identifier extension...of certificates issued by the subject of this certificate.

Calculation of keyIdentifier Value

There are multiple methods available for calculating the **keyIdentifier**. The following methods are specifically mentioned in the Internet Certificate and CRL Profile:

- the 20 byte SHA-1 value of the subject public key (not including tag, length, and unused bits)
- a four bit value 0100 followed by the least significant 60bits of the SHA-1 value of the subject public key (not including tag, length, and unused bits)
- a monotonically increasing sequence of integers

Note that these are recommendations only - the profile does not mandate a particular algorithm.

PKI forum

This leads us to note two important considerations. First, while it is possible that the **keyIdentifier** could be unique across multiple PKI domains (e.g., when the SHA-1 hash value of the public key is used), the only guarantee is that the **keyIdentifier** is unique relative to a given public key in the context of a given issuer. This means that we cannot rely solely on the **keyIdentifier** value to guarantee that we have the right certificate. Second, it is possible that an issuer may renew a certificate (i.e., the same public key is issued in a new certificate) and the **keyIdentifier** value may therefore be the same in more than one certificate[5]. This is why the use of the **authorityCertIssuer**, **authoritySerialNumber** pair may be needed to select one certificate over another (i.e., the serial number is guaranteed to be unique for every certificate issued by a given CA even though the same public key may have been re-issued).

X.509 defines specific directory attributes where CA and end-entity certificates are to be stored. While there has been some controversy over which attributes should be used, the 4th edition of X.509 mandates certain requirements and, in some cases, expected behavior.

Identifying Appropriate Certificates

Now that we have a better understanding surrounding some of the basics, let's turn our attention to some of the issues associated with locating the appropriate certificates.

In the enterprise context, most PKI vendors have relied on the use of repositories for the storage and retrieval of certificates. In many instances, this has been supported by an X.500-based directory with client-server interaction supported by the Lightweight Directory Access Protocol (LDAP). X.509 defines specific directory attributes where CA and end-entity certificates are to be stored. While there has been some controversy over which attributes should be used, the 4th edition of X.509 mandates certain requirements and, in some cases, expected behavior. The syntax and expected use of these attributes in accordance with X.509 are described in the paragraphs that follow.

X.509 Certificate Attributes

X.509 defines several certificate attributes. For the purposes of this paper, we are concerned mostly with the **cACertificate** attribute and the **crossCertificatePair** attribute. The **userCertificate** attribute used to store end-entity certificates also comes into play, but for purposes of simplicity we assume that the end-entity certificate is already in-hand before the certification path construction process begins. We also note that X.509 defines a **pkiPath** attribute, and we discuss this briefly at the end of this section.

As specified in X.509:

> *The **cACertificate** attribute of a CA's directory entry shall be used to store self-issued certificates (if any) and certificates issued to this CA by CAs in the same realm as this CA. The definition of realm is purely a matter of local policy.*

By definition, *self-issued certificates* are CA certificates where the Issuer DN and the Subject DN are the same. A *self-signed certificate* is a self-issued certificate where the private key used to sign that certificate corresponds to the public key within that certificate (i.e., the public key within the certificate is used to verify the digital signature associated with that certificate). X.509 makes the distinction, since a self-issued certificate is not guaranteed to also be a self-signed certificate. This can be illustrated using CA key rollover as an example. To allow for graceful transition from the old signing key pair to the new signing key pair, the CA should issue a certificate that contains the old public key signed by the new private signing key and a certificate that contains the new public key signed by the old private signing key. Both of these certificates are self-issued, but neither is self-signed. Note that these are in addition to the two self-signed certificates (one old, one new).

The cross-certificate pair attribute consists of two elements: 1) **issuedToThisCA** and 2) **issuedByThisCA**. (Prior to the 4th edition of X.509 these elements were referred to as forward and reverse, respectively.) As specified in X.509:

> *The **issuedToThisCA** elements of the **crossCertificatePair** attribute of a CA's directory entry shall be used to store all, except self-issued certificates issued to this CA. Optionally, the **issuedByThisCA** elements of the **crossCertificatePair** attribute of a CA's directory entry may contain a subset of certificates issued by this CA to other CAs. If a CA issues a certificate to another CA, and the subject CA is not a subordinate to the issuer CA in a hierarchy, then the issuer CA shall place that certificate in the **issuedByThisCA** element of **crossCertificatePair** attribute of its own directory entry.*

[5] The PKI Forum does not necessarily endorse this behavior, we simply note that it is possible and therefore needs to be considered.

This leads us to several conclusions:

First, we would expect each CA to populate the **cACertificate** attribute of its own directory entry with its own self-issued certificates, if any. This will include any self-signed certificates as well as any certificates issued by the CA in support of CA key update.

Second, we may see certificates issued to a CA that have been issued by other CAs that belong to the same "realm" to be stored in the **cACertificate** attribute of its own directory entry. While realm is purposely undefined, one possible example is a single PKI domain consisting of a hierarchy of CAs. In this particular case we might expect subordinate CAs to store the certificates issued to them by their superior CA in the **cACertificate** attribute of their own directory entry. However, given that "realm" is undefined and subject to interpretation, there is no guarantee that this will be implemented consistently by all vendors.

Third, we would expect to see the **issuedToThisCA** element of the cross-certificate pair attribute to be populated with certificates that have been issued to this CA by other CAs. (This suggests that both the **cACertificate** and **issuedToThisCA** might be populated when the issuing CA is in the same realm as the issued to CA.)

Finally, we would expect the **issuedByThisCA** attribute to be populated with certificates issued by this CA to other CAs when those CAs do not belong to the same hierarchy. In particular, we would expect to see the **issuedByThisCA** attribute populated with certificates issued to peer CAs as commonly found in a distributed trust model. Although the **issuedByThisCA** could be populated in a hierarchical domain, it is not reasonable to assume that this will always be the case.

Thus, we can expect the following mandatory behavior for CAs that conform to the 4th edition of X.509:

1. all self-issued certificates must be stored in the **cACertificate** attribute of the issuing CA's directory entry;
2. all certificates issued to a CA except for self-issued certificates must be stored in the **issuedToThisCA** element of the cross-certificate pair attribute of that CA's directory entry; and
3. all certificates issued by a CA to a non-subordinate or peer CA must be stored in the **issuedByThisCA** element of the cross-certificate pair attribute of the issuing CA's directory entry.

Thus, we can expect the following mandatory behavior for CAs that conform to the 4th edition of X.509:

1. *all self-issued certificates must be stored in the **cACertificate** attribute of the issuing CA's directory entry;*
2. *all certificates issued to a CA except for self-issued certificates must be stored in the **issuedToThisCA** element of the cross-certificate pair attribute of that CA's directory entry; and*
3. *all certificates issued by a CA to a non-subordinate or peer CA must be stored in the **issuedByThisCA** element of the cross-certificate pair attribute of the issuing CA's directory entry.*

The rest of the behavior described above is considered optional. While some of these options can be implemented and used to help in the path construction process, this paper assumes that these options may not be used and therefore cannot be relied upon.

The 4th Edition of X.509 also defines a **pkiPath** attribute that can be used to store partial or complete certification paths. Each path is represented by a sequence of cross-certificates. X.509 describes the use of this attribute as follows:

This attribute can be stored in the CA directory entry and would contain some certification paths from that CA to other CAs. This attribute, if used, enables more efficient retrieval of cross-certificates that form frequently used certification paths. As such there are no specific requirements for this attribute to be used and the set of values that are stored in the attribute will likely not represent the complete set of forward certification paths for any given CA.

Since we are unaware of any existing implementations that make use of the **pkiPath** attribute in this way, and since X.509 implies that use of this attribute is optional, we do not explore the use of this attribute further.

Use of Private Certificate Extensions

In many cases, client systems are configured with default IP addresses and/or DNS names of one or more repositories. These repositories are queried for the necessary certificates. In addition, search bases are often established so the appropriate certificates can be easily located and retrieved.

As an alternative, the Internet Certificate and CRL Profile defines a private extension referred to as the Authority Information Access (AIA) extension. As stated in RFC3280, the AIA "indicates how to access CA information and services for the issuer of the certificate in which the extension appears." One possible use of this extension is to point to a list of "CAs that have issued certificates superior to the CA that issued the certificate containing this extension…[this] is intended to aid certificate users in the selection of a certification path that terminates at a point trusted by the certificate user." It remains to be seen how use of the AIA private extension may evolve over time. Currently, Microsoft (Windows 2000 and Windows XP) uses the AIA to point to the location of the issuing CA's certificate [Ref5]. In the future, we may see vendors use the AIA to point to additional information such as a list of superior CAs.

It remains to be seen how use of the AIA private extension may evolve over time... In the future, we may see vendors use the AIA to point to additional information such as a list of superior CAs.

The latest Internet Draft Certificate and CRL Profile also defines a private extension referred to as the Subject Information Access (SIA) extension. This extension "indicates how to access information and services for the subject of the certificate in which the extension appears." One possible use of this extension is to identify where a CA publishes the certificates (and possibly CRLs) that it issues. This is something that we may see exploited in the future.

Other Options?

It is common practice to send the public key certificate necessary to verify a digital signature along with digitally signed data[6]. For example, secure e-mail based on S/MIME supports this. It is also possible to send partial or complete certification paths to the relying parties via protocol - perhaps using the **pkiPath** attribute syntax defined in X.509. In these cases, some (or perhaps all) of the certification path may not have to be retrieved from external sources. However, in complex, richly connected PKI domains, it is unlikely that the originator of a message will know a complete certification path between the relying party's trust anchor and the originator's certificate. Even if such a path were known, there is no guarantee that this will be the path that will meet the path validation criteria levied by the relying party (or by any intermediate CAs). While there may be some cases where we can take advantage of the knowledge of partial paths, in most cases we will need a robust path construction algorithm capable of discovering one or more complete certification path(s).

While there may be some cases where we can take advantage of the knowledge of partial paths, in most cases we will need a robust path construction algorithm capable of discovering one or more complete certification path(s).

Constructing Certification Paths

The examples that follow are designed to illustrate some of the issues that can be encountered when building certification paths. In general, the examples are independent from the mechanism used to retrieve the necessary certificates. Conformance with the 4th edition of X.509 is assumed.

Simple Example - Strict Hierarchy

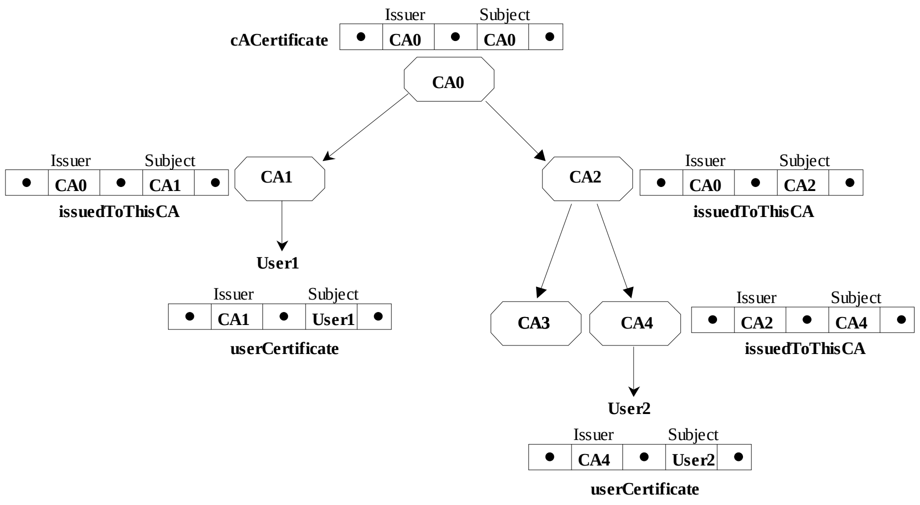
Let's start with a relatively simple example. Figure 4 illustrates a strict hierarchy of CAs. The octagons represent CAs, the arrows represent certificate issuance, and the sectioned rectangles represent certificates. The specific X.509 directory attribute that is used to store each certificate is identified. Let's examine what happens when User1 sends a digitally signed e-mail to User2. It is assumed that User1's verification certificate is conveyed along with the message itself.

In this example, User2 is the relying party. Since we are attempting to verify a digital signature using User1's verification certificate, we need to construct a certification path between User1's certificate and a trust anchor recognized by User2. In this case, CA0 is the root CA and is, by definition, the common trust anchor for all users within this strict hierarchy. Basically, User2 wants to know if CA0 has established a trust relationship (either directly or indirectly) with the issuer of User1's certificate (in this case, CA1). In other words, if the relying party software is able to resolve the certification path CA0->CA1->User1, then we would have a candidate certification path that could be submitted to the path validation logic.

[6] Certificates used to verify digital signatures are sometimes referred to as "verification certificates."

Building certification paths within strict hierarchies is rather straightforward. Paths are typically constructed in the forward direction (i.e., we start with the target certificate and work our way to a recognized trust anchor). Thus, the relying party software will start with User1's verification certificate and work its way to CA0. Since the relying party software knows CA1 is the issuer of User1's certificate (the Issuer DN in User1's certificate is CA1), this is accomplished by retrieving the **issuedToThisCA** element that is stored in the directory entry for CA1. We then discover that CA0 is the issuer of CA1's certificate, and we now have a complete candidate certification path between User1's certificate and a trust anchor recognized by User2.

Figure 4: Path Construction Illustration – Strict Hierarchy



Building certification paths within strict hierarchies is rather straightforward. Paths are typically constructed in the forward direction (i.e., we start with the target certificate and work our way to a recognized trust anchor).

More Complex Example -- Single Cross-Certification with Hierarchical and Distributed Trust Models

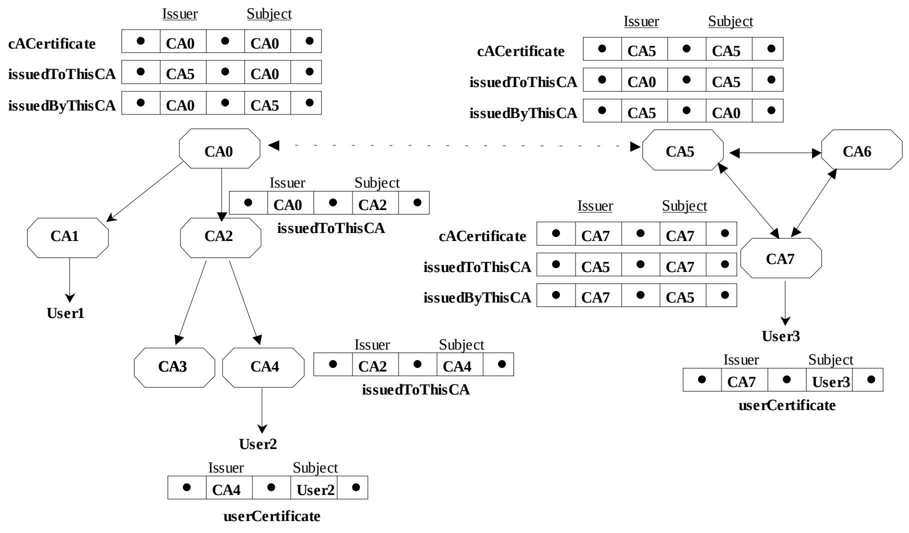
We will use Figure 5 to illustrate a more complex path construction process between two PKI domains that have mutually cross-certified with one another. As before, the octagons represent CAs and the sectioned rectangles represent certificates. The specific X.509 directory attribute that is used to store each certificate is identified. As in the previous example, the unidirectional arrows represent certificate issuance. The bi-directional arrows (including the dashed arrow between CA0 and CA5) represent mutual cross-certification. Thus, the PKI domain on the left is a hierarchical trust model whereas the PKI domain on the right is a distributed trust model.

In this example, User3 receives a digitally signed e-mail from User2. User3 is the relying party and the certificate to be validated is User2's verification certificate. User2's verification certificate accompanies the e-mail sent to User3.

The task at hand is to construct a certification path between User2's verification certificate and a trust anchor that User3 recognizes (in this case, CA7). We can see that one possibility is to work our way "out" from the relying party's trust anchor to eventually discover the cross-certificate that CA5 issued to CA0. This is referred to as path construction in the reverse direction and it produces a partial path CA7->CA5->CA0. This partial path is constructed by retrieving the **issuedByThisCA** certificates under CA7's directory entry, which, in turn, leads to the retrieval of the **issuedByThisCA** certificates under CA5's directory entry.[7]

[7] It is recognized that there are actually two paths between CA7 and CA5 (i.e., CA7->CA5 and CA7->CA6->CA5). However, for the time being we will concentrate on the shorter path. We will revisit multiple path issues a bit later.

Figure 5: Path Construction Illustration – Single Inter-Domain Cross-Certification



*In a strict hierarchy, working in the forward direction makes sense, since we are always guaranteed to find the **issuedToThisCA** element of the cross-certificate pair attribute populated with the certificate(s) that have been issued to each subordinate CA.*

Once we "reach" CA0, it may not be possible to continue to construct the path in the reverse direction, since we cannot depend on the **issuedByThisCA** element to be populated by CAs that belong to a hierarchy (recall that population of this element is considered optional in this case). Even if the **issuedByThisCA** element is populated by each CA in the hierarchical domain, searching in the reverse direction from CA0 would be less efficient than searching in the forward direction from CA4. Specifically, we would encounter two possible paths emanating from CA0 (not counting the link back to CA5), but there is only one possible path leading from CA4 back up to CA0.

So what do we do next? We now try to construct the rest of the path in the forward direction, which means we work our way back from the target certificate to CA0. This is accomplished by looking for the **issuedToThisCA** element in CA4's directory entry (recall that CA4 is the issuer of the target certificate). (Optionally, we may look for this information in the **cACertificate** attribute, but this attribute is not guaranteed to be populated as discussed earlier.) This leads us to the discovery of the certificate issued by CA2 to CA4 which, in turn, leads us to the certificate issued by CA0 to CA2. We now have a complete chain of certificates between the target certificate and the trust anchor recognized by the relying party given by CA7->CA5->CA0->CA2->CA4->User2. Of course, in this example the order of the path construction may be reversed. That is, we could have constructed the partial path CA0->CA2->CA4->User2 first, and then we could work our way back from CA7 to CA0.

To complete the picture, let's take a quick look at what happens when User3 sends a digitally signed e-mail to User2. Now User2 is the relying party, and we need to construct a certification path between User3's verification certificate and a trust anchor recognized by User2 (in this case, CA0). Specifically, we need CA0->CA5->CA7->User3 (again ignoring the path through CA6). Similar to the previous example, we want to work our way "out" from the relying party's trust anchor. Thus, we can construct the necessary path by retrieving the cross-certificate that CA0 issued to CA5 obtained from the **issuedByThisCA** attribute in CA0's directory entry, which in turn leads us to the cross-certificate issued by CA5 to CA7 obtained from the **issuedByThisCA** attribute in CA5's directory entry.

Forward versus Reverse Certification Path Construction

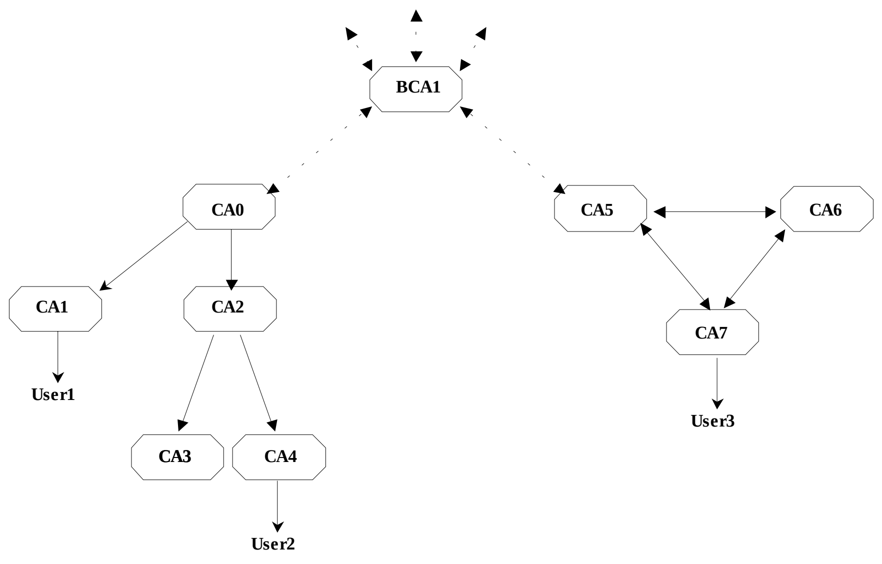
In a strict hierarchy, working in the forward direction makes sense, since we are always guaranteed to find the **issuedToThisCA** element of the cross-certificate pair attribute populated with the certificate(s) that have been issued to each subordinate CA. In addition, it is typically more efficient to construct paths in the forward direction within a hierarchy. In some cases, this will resolve to one and only one possible certification path. In others, a limited number of alternative paths may be constructed as a result of the existence of multiple certificates per CA. Use of the **keyIdentifier** may be used to resolve to a single path in some cases, but this may not always be possible (e.g., when Version 1 or 2 certificates are being used or when the same public key is re-issued and the keyIdentifier value is based on that public key). In any case, the number of possible certification paths is well bounded when working in the forward direction within a strict hierarchy.

In the previous two examples above, one might suggest that the certification paths could have been constructed in the forward direction – even in the distributed environment. Although this may be true in this simple example, this is possible only due to the fact that there is a single inter-domain cross-certification and a limited number of CAs involved. When we encounter a distributed trust model, building certification paths in the forward direction can become much less efficient. This is due to the fact that we may encounter tens or even hundreds of forward elements associated with a given CA, not just one or two.

This is illustrated in Figure 6, where we have introduced a bridge CA (BCA1) between the two PKI domains. Although we can easily construct the path from the target certificate to CA0 (as we saw earlier), this becomes much more complex once we try to continue in the forward direction working "outward" from CA0. Specifically, BCA1 might be cross-certified with tens or even hundreds of other CAs (including other bridge CAs), and those CAs may be cross-certified with tens or hundreds more. If we are looking at the **issuedToThisCA** element only, we could encounter a large number of CAs that lead away from the path we are seeking. Attempting to construct every possible path in the forward direction is clearly less practical under these circumstances. Constructing in the reverse direction also involves trial and error, but we will always be working with a (partial) path emanating from the relying party's trust anchor since we are answering the question "who have you issued certificates to" rather than "who has issued certificates to you." While this may not be enough in itself to convince everyone that reverse direction is most appropriate in a distributed environment, there are also a number of efficiencies that can be realized when building paths in the reverse direction, as discussed in [Ref3].

When we encounter a distributed trust model, building certification paths in the forward direction can become much less efficient. This is due to the fact that we may encounter tens or even hundreds of forward elements associated with a given CA, not just one or two.

Figure 6: Path Construction Illustration – Fully Distributed



This leads us to conclude that certification path construction in the forward direction is optimal for hierarchical trust models, and certification path construction in the reverse direction is optimal for distributed trust models.

This leads us to conclude that certification path construction in the forward direction is optimal for hierarchical trust models, and certification path construction in the reverse direction is optimal for distributed trust models.

Evaluating Certification Paths During the Path Construction Process

So far, our discussion has not addressed any form of path validation or other helpful criteria that might be taken into consideration during the path construction process. While it is possible to ignore any validation criteria during the path construction process, this can lead to enormous inefficiencies that could otherwise be avoided. This is especially true in complex, richly interconnected PKI environments.

Let's illustrate this point by revisiting the example we discussed using Figure 5, only now using the topology represented in Figure 6. As we work our way "out" from User3's trust anchor, we now encounter the bridge CA (BCA1) instead of the direct cross-certificate that CA5 issued to CA0. Not counting the cross-certificate BCA1 issued to CA5, we now have four cross-certificates emanating from BCA1 that come into play, which, in turn, could lead to dozens more. We would be much better off if we recognized that the cross-certificate BCA1 issued to CA0 is a much more likely candidate to take us closer to where we need to go. So when we retrieve all the **issuedByThisCA** entries associated with BCA1's directory entry, it would greatly improve performance if we somehow knew CA0 is in the path we are seeking.

Since strict hierarchies are typically based on hierarchical naming conventions, we could take advantage of the fact that the DN of CA0 is very likely a subset of the Relative Distinguished Names (RDNs) in the target certificate (i.e., User2's certificate) or the RDNs of CA4 (which we know from the Issuer DN in User2's certificate). In other words, once we discover the cross-certificate issued by CA5 to CA0, we will know we are on the right track, since the RDNs in the DN of CA0 will match a subset of the RDNs in the Issuer DN and/or Subject DN of User2's certificate. This does not mean that the other entries will not lead to a legitimate path, but it does mean that we may want to try CA0 first. Alternatively, we may have constructed the partial path CA0->CA2->CA4->User2 first, so we would know we have a candidate certification path once we encountered the cross-certificate BCA1 issued to CA0 when working in the reverse direction.

Some vendors have developed a certification path prioritization system in an attempt to submit the most likely candidate paths to the path validation logic first.

As mentioned earlier, it is possible to take certain things into consideration in order to maximize the chance that the candidate certification path submitted to the path validation logic will turn out to be an acceptable path. In fact, some vendors have developed a certification path prioritization system in an attempt to submit the most likely candidate paths to the path validation logic first. For example, a freeware Certification Path Library (CPL) that prioritizes the order in which certificates are considered during the path construction process is available for download from http://www.entrust.com/entrustcygnacom/products/index.htm. According to the associated *Path Development API - Interface Control Document* [Ref4], the prioritization rules are as follows:

1. Certificates retrieved from the cACertificate attribute should have priority over certificates retrieved from the crossCertificate attribute
2. Certificates in which issuer algorithm OID = subject algorithm OID should have priority
3. Certificates that assert policies should have priority over certificates that do not. Among certificates that assert policies, those certificates which match more policies in the initial-acceptable-policy-set should have priority
4. Certificates with fewer RDN elements in the Issuer DN should have priority
5. Certificates that match more RDNs between the issuer DN and relying party trust anchor DN should have priority
6. Certificates that match more RDNs between the subject DN and the issuer DN should have priority
7. Certificates with longer validity periods (furthest notAfter date) should have priority

Note that a number of assumptions (such as breadth versus depth) will have an influence on prioritization rules, and this information is supplied as an example only. The referenced document is several years old and it is possible that different prioritization rules may have been selected based on more recent developments, including lessons learned from some of the recently concluded interoperability initiatives, such as the US Federal Bridge CA demonstration and the UK Communications-Electronics Security Group

(CESG) interoperability initiatives. In any case, vendors may elect to adopt variations on such a prioritization system. This paper does not make any specific recommendations in this regard.

It may also be possible to take certain path validation criteria into consideration during the path construction process. As above, the idea is to help increase the chances that one or more certification paths will pass the more comprehensive path validation procedure. However, this allows us to eliminate certain paths from consideration that might otherwise be considered viable candidates.

Building Certification Paths: Forward vs. Reverse [Ref3] identifies several path validation criteria that might be taken into consideration during the path construction process, including:

- name constraints,
- certificate policy processing, and
- certificate validation (including revocation status check and digital signature verification).

The idea is that you can throw out certificates that will not meet a subset of the path validation criteria. This helps to promote a "prune as you go" process, particularly when building paths in the reverse direction.

Taking one or more of these into consideration during the certification path construction process could possibly eliminate certain paths that might otherwise be considered viable candidates for submission to the path validation process. There may be other path validation criteria that a vendor may want to consider as well (e.g., path length constraints). The idea is that you can throw out certificates that will not meet a subset of the path validation criteria. This helps to promote a "prune as you go" process, particularly when building paths in the reverse direction.

However, there are potential drawbacks that may be associated with this. For example, there are certain things that cannot be determined until the entire candidate path is constructed (e.g., a require explicit policy requirement could be introduced by the last intermediate CA certificate in the path). In addition, this pre-supposes that the path construction logic will be able to process at least a subset of the path validation logic as outlined in the 4th edition of X.509 and the Internet Certificate and CRL Profile. This may not be straightforward in the case where a vendor has separated the path construction and path validation processes. Finally, we could potentially perform certain path validation checking that fails to eliminate a candidate path, even though the full path validation procedure later determines that some other criterion was not met. This increases the amount of work during path construction. It may also duplicate work, since many of the same checks made during path construction may also be made during the path validation process. Nonetheless, it seems clear that certain factors should be taken into consideration during the path construction process. The extent to which these are implemented are at the discretion of each vendor.

Multiple Paths

Even in our simpler examples, we saw that it is possible to have more than one candidate path between the relying party's trust anchor and the target end-entity certificate. For example, in the single cross-certification example (using Figure 5), there were two candidate certification paths between User3's trust anchor and User2's certificate (i.e., CA7->CA5->CA0->CA2->CA4->User2 and CA7->CA6->CA5->CA0->CA2->CA4->User2). It is possible that multiple candidate paths would meet all of the certification path evaluation criteria equally. In the absence of any other criteria that may be used to prioritize one candidate path over another, it would make sense to submit the shortest path to the path validation logic before submitting longer paths.

In the absence of any other criteria that may be used to prioritize one candidate path over another, it would make sense to submit the shortest path to the path validation logic before submitting longer paths.

Detecting Loops

One of the things we need to avoid when constructing certification paths is looping. This could occur when a series of cross-certificates leads us back to a cross-certificate that is already part of the candidate certification path. In a richly interconnected environment, this possibility may often be encountered.

The 4th edition of X.509 makes it clear that a certificate is never permitted to appear more than once in any given certification path (see Section 10.1, sub-paragraph a). We can therefore avoid loops by keeping track of all certificates as the path is constructed and making sure that any certificate already included in the prospective path does not appear in the path again. If we encounter a duplicate, we can backtrack as necessary. Note that keeping track of subject DNs of intermediate CAs is not an appropriate duplicate detection mechanism, since a CA may have more than one active public key certificate issued under the same DN. We must check to make sure the exact same certificate is not used again, which could be accomplished through direct certificate comparison.

Summary

This paper discusses many of the issues that should be taken into consideration when developing or evaluating certification path construction algorithms.

A number of observations and/or recommendations have been made throughout this paper. In summary, a robust path construction algorithm should be able to:

- construct certification paths in both the forward and reverse directions,
- search repositories for appropriate certificates in accordance with the 4th edition of X.509,
- recognize the AIA extension and use it to retrieve CA certificates,
- use RDN matching to identify more likely candidate certification paths,
- give precedence to shorter paths over longer paths (in the absence of any other overriding criteria), and
- detect and avoid duplicate certificates.

In addition, a number of path validation criteria might be taken into consideration during the path construction process, including those discussed in [Ref3].

References

[Ref1] ITU-T Recommendation X.509 "Information Technology—Open Systems Interconnection—The Directory: Public Key and Attribute Certificate Frameworks, *March 2000 (equivalent to ISO/IEC 9594-8, 2000)*.

[Ref2] *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile,* Internet Request for Comments 3280, Housley, R., W. Ford, W. Polk, and D. Solo., April 2002.

[Ref3] *Building Certification Paths: Forward vs. Reverse*, Yassir Elley, Anne Anderson, Steve Hanna, Sean Mullen, Radia Perlman, Seth Proctor (See http://www.isoc.org/isoc/conferences/ndss/01/2001/papers/elley.pdf).

[Ref4] Path Development API – Interface Control Document, Cygnacom Solutions (See http://www.entrust.com/entrustcygnacom/cert/CPL_1_3_ICD.doc).

[Ref5] Troubleshooting Certificate Status and Revocation, Microsoft Technical Paper by Brian Komar (See http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/tshtcrl.asp).