                   Internet X.509 Public Key Infrastructure:
                         Certification Path Building

Status of This Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard of any kind. Distribution of this
   memo is unlimited.

Abstract

   This document provides guidance and recommendations to developers
   building X.509 public-key certification paths within their
   applications. By following the guidance and recommendations defined
   in this document, an application developer is more likely to develop
   a robust X.509 certificate-enabled application that can build valid
   certification paths across a wide range of PKI environments.

Table of Contents

1. Introduction

   [X.509] public key certificates have become an accepted method for
   securely binding the identity of an individual or device to a public
   key, in order to support public key cryptographic operations such as
   digital signature verification and public key-based encryption.
   However, prior to using the public key contained in a certificate, an
   application first has to determine the authenticity of that
   certificate, and specifically, the validity of all the certificates
   leading to a trusted public key, called a trust anchor. Through
   validating this certification path, the assertion of the binding made
   between the identity and the public key in each of the certificates
   can be traced back to a single trust anchor.

   The process by which an application determines this authenticity of a
   certificate is called certification path processing. Certification
   path processing establishes a chain of trust between a trust anchor
   and a certificate. This chain of trust is composed of a series of

certificates known as a certification path. A certification path
begins with a certificate whose signature can be verified using a
trust anchor and ends with the target certificate. Path processing
entails building and validating the certification path to determine
whether a target certificate is appropriate for use in a particular
application context. See Section 3.2 of [RFC3280] for more
information on certification paths and trust.

1.1. Motivation

Many other documents (such as [RFC3280]) cover certification path
validation requirements and procedures in detail but do not discuss
certification path building because the means used to find the path
does not affect its validation. This document therefore is an effort
to provide useful guidance for developers of certification path-
building implementations.

Additionally, the need to develop complex certification paths is
increasing. Many PKIs are now using complex structures (see Section
1.5) rather than simple hierarchies. Additionally, some enterprises
are gradually moving away from trust lists filled with many trust
anchors, and toward an infrastructure with one trust anchor and many
cross-certified relationships. This document provides helpful
information for developing certification paths in these more
complicated situations.

1.2. Purpose

This document provides information and guidance for certification
path building. There are no requirements or protocol specifications
in this document. This document provides many options for performing
certification path building, as opposed to just one particular way.
This document draws upon the authors' experiences with existing
complex certification paths to offer insights and recommendations to
developers who are integrating support for [X.509] certificates into
their applications.

In addition, this document suggests using an effective general
approach to path building that involves a depth first tree traversal.
While the authors believe this approach offers the balance of
simplicity in design with very effective and infrastructure-neutral
path-building capabilities, the algorithm is no more than a suggested
approach. Other approaches (e.g., breadth first tree traversals)
exist and may be shown to be more effective under certain conditions.
Certification path validation is described in detail in both [X.509]
and [RFC3280] and is not repeated in this document.

This document does not provide guidance for building the
certification path from an end entity certificate to a proxy
certificate as described in [RFC3820].

1.3. Terminology

Terms used throughout this document will be used in the following
ways:

Building in the Forward direction: The process of building a
   certification path from the target certificate to a trust anchor.
   'Forward' is the former name of the crossCertificatePair element
   'issuedToThisCA'.

Building in the Reverse direction: The process of building a
   certification path from a trust anchor to the target certificate.
   'Reverse' is the former name of the crossCertificatePair element
   'issuedByThisCA'.

Certificate: A digital binding that cannot be counterfeited between
   a named entity and a public key.

Certificate Graph: A graph that represents the entire PKI (or all
   cross-certified PKIs) in which all named entities are viewed as
   nodes and all certificates are viewed as arcs between nodes.

Certificate Processing System: An application or device that
   performs the functions of certification path building and
   certification path validation.

Certification Authority (CA): An entity that issues and manages
   certificates.

Certification Path: An ordered list of certificates starting with a
   certificate signed by a trust anchor and ending with the target
   certificate.

Certification Path Building: The process used to assemble the
   certification path between the trust anchor and the target
   certificate.

Certification Path Validation: The process that verifies the binding
   between the subject and the subject-public-key defined in the
   target certificate, using a trust anchor and set of known
   constraints.

Certificate Revocation List (CRL): A signed, time stamped list
   identifying a set of certificates that are no longer considered
   valid by the certificate issuer.

CRL Signer Certificate: The specific certificate that may be used for
   verifying the signature on a CRL issued by, or on behalf of, a
   specific CA.

Cross-Certificate: A certificate issued by one CA to another CA for
   the purpose of establishing a trust relationship between the two
   CAs.

Cross-Certification: The act of issuing cross-certificates.

Decision Tree: When the path-building software has multiple
   certificates to choose from, and must make a decision, the
   collection of possible choices is called a decision tree.

Directory: Generally used to refer an LDAP accessible repository for
   certificates and PKI information. The term may also be used
   generically to refer to any certificate storing repository.

End Entity: The holder of a private key and corresponding
   certificate, whose identity is defined as the Subject of the
   certificate. Human end entities are often called "subscribers".

Is-revocation-signer indicator: A boolean flag furnished to the
   path-building software. If set, this indicates that the target
   certificate is a Revocation Signer certificate for a specific CA.
   For example, if building a certification path for an indirect CRL
   Signer certificate, this flag would be set.

Local PKI: The set of PKI components and data (certificates,
   directories, CRLs, etc.) that are created and used by the
   certificate using organization. In general, this concept refers
   to the components that are in close proximity to the certificate
   using application. The assumption is that the local data is more
   easily accessible and/or inexpensive to retrieve than non-local
   PKI data.

Local Realm: See Local PKI.

Node (in a certificate graph): The collection of certificates having
   identical subject distinguished names.

Online Certificate Status Protocol (OCSP): An Internet protocol used
   by a client to obtain the revocation status of a certificate from
   a server.

   OCSP Response Signer Certificate: The specific certificate that may
      be used for verifying the signature on an OCSP response. This
      response may be provided by the CA, on behalf of the CA, or by a
      different signer as determined by the Relying Party's local
      policy.

   Public Key Infrastructure (PKI): The set of hardware, software,
      personnel, policy, and procedures used by a CA to issue and manage
      certificates.

   Relying Party (RP): An application or entity that processes
      certificates for the purpose of 1) verifying a digital signature,
      2) authenticating another entity, or 3) establishing confidential
      communications.

   Revocation Signer Certificate: Refers collectively to either a CRL
      Signer Certificate or OCSP Response Signer Certificate.

   Target Certificate: The certificate that is to be validated by a
      Relying Party. It is the "Certificate targeted for validation".
      Although frequently this is the End Entity or a leaf node in the
      PKI structure, this could also be a CA certificate if a CA
      certificate is being validated. (e.g., This could be for the
      purpose of building and validating a certification path for the
      signer of a CRL.)

   Trust (of public keys): In the scope of this document, a public key
      is considered trustworthy if the certificate containing the public
      key can be validated according to the procedures in [RFC3280].

   Trust List: A list of trust anchors.

   Trust Anchor: The combination of a trusted public key and the name of
      the entity to which the corresponding private key belongs.

   Trust Anchor Certificate: A self-signed certificate for a trust
      anchor that is used in certification path processing.

   User: An individual that is using a certificate processing system.
      This document refers to some cases in which users may or may not
      be prompted with information or requests, depending upon the
      implementation of the certificate processing system.

1.4. Notation

   This document makes use of a few common notations that are used in
   the diagrams and examples.

   The first is the arrow symbol (->) which represents the issuance of a
   certificate from one entity to another. For example, if entity H
   were to issue a certificate to entity K, this is denoted as H->K.

   Sometimes it is necessary to specify the subject and issuer of a
   given certificate. If entity H were to issue a certificate to entity
   K this can be denoted as K(H).

   These notations can be combined to denote complicated certification
   paths such as C(D)->B(C)->A(B).

1.5. Overview of PKI Structures

   When verifying [X.509] public key certificates, often the application
   performing the verification has no knowledge of the underlying Public
   Key Infrastructure (PKI) that issued the certificate. PKI structures
   can range from very simple, hierarchical structures to complex
   structures such as mesh architectures involving multiple bridges (see
   Section 1.5.4). These structures define the types of certification
   paths that might be built and validated by an application [MINHPKIS].
   This section describes four common PKI structures.

1.5.1. Hierarchical Structures

   A hierarchical PKI, depicted in Figure 1, is one in which all of the
   end entities and relying parties use a single "Root CA" as their
   trust anchor. If the hierarchy has multiple levels, the Root CA
   certifies the public keys of intermediate CAs (also known as
   subordinate CAs). These CAs then certify end entities'
   (subscribers') public keys or may, in a large PKI, certify other CAs.
   In this architecture, certificates are issued in only one direction,
   and a CA never certifies another CA "superior" to itself. Typically,
   only one superior CA certifies each CA.

```
                        +---------+
                   +---| Root CA |---+
                   |    +---------+   |
                   |                  |
                   |                  |
                   v                  v
                +----+             +----+
           +-----| CA |      +-----| CA |------+
           |    +----+       |    +----+        |
           |                 |                  |
           v                 v                  v
        +----+            +----+             +----+
    +--| CA |-----+      | CA |-+      +---| CA |---+
    |   +----+     |     +----+ |      |   +----+    |
    |    |         |      |     |      |    |        |
    |    |         |      |     |      |    |        |
    v    v         v      v     v      v    v        v
  +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
  | EE | | EE | | EE | | EE | | EE | | EE | | EE | | EE |
  +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
```
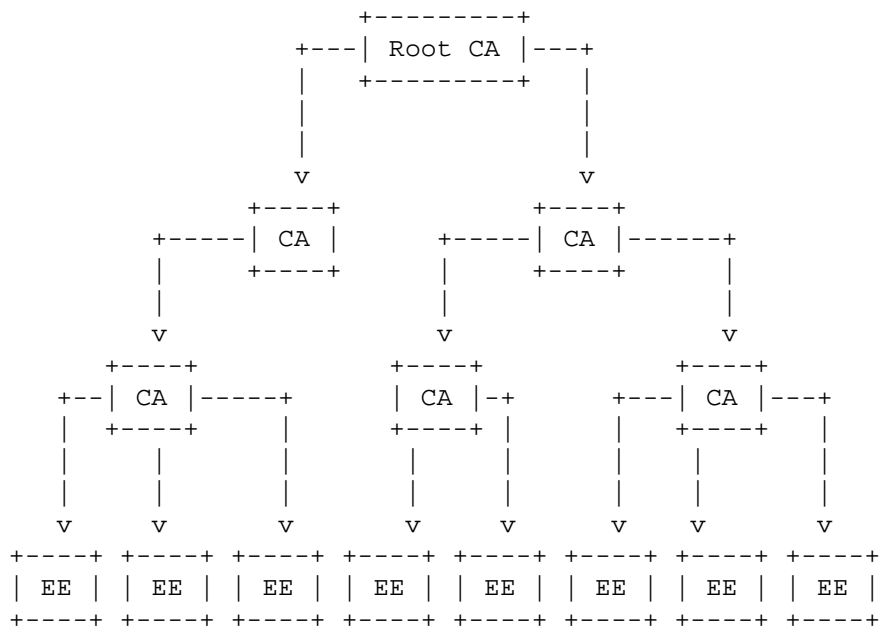
                Figure 1 - Sample Hierarchical PKI

   Certification path building in a hierarchical PKI is a
   straightforward process that simply requires the relying party to
   successively retrieve issuer certificates until a certificate that
   was issued by the trust anchor (the "Root CA" in Figure 1) is
   located.

   A widely used variation on the single-rooted hierarchical PKI is the
   inclusion of multiple CAs as trust anchors. (See Figure 2.) Here,
   end entity certificates are validated using the same approach as with
   any hierarchical PKI. The difference is that a certificate will be
   accepted if it can be verified back to any of the set of trust
   anchors. Popular web browsers use this approach, and are shipped
   with trust lists containing dozens to more than one hundred CAs.
   While this approach simplifies the implementation of a limited form
   of certificate verification, it also may introduce certain security
   vulnerabilities. For example, the user may have little or no idea of
   the policies or operating practices of the various trust anchors, and
   may not be aware of which root was used to verify a given
   certificate. Additionally, the compromise of any trusted CA private
   key or the insertion of a rogue CA certificate to the trust list may
   compromise the entire system. Conversely, if the trust list is
   properly managed and kept to a reasonable size, it can be an
   efficient solution to building and validating certification paths.

```
     +------------------------------------------------------------+
     |                        Trust List                          |
     |                                                            |
     |      +---------+        +---------+         +---------+     |
     |  +--| Root CA |        | Root CA |         | Root CA |     |
     |  |   +---------+        +---------+         +---------+     |
     |  |        |                  |                  |          |
     +--|------|------------------|----------------|----------+
        |      |                  |                |
        |      |                  |                |
        |      |                  v                |
        |      |               +----+              |
        |      |         +----| CA |---+           |
        |      |         |     +----+   |          |
        |      |         |              |          |
        |      |         v              v          v
        |      |      +----+         +----+     +----+
        |      |      | CA |---+     | CA |-+   | CA |---+
        |      |      +----+   |     +----+ |   +----+   |
        |      |        |      |       |    |     |      |
        |      |        |      |       |    |     |      |
        v      v        v      v       v    v     v      v
     +----+ +----+ +----+ +----+ +----+ +----+ +----+ +----+
     | EE | | EE | | EE | | EE | | EE | | EE | | EE | | EE |
     +----+ +----+ +----+ +----+ +----+ +----+ +----+ +----+
```

                 Figure 2 - Multi-Rooted Hierarchical PKI

1.5.2. Mesh Structures

   In a typical mesh style PKI (depicted in Figure 3), each end entity
   trusts the CA that issued their own certificate(s). Thus, there is
   no 'Root CA' for the entire PKI. The CAs in this environment have
   peer relationships; they are neither superior nor subordinate to one
   another. In a mesh, CAs in the PKI cross-certify. That is, each CA
   issues a certificate to, and is issued a certificate by, peer CAs in
   the PKI. The figure depicts a mesh PKI that is fully cross-certified
   (sometimes called a full mesh). However, it is possible to architect
   and deploy a mesh PKI with a mixture of uni-directional and bi-
   directional cross-certifications (called a partial mesh). Partial
   meshes may also include CAs that are not cross-certified with other
   CAs in the mesh.

```
                  +--------------------------------+
                  |                                |
        +-----------+---------------------+        |
        |           v                     v        |
        |        +-------+            +------+      |
        | +--->| CA B  |<------------>| CA C |<--+  |
        | |      +-------+            +------+   |  |
        | |        |      ^            ^     |   |  |
        | |        v      |            |     |   |  |
        | |      +----+   |            |     |   |  |
        | |      | EE |   +----+    +--------+  v   |  |
        | |      +----+        |    |         +----+ |  |
        | |                    |    |         | EE | |  |
        | |                    |    |         +----+ v  v
        v v                    v    v                
      +------+            +------+            +------+
      | CA E |<---------->| CA A |<---------->| CA D |
      +------+            +------+            +------+
        | ^  ^                                 ^ ^  |
        | |  |                                 | |  |
        v |  +--------------------------------+ | v
      +----+ |                                | +----+
      | EE | |             +------+           | | EE |
      +----+ +-------------| CA F |-----------+ +----+
                           +------+
```
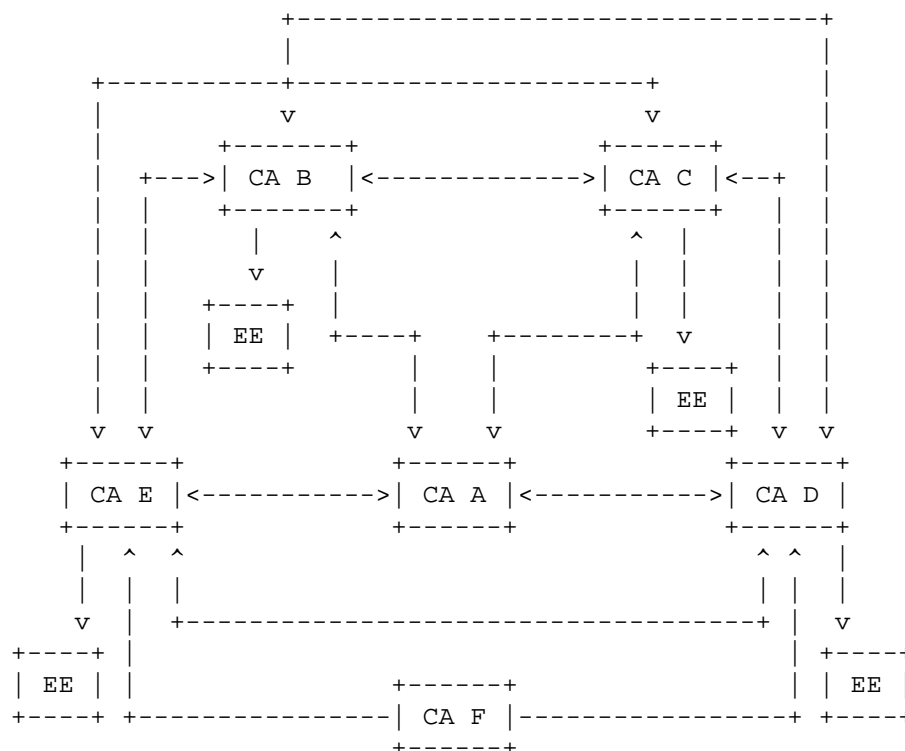
                    Figure 3 - Mesh PKI

   Certification path building in a mesh PKI is more complex than in a
   hierarchical PKI due to the likely existence of multiple paths
   between a relying party's trust anchor and the certificate to be
   verified. These multiple paths increase the potential for creating
   "loops", "dead ends", or invalid paths while building the
   certification path between a trust anchor and a target certificate.
   In addition, in cases where no valid path exists, the total number of
   paths traversed by the path-building software in order to conclude
   "no path exists" can grow exceedingly large. For example, if
   ignoring everything except the structure of the graph, the Mesh PKI
   figure above has 22 non-self issued CA certificates and a total of
   5,092,429 certification paths between CA F and the EE issued by CA D
   without repeating any certificates.

1.5.3. Bi-Lateral Cross-Certified Structures

   PKIs can be connected via cross-certification to enable the relying
   parties of each to verify and accept certificates issued by the other
   PKI. If the PKIs are hierarchical, cross-certification will
   typically be accomplished by each Root CA issuing a certificate for
   the other PKI's Root CA. This results in a slightly more complex,

but still essentially hierarchical environment. If the PKIs are mesh
style, then a CA within each PKI is selected, more or less
arbitrarily, to establish the cross-certification, effectively
creating a larger mesh PKI. Figure 4 depicts a hybrid situation
resulting from a hierarchical PKI cross-certifying with a mesh PKI.

```
                    PKI 1 and 2 cross-certificates
            +-------------------------------+
            |                               |
            |                               v
            |                         +---------+
            |                 +----| Root CA |---+
            |                 |    +---------+   |
            |                 |       PKI 1      |
            |                 v                  v
            |              +------+           +------+
            v PKI 2       +-|  CA  |-+         |  CA  |
         +------+         | +------+ |         +------+
    +------->|  CA  |<-----+         |   |    |   |   |   |
    |        +------+         |      |   |    |   |   |   |
    |         |    |          |      |   |    |   |   |   |
    |         |    |          |      v   v    v   v   v
    |         |    |          |   +----+ +----+ +----+ +----+ +----+
    |         v    v          |   | EE | | EE | | EE | | EE | | EE |
    |       +----+ +----+     |   +----+ +----+ +----+ +----+ +----+
    |       | EE | | EE |     |
    |       +----+ +----+     |
    |                         |
    v                         v
 +------+                  +------+
 |  CA  |<--------------->|  CA  |------+
 +------+                  +------+      |
  |    |                    |    |       |
  |    |                    |    |       |
  v    v                    v    v       v
+----+ +----+            +----+ +----+ +----+
| EE | | EE |            | EE | | EE | | EE |
+----+ +----+            +----+ +----+ +----+
```
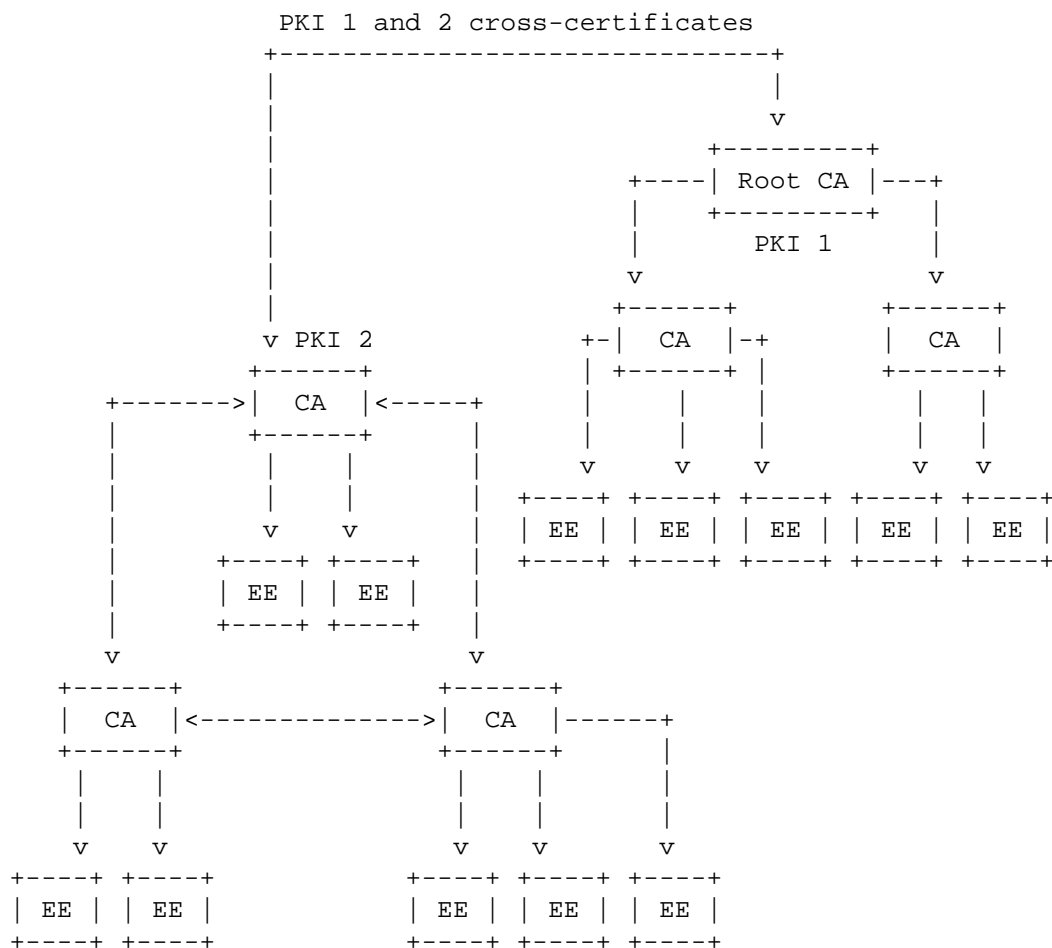
                    Figure 4 - Hybrid PKI

   In current implementations, this situation creates a concern that the
   applications used under the hierarchical PKIs will not have path
   building capabilities robust enough to handle this more complex
   certificate graph. As the number of cross-certified PKIs grows, the
   number of the relationships between them grows exponentially. Two
   principal concerns about cross-certification are the creation of
   unintended certification paths through transitive trust, and the
   dilution of assurance when a high-assurance PKI with restrictive
   operating policies is cross-certified with a PKI with less

restrictive policies. (Proper name constraints and certificate
policies processing can help mitigate the problem of assurance
dilution.)

1.5.4. Bridge Structures

Another approach to the interconnection of PKIs is the use of a
"bridge" certification authority (BCA). A BCA is a nexus to
establish trust paths among multiple PKIs. The BCA cross-certifies
with one CA in each participating PKI. Each PKI only cross-certifies
with one other CA (i.e., the BCA), and the BCA cross-certifies only
once with each participating PKI. As a result, the number of cross
certified relationships in the bridged environment grows linearly
with the number of PKIs whereas the number of cross-certified
relationships in mesh architectures grows exponentially. However,
when connecting PKIs in this way, the number and variety of PKIs
involved results in a non-hierarchical environment, such as the one
as depicted in Figure 5. (Note: as discussed in Section 2.3, non-
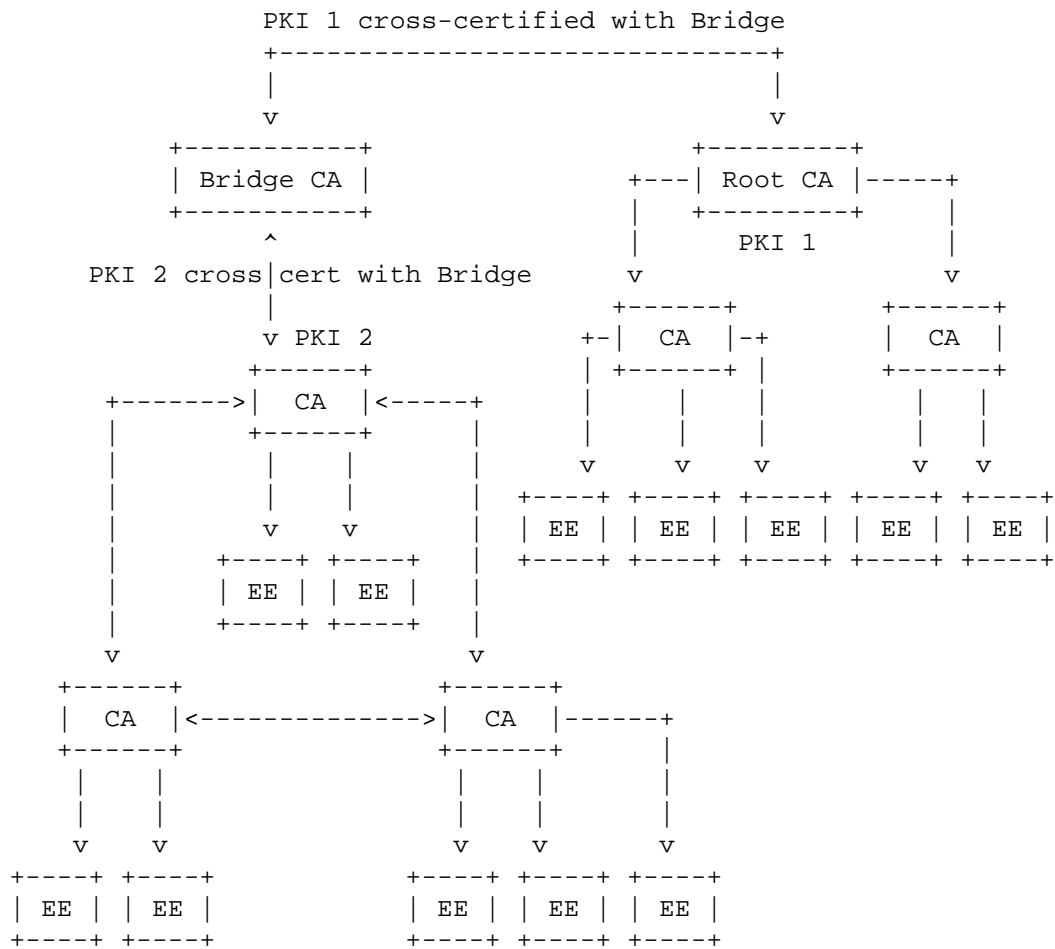hierarchical PKIs can be considered hierarchical, depending upon
perspective.)

```
                  PKI 1 cross-certified with Bridge
                  +-----------------------------+
                  |                             |
                  v                             v
           +-----------+                 +---------+
           | Bridge CA |            +---| Root CA |-----+
           +-----------+            |   +---------+     |
                 ^                  |       PKI 1       |
         PKI 2 cross|cert with Bridge      v                 v
                    |           +------+          +------+
                 v PKI 2        +-|  CA  |-+       |  CA  |
              +------+          | +------+ |       +------+
        +------->|  CA  |<-----+       |   |    |       |    |
        |     +------+          |   |    |       |    |
        |        |    |         |   |    |       |    |
        |        |    |         |   v    v    v       v    v
        |        v    v         | +----+ +----+ +----+ +----+ +----+
        |      +----+ +----+    | | EE | | EE | | EE | | EE | | EE |
        |      | EE | | EE |    | +----+ +----+ +----+ +----+ +----+
        |      +----+ +----+    |
        |                       |
        v                       v
     +------+              +------+
     |  CA  |<------------->|  CA  |------+
     +------+              +------+       |
       |    |               |    |       |
       |    |               |    |       |
       v    v               v    v       v
    +----+ +----+        +----+ +----+ +----+
    | EE | | EE |        | EE | | EE | | EE |
    +----+ +----+        +----+ +----+ +----+
```

              Figure 5 - Cross-Certification with a Bridge CA

1.6. Bridge Structures and Certification Path Processing

   Developers building certificate-enabled applications intended for
   widespread use throughout various sectors are encouraged to consider
   supporting a Bridge PKI structure because implementation of
   certification path processing functions to support a Bridge PKI
   structure requires support of all the PKI structures (e.g.,
   hierarchical, mesh, hybrid) which the Bridge may connect. An
   application that can successfully build valid certification paths in
   all Bridge PKIs will therefore have implemented all of the processing
   logic required to support the less complicated PKI structures. Thus,
   if an application fully supports the Bridge PKI structure, it can be
   deployed in any standards-compliant PKI environment and will perform
   the required certification path processing properly.

2. Certification Path Building

   Certification path building is the process by which the certificate
   processing system obtains the certification path between a trust
   anchor and the target certificate. Different implementations can
   build the certification path in different ways; therefore, it is not
   the intent of this document to recommend a single "best" way to
   perform this function. Rather, guidance is provided on the technical
   issues that surround the path-building process, and on the
   capabilities path-building implementations need in order to build
   certification paths successfully, irrespective of PKI structures.

2.1. Introduction to Certification Path Building

   A certification path is an ordered list of certificates starting with
   a certificate that can be validated by one of the relying party's
   trust anchors, and ending with the certificate to be validated. (The
   certificate to be validated is referred to as the "target
   certificate" throughout this document.) Though not required, as a
   matter of convenience these trust anchors are typically stored in
   trust anchor certificates. The intermediate certificates that
   comprise the certification path may be retrieved by any means
   available to the validating application. These sources may include
   LDAP, HTTP, SQL, a local cache or certificate store, or as part of
   the security protocol itself as is common practice with signed S/MIME
   messages and SSL/TLS sessions.

   Figure 6 shows an example of a certification path. In this figure,
   the horizontal arrows represent certificates, and the notation B(A)
   signifies a certificate issued to B, signed by A.

```
    +---------+       +-----+     +-----+     +-----+     +--------+
    | Trust   |----->| CA  |---->| CA  |---->| CA  |---->| Target |
    | Anchor  | :     | A   | :   | B   | :   | C   | :   | EE     |
    +---------+ :     +-----+ :   +-----+ :   +-----+ :   +--------+
              :             :           :           :
              :             :           :           :
          Cert 1        Cert 2      Cert 3      Cert 4
       A(Trust Anchor)    B(A)        C(B)      Target(C)
```
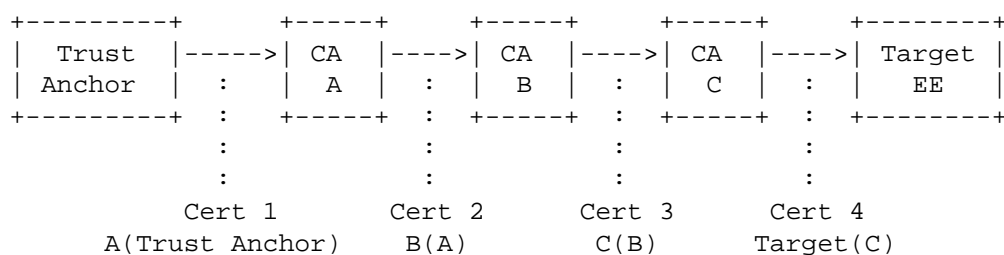
                  Figure 6 - Example Certification Path

   Unlike certification path validation, certification path building is
   not addressed by the standards that define the semantics and
   structure of a PKI. This is because the validation of a
   certification path is unaffected by the method in which the
   certification path was built. However, the ability to build a valid
   certification path is of paramount importance for applications that

rely on a PKI. Without valid certification paths, certificates
cannot be validated according to [RFC3280] and therefore cannot be
trusted. Thus, the ability to build a path is every bit as important
as the ability to validate it properly.

There are many issues that can complicate the path-building process.
For example, building a path through a cross-certified environment
could require the path-building module to traverse multiple PKI
domains spanning multiple directories, using multiple algorithms, and
employing varying key lengths. A path-building client may also need
to manage a number of trust anchors, partially populated directory
entries (e.g., missing issuedToThisCA entries in the
crossCertificatePair attribute), parsing of certain certificate
extensions (e.g., authorityInformationAccess) and directory
attributes (e.g., crossCertificatePair), and error handling such as
loop detection.

In addition, a developer has to decide whether to build paths from a
trust anchor (the reverse direction) to the target certificate or
from the target certificate (the forward direction) to a trust
anchor. Some implementations may even decide to use both. The
choice a developer makes should be dependent on the environment and
the underlying PKI for that environment. More information on making
this choice can be found in Section 2.3.

2.2. Criteria for Path Building

From this point forward, this document will be discussing specific
algorithms and mechanisms to assist developers of certification
path-building implementations. To provide justification for these
mechanisms, it is important to denote what the authors considered the
criteria for a path-building implementation.

Criterion 1: The implementation is able to find all possible paths,
excepting paths containing repeated subject name/public key pairs.
This means that all potentially valid certification paths between the
trust anchor and the target certificate which may be valid paths can
be built by the algorithm. As discussed in Section 2.4.2, we
recommend that subject names and public key pairs are not repeated in
paths.

Criterion 2: The implementation is as efficient as possible. An
efficient certification path-building implementation is defined to be
one that builds paths that are more likely to validate following
[RFC3280], before building paths that are not likely to validate,
with the understanding that there is no way to account for all
possible configurations and infrastructures. This criterion is
intended to ensure implementations that can produce useful error

   information. If a particular path is entirely valid except for a
   single expired certificate, this is most likely the 'right' path. If
   other paths are developed that are invalid for multiple obscure
   reasons, this provides little useful information.

   The algorithms and mechanisms discussed henceforth are chosen because
   the authors consider them to be good methods for meeting the above
   criteria.

2.3. Path-Building Algorithms

   It is intuitive for people familiar with the Bridge CA concept or
   mesh type PKIs to view path building as traversing a complex graph.
   However, from the simplest viewpoint, writing a path-building module
   can be nothing more than traversal of a spanning tree, even in a very
   complex cross-certified environment. Complex environments as well as
   hierarchical PKIs can be represented as trees because certificates
   are not permitted to repeat in a path. If certificates could be
   repeated, loops can be formed such that the number of paths and
   number of certificates in a path both increase without bound (e.g., A
   issues to B, B issues to C, and C issues to A). Figure 7 below
   illustrates this concept from the trust anchor's perspective.
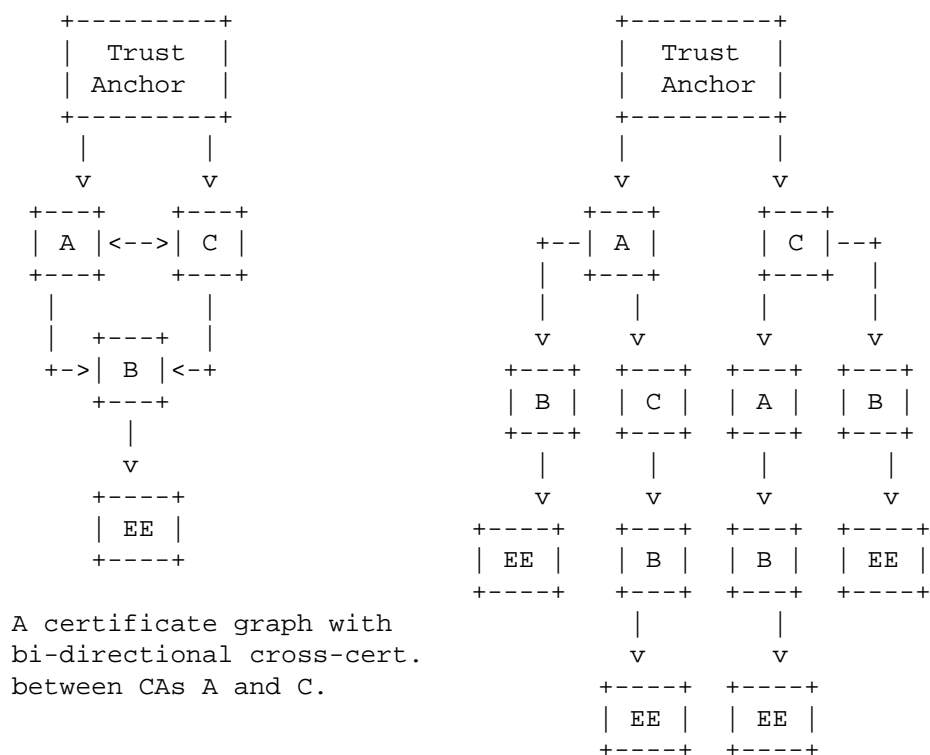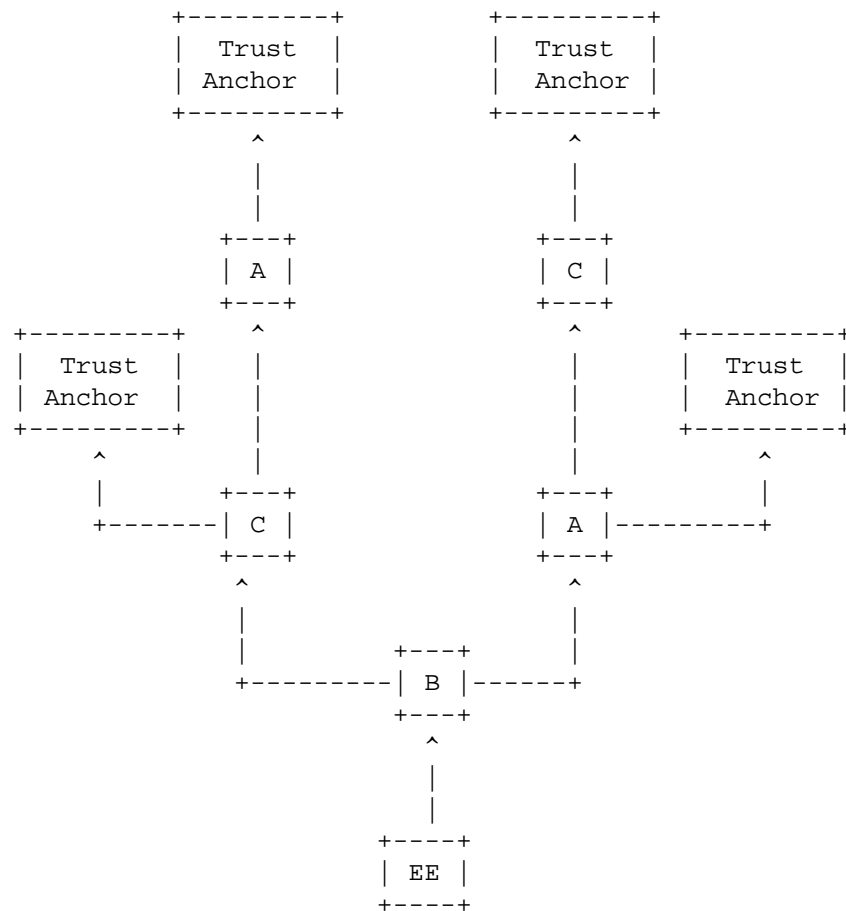
```
        +---------+                      +---------+
        |  Trust  |                      |  Trust  |
        | Anchor  |                      | Anchor  |
        +---------+                      +---------+
         |     |                          |      |
         v     v                          v      v
       +---+   +---+                     +---+     +---+
       | A |<-->| C |                +--| A |     | C |--+
       +---+   +---+                 |  +---+     +---+  |
        |       |                    |   |          |   |
        |  +---+ |                   |   |          |   |
        +->| B |<-+                  v   v          v   v
           +---+                   +---+ +---+   +---+ +---+
            |                      | B | | C |   | A | | B |
            v                      +---+ +---+   +---+ +---+
          +----+                    |     |       |     |
          | EE |                    v     v       v     v
          +----+                  +----+ +---+   +---+ +----+
                                  | EE | | B |   | B | | EE |
      A certificate graph with    +----+ +---+   +---+ +----+
      bi-directional cross-cert.    |       |
      between CAs A and C.          v       v
                                  +----+  +----+
                                  | EE |  | EE |
                                  +----+  +----+


                                 The same certificate graph
                                 rendered as a tree - the
                                 way path-building software
                                 could see it.
```

   Figure 7 - Simple Certificate Graph - From Anchor Tree Depiction

   When viewed from this perspective, all PKIs look like hierarchies
   emanating from the trust anchor. An infrastructure can be depicted
   in this way regardless of its complexity. In Figure 8, the same
   graph is depicted from the end entity (EE) (the target certificate in
   this example). It would appear this way if building in the forward
   (from EE or from target) direction. In this example, without knowing
   any particulars of the certificates, it appears at first that
   building from EE has a smaller decision tree than building from the
   trust anchor. While it is true that there are fewer nodes in the
   tree, it is not necessarily more efficient in this example.

```
              +---------+              +---------+
              |  Trust  |              |  Trust  |
              | Anchor  |              | Anchor  |
              +---------+              +---------+
                   ^                        ^
                   |                        |
                   |                        |
                 +---+                    +---+
                 | A |                    | C |
                 +---+                    +---+
   +---------+     ^                        ^      +---------+
   |  Trust  |     |                        |      |  Trust  |
   | Anchor  |     |                        |      | Anchor  |
   +---------+     |                        |      +---------+
        ^          |                        |           ^
        |        +---+                    +---+          |
        +-------| C |                    | A |---------+
                 +---+                    +---+
                   ^                        ^
                   |                        |
                   |          +---+         |
                   +---------| B |------+
                              +---+
                                ^
                                |
                                |
                              +----+
                              | EE |
                              +----+
```

               The same certificate graph rendered
                as a tree but from the end entity
                  rather than the trust anchor.

      Figure 8 - Certificate Graph - From Target Certificate Depiction

   Suppose a path-building algorithm performed no optimizations. That
   is, the algorithm is only capable of detecting that the current
   certificate in the tree was issued by the trust anchor, or that it
   issued the target certificate (EE). From the tree above, building
   from the target certificate will require going through two
   intermediate certificates before encountering a certificate issued by
   the trust anchor 100% of the time (e.g., EE chains to B, which then
   chains to C, which is issued by the Trust Anchor). The path-building
   module would not chain C to A because it can recognize that C has a
   certificate issued by the Trust Anchor (TA).

On the other hand, in the first tree (Figure 7: from anchor
depiction), there is a 50% probability of building a path longer than
needed (e.g., TA to A to C to B to EE rather than the shorter TA to A
to B to EE). However, even given our simplistic example, the path-
building software, when at A, could be designed to recognize that B's
subject distinguished name (DN) matches the issuer DN of the EE.
Given this one optimization, the builder could prefer B to C. (B's
subject DN matches that of the EE's issuer whereas C's subject DN
does not.) So, for this example, assuming the issuedByThisCA
(reverse) and issuedToThisCA (forward) elements were fully populated
in the directory and our path-building module implemented the
aforementioned DN matching optimization method, path building from
either the trust anchor or the target certificate could be made
roughly equivalent. A list of possible optimization methods is
provided later in this document.

A more complicated example is created when the path-building software
encounters a situation when there are multiple certificates from
which to choose while building a path. We refer to this as a large
decision tree, or a situation with high fan-out. This might occur if
an implementation has multiple trust anchors to choose from, and is
building in the reverse (from trust anchor) direction. Or, it may
occur in either direction if a Bridge CA is encountered. Large
decision trees are the enemy of efficient path-building software. To
combat this problem, implementations should make careful decisions
about the path-building direction, and should utilize optimizations
such as those discussed in Section 3.1 when confronted with a large
decision tree.

Irrespective of the path-building approach for any path-building
algorithm, cases can be constructed that make the algorithm perform
poorly. The following questions should help a developer decide from
which direction to build certification paths for their application:

1) What is required to accommodate the local PKI environment and the
   PKI environments with which interoperability will be required?

   a. If using a directory, is the directory [RFC2587] compliant
      (specifically, are the issuedToThisCA [forward] cross-
      certificates and/or the cACertificate attributes fully
      populated in the directory)? If yes, you are able to build in
      the forward direction.

   b. If using a directory, does the directory contain all the
      issuedByThisCA (reverse) cross-certificates in the
      crossCertificatePair attribute, or, alternately, are all
      certificates issued from each CA available via some other
      means? If yes, it is possible to build in the reverse

direction. Note: [RFC2587] does not require the issuedByThisCA
(reverse) cross-certificates to be populated; if they are
absent it will not be possible to build solely in the reverse
direction.

c. Are all issuer certificates available via some means other than
   a directory (e.g., the authorityInformationAccess extension is
   present and populated in all certificates)? If yes, you are
   able to build in the forward direction.

2) How many trust anchors will the path-building and validation
   software be using?

   a. Are there (or will there be) multiple trust anchors in the
      local PKI? If yes, forward path building may offer better
      performance.

   b. Will the path-building and validation software need to place
      trust in trust anchors from PKIs that do not populate reverse
      cross-certificates for all intermediate CAs? If no, and the
      local PKI populates reverse cross-certificates, reverse path
      building is an option.

2.4. How to Build a Certification Path

   As was discussed in the prior section, path building is essentially a
   tree traversal. It was easy to see how this is true in a simple
   example, but how about a more complicated one? Before taking a look
   at more a complicated scenario, it is worthwhile to address loops and
   what constitutes a loop in a certification path. [X.509] specifies
   that the same certificate may not repeat in a path. In a strict
   sense, this works well as it is not possible to create an endless
   loop without repeating one or more certificates in the path.
   However, this requirement fails to adequately address Bridged PKI
   environments.
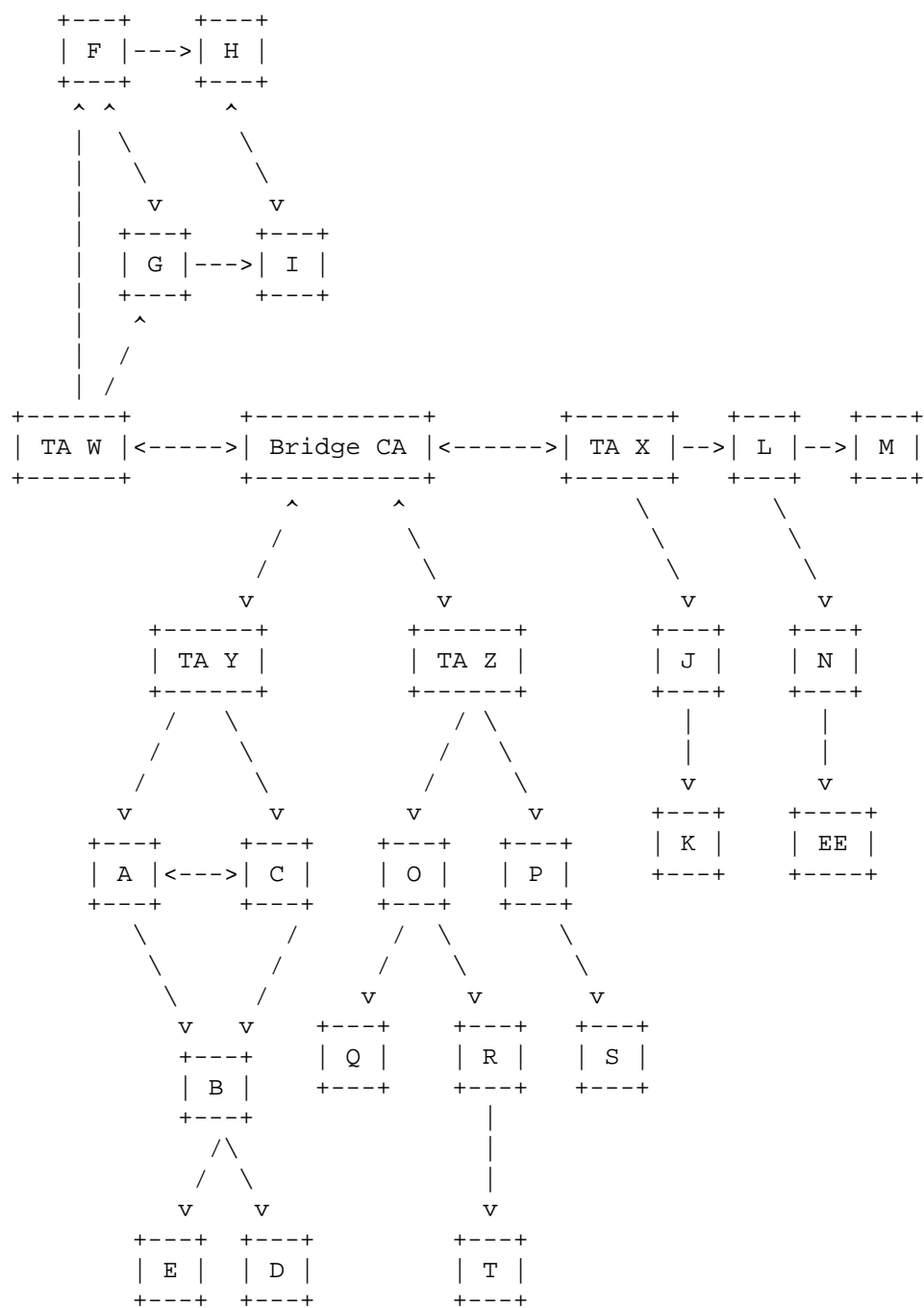
```
        +---+     +---+
        | F |--->| H |
        +---+     +---+
         ^ ^       ^
         | \        \
         |  \        \
         |   v        v
         |  +---+     +---+
         |  | G |--->| I |
         |  +---+     +---+
         |   ^
         |  /
         | /
    +------+      +-----------+        +------+    +---+    +---+
    | TA W |<----->| Bridge CA |<------>| TA X |-->| L |-->| M |
    +------+      +-----------+        +------+    +---+    +---+
                   ^       ^             \          \
                  /         \             \          \
                 /           \             \          \
                v             v             v          v
            +------+      +------+        +---+      +---+
            | TA Y |      | TA Z |        | J |      | N |
            +------+      +------+        +---+      +---+
             / \           / \            |          |
            /   \         /   \           |          |
           /     \       /     \          v          v
          v       v     v       v       +---+      +----+
        +---+   +---+  +---+   +---+     | K |      | EE |
        | A |<-->| C |  | O |   | P |     +---+      +----+
        +---+   +---+  +---+   +---+
          \     /      / \       \
           \   /      /   \       \
            \ /      v     v       v
           v v    +---+  +---+   +---+
           +---+  | Q |  | R |   | S |
           | B |  +---+  +---+   +---+
           +---+                  |
            /\                    |
           /  \                   |
          v    v                  v
        +---+  +---+            +---+
        | E |  | D |            | T |
        +---+  +---+            +---+


                  Figure 9 - Four Bridged PKIs
```

   Figure 9 depicts four root certification authorities cross-certified
   with a Bridge CA (BCA). While multiple trust anchors are shown in
   the Figure, our examples all consider TA Z as the trust anchor. The
   other trust anchors serve different relying parties. By building
   certification paths through the BCA, trust can be extended across the
   four infrastructures. In Figure 9, the BCA has four certificates
   issued to it; one issued from each of the trust anchors in the graph.
   If stored in the BCA directory system, the four certificates issued
   to the BCA would be stored in the issuedToThisCA (forward) entry of
   four different crossCertificatePair structures. The BCA also has
   issued four certificates, one to each of the trust anchors. If
   stored in the BCA directory system, those certificates would be
   stored in the issuedByThisCA (reverse) entry of the same four
   crossCertificatePair structures. (Note that the cross-certificates
   are stored as matched pairs in the crossCertificatePair attribute.
   For example, a crossCertificatePair structure might contain both A(B)
   and B(A), but not contain A(C) and B(A).) The four
   crossCertificatePair structures would then be stored in the BCA's
   directory entry in the crossCertificatePair attribute.

2.4.1. Certificate Repetition

   [X.509] requires that certificates are not repeated when building
   paths. For instance, from the figure above, do not build the path TA
   Z->BCA->Y->A->C->A->C->B->D. Not only is the repetition unnecessary
   to build the path from Z to D, but it also requires the reuse of a
   certificate (the one issued from C to A), which makes the path non-
   compliant with [X.509].

   What about the following path from TA Z to EE?

              TA Z->BCA->Y->BCA->W->BCA->X->L->N->EE

   Unlike the first example, this path does not require a developer to
   repeat any certificates; therefore, it is compliant with [X.509].
   Each of the BCA certificates is issued from a different source and is
   therefore a different certificate. Suppose now that the bottom left
   PKI (in Figure 9) had double arrows between Y and C, as well as
   between Y and A. The following path could then be built:

              TA Z->BCA->Y->A->C->Y->BCA->W->BCA->X->L->N->EE

   A path such as this could become arbitrarily complex and traverse
   every cross-certified CA in every PKI in a cross-certified
   environment while still remaining compliant with [X.509]. As a
   practical matter, the path above is not something an application
   would typically want or need to build for a variety of reasons:

     - First, certification paths like the example above are generally
       not intended by the PKI designers and should not be necessary in
       order to validate any given certificate. If a convoluted path
       such as the example above is required (there is no corresponding
       simple path) in order to validate a given certificate, this is
       most likely indicative of a flaw in the PKI design.

     - Second, the longer a path becomes, the greater the potential
       dilution of trust in the certification path. That is, with each
       successive link in the infrastructure (i.e., certification by
       CAs and cross-certification between CAs) some amount of
       assurance may be considered lost.

     - Third, the longer and more complicated a path, the less likely
       it is to validate because of basic constraints, policies or
       policy constraints, name constraints, CRL availability, or even
       revocation.

     - Lastly, and certainly not least important from a developer's or
       user's perspective, is performance. Allowing paths like the one
       above dramatically increases the number of possible paths for
       every certificate in a mesh or cross-certified environment.
       Every path built may require one or more of the following:
       validation of certificate properties, CPU intensive signature
       validations, CRL retrievals, increased network load, and local
       memory caching. Eliminating the superfluous paths can greatly
       improve performance, especially in the case where no path
       exists.

   There is a special case involving certificates with the same
   distinguished names but differing encodings required by [RFC3280].
   This case should not be considered a repeated certificate. See
   Section 5.4 for more information.

2.4.2. Introduction to Path-Building Optimization

   How can these superfluous paths be eliminated? Rather than only
   disallowing identical certificates from repeating, it is recommended
   that a developer disallow the same public key and subject name pair
   from being repeated. For maximum flexibility, the subject name
   should collectively include any subject alternative names. Using
   this approach, all of the intended and needed paths should be
   available, and the excess and diluted paths should be eliminated.
   For example, using this approach, only one path exists from the TA Z
   to EE in the diagram above: TA Z->BCA->X->L->N->EE.

Given the simplifying rule of not repeating pairs of subject names
(including subject alternative names) and public keys, and only using
certificates found in the cACertificate and forward (issuedToThisCA)
element of the crossCertificatePair attributes, Figure 10 depicts the
forward path-building decision tree from the EE to all reachable
nodes in the graph. This is the ideal graph for a path builder
attempting to build a path from TA Z to EE.

```
     +------+          +-----------+          +------+   +---+
     | TA W |<------| Bridge CA |<-------| TA X |<--| L |
     +------+          +-----------+          +------+   +---+
                        /      \                          ^
                       /        \                          \
                      /          \                          \
                     v            v                          \
                  +------+       +------+                  +---+
                  | TA Y |       | TA Z |                  | N |
                  +------+       +------+                  +---+
                                                            ^
                                                             \
                                                              \
                                                          +----+
                                                          | EE |
                                                          +----+
```
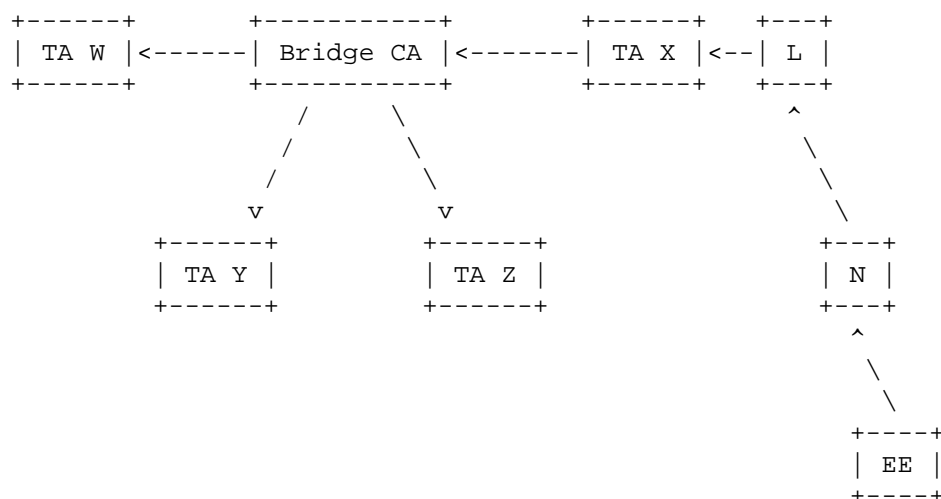
              Figure 10 - Forward (From Entity) Decision Tree

It is not possible to build forward direction paths into the
infrastructures behind CAs W, Y, and Z, because W, Y, and Z have not
been issued certificates by their subordinate CAs. (The subordinate
CAs are F and G, A and C, and O and P, respectively.) If simplicity
and speed are desirable, the graph in Figure 10 is a very appealing
way to structure the path-building algorithm. Finding a path from
the EE to one of the four trust anchors is reasonably simple.
Alternately, a developer could choose to build in the opposite
direction, using the reverse cross-certificates from any one of the
four trust anchors around the BCA. The graph in Figure 11 depicts
all possible paths as a tree emanating from TA Z. (Note: it is not
recommended that implementations attempt to determine all possible
paths, this would require retrieval and storage of all PKI data
including certificates and CRLs! This example is provided to
demonstrate the complexity which might be encountered.)

```
    +---+     +---+
    | I |--->| H |
    +---+     +---+
      ^
      |        +---+     +---+
      |        | H |--->| I |
      |        +---+     +---+
    +---+      ^
    | G |     /        +---+     +---+     +---+
    +---+    /         | F |--->| H |--->| I |
      ^     /          +---+     +---+     +---+
       \   /            ^
        \ /            /
         \/           /
      +---+     +---+     +---+     +---+               +---+
      | F |     | G |--->| I |--->| H |               | M |
      +---+     +---+     +---+     +---+               +---+
        ^         ^                                      ^
        |        /                                       |
      +------+         +-----------+         +------+     +---+
      | TA W |<------| Bridge CA |-------->| TA X |-->| L |
      +------+         +-----------+         +------+     +---+
             /              ^                     \         \
            v                \                     v         v
         +------+         +------+         +---+     +---+
         | TA Y |         | TA Z |         | J |     | N |
         +------+         +------+         +---+     +---+
         /      \         /      \           \         \
        v        v       v        v           v         v
     +---+     +---+   +---+     +---+     +---+     +----+
     | A |     | C |   | O |     | P |     | K |     | EE |
     +---+     +---+   +---+     +---+     +---+     +----+
     /   \     /   \   /   \       \
    v     v   v     v v     v       v
  +---+ +---+ +---+ +---+ +---+ +---+     +---+
  | B | | C | | A | | B | | Q | | R |     | S |
  +---+ +---+ +---+ +---+ +---+ +---+     +---+
  /   \     \     \     \     \     \
 v     v     v     v     v     v     v
+---+ +---+ +---+ +---+ +---+ +---+ +---+
| E | | D | | B | | B | | E | | D | | T |
+---+ +---+ +---+ +---+ +---+ +---+ +---+
           /   |     |   \
          v    v     v    v
        +---+ +---+ +---+ +---+
        | E | | D | | E | | D |
        +---+ +---+ +---+ +---+
```
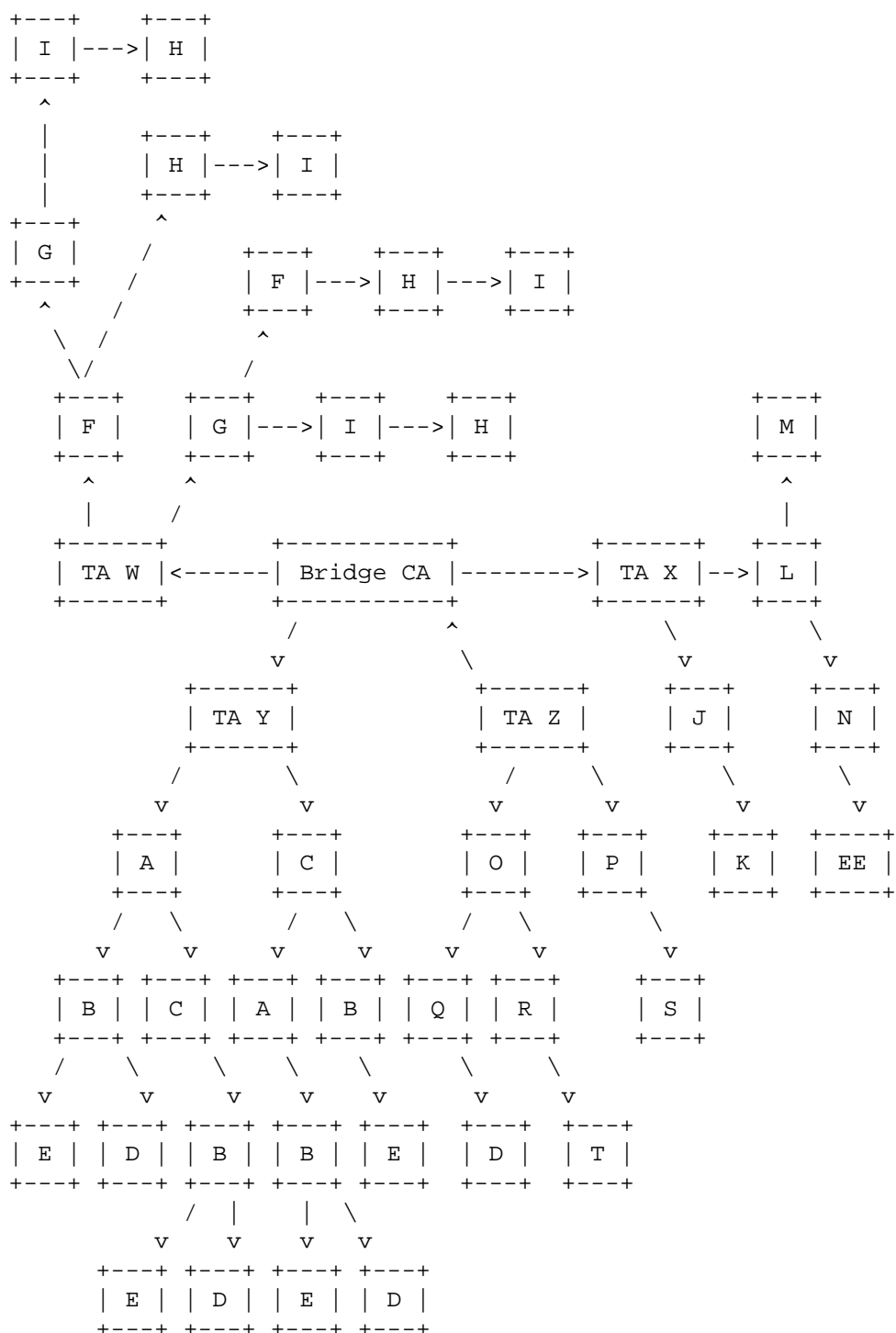
            Figure 11 - Reverse (From Anchor) Decision Tree

Given the relative complexity of this decision tree, it becomes clear
that making the right choices while navigating the tree can make a
large difference in how quickly a valid path is returned. The path-
building software could potentially traverse the entire graph before
choosing the shortest path: TA Z->BCA->X->L->N->EE. With a decision
tree like the one above, the basic depth first traversal approach
introduces obvious inefficiencies in the path-building process. To
compensate for this, a path-building module needs to decide not only
in which direction to traverse the tree, but also which branches of
the tree are more likely to yield a valid path.

The path-building algorithm then ideally becomes a tree traversal
algorithm with weights or priorities assigned to each branch point to
guide the decision making. If properly designed, such an approach
would effectively yield the "best path first" more often than not.
(The terminology "best path first" is quoted because the definition
of the "best" path may differ from PKI to PKI. That is ultimately to
be determined by the developer, not by this document.) Finding the
"best path first" is an effort to make the implementation efficient,
which is one of our criteria as stated in Section 2.2.

So how would a developer go about finding the best path first? Given
the simplifying idea of addressing path building as a tree traversal,
path building could be structured as a depth first search. A simple
example of depth first tree traversal path building is depicted in
Figure 12, with no preference given to sort order.

Note: The arrows in the lower portion of the figure do not indicate
the direction of certificate issuance; they indicate the direction of
the tree traversal from the target certificate (EE).
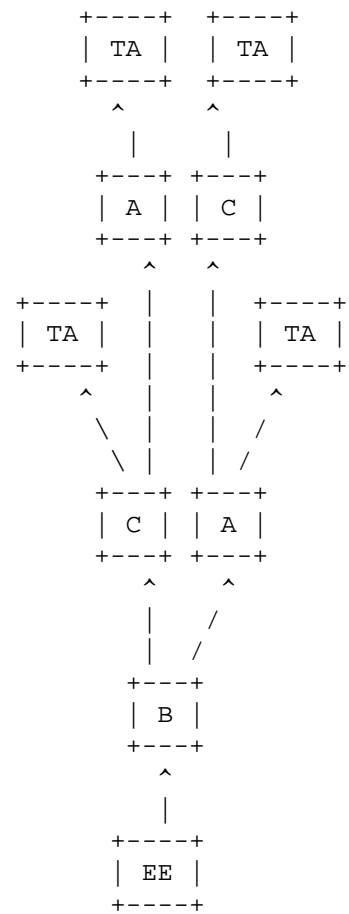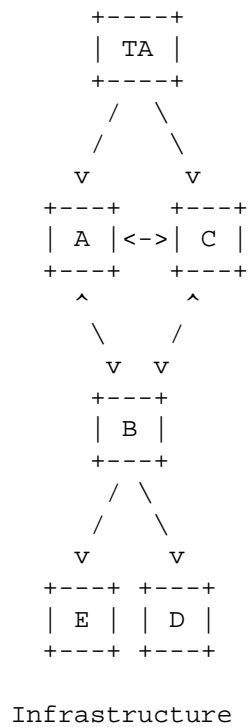
```
            +----+                        +----+  +----+
            | TA |                        | TA |  | TA |
            +----+                        +----+  +----+
             / \                            ^       ^
            /   \                           |       |
           v     v                        +---+ +---+
         +---+   +---+                     | A | | C |
         | A |<->| C |                     +---+ +---+
         +---+   +---+                       ^     ^
          ^       ^               +----+     |     |    +----+
           \     /                | TA |     |     |    | TA |
            v   v                 +----+     |     |    +----+
           +---+                   ^         |     |     ^
           | B |                    \        |     |    /
           +---+                     \       |     |   /
           / \                        \      |     |  /
          /   \                      +---+ +---+
         v     v                     | C | | A |
       +---+ +---+                    +---+ +---+
       | E | | D |                      ^     ^
       +---+ +---+                       |    /
                                         |   /
          Infrastructure                 |  /
                                        +---+
                                        | B |
                                        +---+
                                          ^
                                          |
                                        +----+
                                        | EE |
                                        +----+


                                   The Same Infrastructure
                                   Represented as a Tree
```

```
             +----+                  +----+
             | TA |                  | TA |
             +----+                  +----+
               ^                       ^
               |                       |
             +---+                   +---+
             | A |                   | C |
             +---+                   +---+
  +----+       ^                       ^        +----+
  | TA |       |                       |        | TA |
  +----+       |                       |        +----+
    ^          |                       |          ^
     \         |                       |         /
   +---+     +---+                   +---+     +---+
   | C |     | C |                   | A |     | A |
   +---+     +---+                   +---+     +---+
     ^         ^                       ^         ^
     |         |                      /         /
     |         |                     /         /
   +---+     +---+                 +---+     +---+
   | B |     | B |                 | B |     | B |
   +---+     +---+                 +---+     +---+
     ^         ^                     ^         ^
     |         |                     |         |
     |         |                     |         |
   +----+    +----+                +----+    +----+
   | EE |    | EE |                | EE |    | EE |
   +----+    +----+                +----+    +----+
```
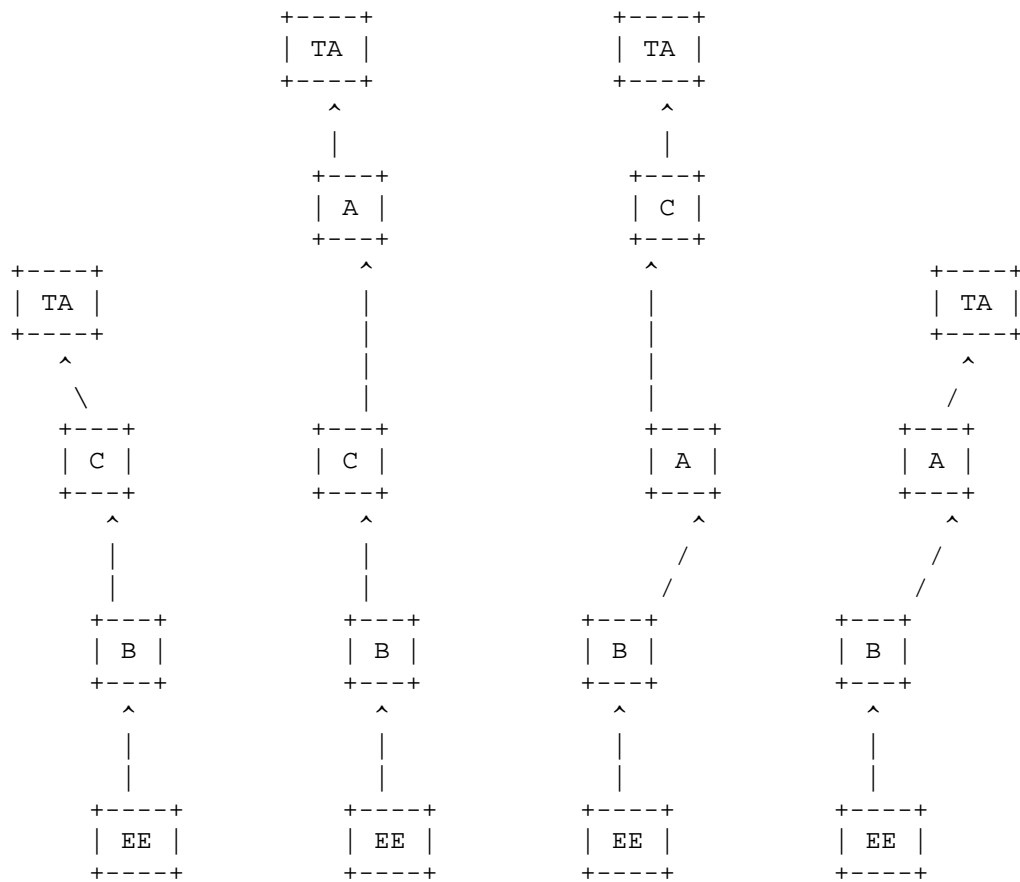
                  All possible paths from EE to TA
                using a depth first decision tree traversal

        Figure 12 - Path Building Using a Depth First Tree Traversal

   Figure 12 illustrates that four possible paths exist for this
   example. Suppose that the last path (TA->A->B->EE) is the only path
   that will validate. This could be for any combination of reasons
   such as name constraints, policy processing, validity periods, or
   path length constraints. The goal of an efficient path-building
   component is to select the fourth path first by testing properties of
   the certificates as the tree is traversed. For example, when the
   path-building software is at entity B in the graph, it should examine
   both choices A and C to determine which certificate is the most
   likely best choice. An efficient module would conclude that A is the
   more likely correct path. Then, at A, the module compares
   terminating the path at TA, or moving to C. Again, an efficient
   module will make the better choice (TA) and thereby find the "best
   path first".

What if the choice between CA certificates is not binary as it was in
the previous example? What if the path-building software encounters
a branch point with some arbitrary number of CA certificates thereby
creating the same arbitrary number of tree branches? (This would be
typical in a mesh style PKI CA, or at a Bridge CA directory entry, as
each will have multiple certificates issued to itself from other
CAs.) This situation actually does not change the algorithm at all,
if it is structured properly. In our example, rather than treating
each decision as binary (i.e., choosing A or C), the path-building
software should sort all the available possibilities at any given
branch point, and then select the best choice from the list. In the
event the path could not be built through the first choice, then the
second choice should be tried next upon traversing back to that point
in the tree. Continue following this pattern until a path is found
or all CA nodes in the tree have been traversed. Note that the
certificates at any given point in the tree should only be sorted at
the time a decision is first made. Specifically, in the example, the
sorting of A and C is done when the algorithm reached B. There is no
memory resident representation of the entire tree. Just like any
other recursive depth first search algorithm, the only information
the algorithm needs to keep track of is what nodes (entities) in the
tree lie behind it on the current path, and for each of those nodes,
which arcs (certificates) have already been tried.

2.5. Building Certification Paths for Revocation Signer Certificates

   Special consideration is given to building a certification path for
   the Revocation Signer certificate because it may or may not be the
   same as the Certification Authority certificate. For example, after
   a CA performs a key rollover, the new CA certificate will be the CRL
   Signer certificate, whereas the old CA certificate is the
   Certification Authority certificate for previously issued
   certificates. In the case of indirect CRLs, the CRL Signer
   certificate will contain a different name and key than the
   Certification Authority certificate. In the case of OCSP, the
   Revocation Signer certificate may represent an OCSP Responder that is
   not the same entity as the Certification Authority.

   When the Revocation Signer certificate and the Certification
   Authority certificate are identical, no additional consideration is
   required from a certification path-building standpoint. That is, the
   certification path built (and validated) for the Certification
   Authority certificate can also be used as the certification path for
   the Revocation Signer certificate. In this case, the signature on
   the revocation data (e.g., CRL or OCSP response) is verified using
   the same certificate, and no other certification path building is
   required. An efficient certification path validation algorithm
   should first try all possible CRLs issued by the Certification

Authority to determine if any of the CRLs (a) cover the certificate
in question, (b) are current, and (c) are signed using the same key
used to sign the certificate.

When the Revocation Signer certificate is not identical to the
Certification Authority certificate, a certification path must be
built (and validated) for the Revocation Signer certificate. In
general, the certification path-building software may build the path
as it would for any other certificate. However, this document also
outlines methods in later sections for greatly improving path
building efficiency for Revocation Signer certificate case.

2.6.   Suggested Path-Building Software Components

There is no single way to define an interface to a path-building
module. It is not the intent of this document to prescribe a
particular method or semantic; rather, it is up to the implementer to
decide. There are many ways this could be done. For example, a
path-building module could build every conceivable path and return
the entire list to the caller. Or, the module could build until it
finds just one that validates and then terminate the procedure. Or,
it could build paths in an iterative fashion, depending on validation
outside of the builder and successive calls to the builder to get
more paths until one valid path is found or all possible paths have
been found. All of these are possible approaches, and each of these
may offer different benefits to a particular environment or
application.

Regardless of semantics, a path-building module needs to contain the
following components:

1) The logic for building and traversing the certificate graph.

2) Logic for retrieving the necessary certificates (and CRLs and/or
   other revocation status information if the path is to be
   validated) from the available source(s).

Assuming a more efficient and agile path-building module is desired,
the following is a good starting point and will tie into the
remainder of this document. For a path-building module to take full
advantage of all the suggested optimizations listed in this document,
it will need all of the components listed below.

1) A local certificate and CRL cache.

   a. This may be used by all certificate-using components; it does
      not need to be specific to the path-building software. A local
      cache could be memory resident, stored in an operating system

          or application certificate store, stored in a database, or even
          stored in individual files on the hard disk. While the
          implementation of this cache is beyond the scope of this
          document, some design considerations are listed below.

   2) The logic for building and traversing the certificate graph/tree.

      a. This performs sorting functionality for prioritizing
         certificates (thereby optimizing path building) while
         traversing the tree.

      b. There is no need to build a complete graph prior to commencing
         path building. Since path building can be implemented as a
         depth first tree traversal, the path builder only needs to
         store the current location in the tree along with the points
         traversed to the current location. All completed branches can
         be discarded from memory and future branches are discovered as
         the tree is traversed.

   3) Logic for retrieving the necessary certificates from the available
      certificate source(s):

      a. Local cache.

           i. Be able to retrieve all certificates for an entity by
              subject name, as well as individual certificates by
              issuer and serial number tuple.

          ii. Tracking which directory attribute (including
              issuedToThisCA <forward> and issuedByThisCA <reverse>
              for split crossCertificatePair attributes) each
              certificate was found in may be useful. This allows for
              functionality such as retrieving only forward cross-
              certificates, etc.

         iii. A "freshness" timestamp (cache expiry time) can be used
              to determine when the directory should be searched
              again.

      b. LDAPv3 directory for certificates and CRLs.

           i. Consider supporting multiple directories for general
              queries.

          ii. Consider supporting dynamic LDAP connections for
              retrieving CRLs using an LDAP URI [RFC3986] in the CRL
              distribution point certificate extension.

          iii. Support LDAP referrals. This is typically only a matter
              of activating the appropriate flag in the LDAP API.

     c. HTTP support for CRL distribution points and authority
       information access (AIA) support.

       i. Consider HTTPS support, but be aware that this may create
         an unbounded recursion when the implementation tries to
         build a certification path for the server's certificate if
         this in turn requires an additional HTTPS lookup.

   4) A certification path cache that stores previously validated
     relationships between certificates. This cache should include:

     a. A configurable expiration date for each entry. This date can
       be configured based upon factors such as the expiry of the
       information used to determine the validity of an entry,
       bandwidth, assurance level, storage space, etc.

     b. Support to store previously verified issuer certificate to
       subject certificate relationships.

       i. Since the issuer DN and serial number tuple uniquely
         identifies a certificate, a pair of these tuples (one for
         both the issuer and subject) is an effective method of
         storing this relationship.

     c. Support for storing "known bad" paths and certificates. Once a
       certificate is determined to be invalid, implementations can
       decide not to retry path development and validation.

2.7. Inputs to the Path-Building Module

   [X.509] specifically addresses the list of inputs required for path
   validation but makes no specific suggestions concerning useful inputs
   to path building. However, given that the goal of path building is
   to find certification paths that will validate, it follows that the
   same inputs used for validation could be used to optimize path
   building.

2.7.1. Required Inputs

   Setting aside configuration information such as repository or cache
   locations, the following are required inputs to the certification
   path-building process:

   1) The Target Certificate: The certificate that is to be validated.
     This is one endpoint for the path. (It is also possible to

       provide information used to retrieve a certificate for a target,
       rather than the certificate itself.)

    2) Trust List: This is the other endpoint of the path, and can
       consist of either:

       a. Trusted CA certificates

       b. Trusted keys and DNs; a certificate is not necessarily required

2.7.2. Optional Inputs

    In addition to the inputs listed in Section 2.7.1, the following
    optional inputs can also be useful for optimizing path building.
    However, if the path-building software takes advantage of all of the
    optimization methods described later in this document, all of the
    following optional inputs will be required.

    1) Time (T): The time for which the certificate is to be validated
       (e.g., if validating a historical signature from one year ago, T
       is needed to build a valid path)

       a. If not included as an input, the path-building software should
          always build for T equal to the current system time.

    2) Initial-inhibit-policy-mapping indicator

    3) Initial-require-explicit-policy indicator

    4) Initial-any-policy-inhibit indicator

    5) Initial user acceptable policy set

    6) Error handlers (call backs or virtual classes)

    7) Handlers for custom certificate extensions

    8) Is-revocation-provider indicator

       a. IMPORTANT: When building a certification path for an OCSP
          Responder certificate specified as part of the local
          configuration, this flag should not be set. It is set when
          building a certification path for a CRL Signer certificate or
          for an OCSP Responder Signer certificate discovered using the
          information asserted in an authorityInformationAccess
          certificate extension.

9) The complete certification path for the Certification Authority
   (if Is-revocation-provider is set)

10) Collection of certificates that may be useful in building the
    path

11) Collection of certificate revocation lists and/or other
    revocation data

The last two items are a matter of convenience. Alternately,
certificates and revocation information could be placed in a local
cache accessible to the path-building module prior to attempting to
build a path.

3. Optimizing Path Building

   This section recommends methods for optimizing path-building
   processes.

3.1. Optimized Path Building

   Path building can be optimized by sorting the certificates at every
   decision point (at every node in the tree) and then selecting the
   most promising certificate not yet selected as described in Section
   2.4.2. This process continues until the path terminates. This is
   roughly equivalent to the concept of creating a weighted edge tree,
   where the edges are represented by certificates and nodes represent
   subject DNs. However, unlike the weighted edge graph concept, a
   certification path builder need not have the entire graph available
   in order to function efficiently. In addition, the path builder can
   be stateless with respect to nodes of the graph not present in the
   current path, so the working data set can be relatively small.

   The concept of statelessness with respect to nodes not in the current
   path is instrumental to using the sorting optimizations listed in
   this document. Initially, it may seem that sorting a given group of
   certificates for a CA once and then preserving that sorted order for
   later use would be an efficient way to write the path builder.
   However, maintaining this state can quickly eliminate the efficiency
   that sorting provides. Consider the following diagram:

```
        +---+
        | R |
        +---+
         ^
         /
        v
 +---+        +---+        +---+      +---+     +----+
 | A |<----->| E |<---->| D |--->| Z |--->| EE |
 +---+        +---+        +---+      +---+     +----+
   ^           ^  ^          ^
    \         / \         /
     \       /   \       /
      v     v     v     v
      +---+        +---+
      | B |<----->| C |
      +---+        +---+
```
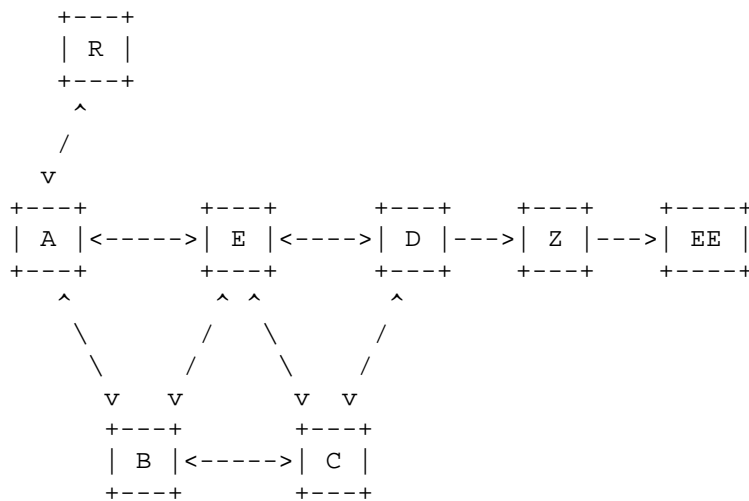
               Figure 13 - Example of Path-Building Optimization

   In this example, the path builder is building in the forward (from
   target) direction for a path between R and EE. The path builder has
   also opted to allow subject name and key to repeat. (This will allow
   multiple traversals through any of the cross-certified CAs, creating
   enough complexity in this small example to illustrate proper state
   maintenance. Note that a similarly complex example could be designed
   by using multiple keys for each entity and prohibiting repetition.)

   The first step is simple; the builder builds the path Z(D)->EE(Z).
   Next the builder adds D and faces a decision between two
   certificates. (Choose between D(C) or D(E)). The builder now sorts
   the two choices in order of priority. The sorting is partially based
   upon what is currently in the path.

   Suppose the order the builder selects is [D(E), D(C)]. The current
   path is now D(E)->Z(D)->EE(Z). Currently the builder has three nodes
   in the graph (EE, Z, and D) and should maintain the state, including
   sort order of the certificates at D, when adding the next node, E.
   When E is added, the builder now has four certificates to sort: E(A),
   E(B), E(C), and E(D). In this case, the example builder opts for the
   order [E(C), E(B), E(A), E(D)]. The current path is now E(C)->D(E)->
   Z(D)->EE(Z) and the path has four nodes; EE, Z, D, and E.

   Upon adding the fifth node, C, the builder sorts the certificates
   (C(B), C(D), and C(E)) at C, and selects C(E). The path is now
   C(E)->E(C)->D(E)->Z(D)->EE(Z) and the path has five nodes: EE, Z, D,
   E, and C.

Now the builder finds itself back at node E with four certificates.
If the builder were to use the prior sort order from the first
encounter with E, it would have [E(C), E(B), E(A), E(D)]. In the
current path's context, this ordering may be inappropriate. To begin
with, the certificate E(C) is already in the path so it certainly
does not deserve first place.

The best way to handle this situation is for the path builder to
handle this instance of E as a new (sixth) node in the tree. In
other words, there is no state information for this new instance of E
- it is treated just as any other new node. The certificates at the
new node are sorted based upon the current path content and the first
certificate is then selected. For example, the builder may examine
E(B) and note that it contains a name constraint prohibiting "C". At
this point in the decision tree, E(B) could not be added to the path
and produce a valid result since "C" is already in the path. As a
result, the certificate E(B) should placed at the bottom of the
prioritized list.

Alternatively, E(B) could be eliminated from this new node in the
tree. It is very important to see that this certificate is
eliminated only at this node and only for the current path. If path
building fails through C and traverses back up the tree to the first
instance of E, E(B) could still produce a valid path that does not
include C; specifically R->A->B->E->D->Z->EE. Thus the state at any
node should not alter the state of previous or subsequent nodes.
(Except for prioritizing certificates in the subsequent nodes.)

In this example, the builder should also note that E(C) is already in
the path and should make it last or eliminate it from this node since
certificates cannot be repeated in a path.

If the builder eliminates both certificates E(B) and E(C) at this
node, it is now only left to select between E(A) and E(D). Now the
path has six nodes: EE, Z, D, E(1), C, and E(2). E(1) has four
certificates, and E(2) has two, which the builder sorts to yield
[E(A), E(D)]. The current path is now E(A)->C(E)->E(C)->D(E)->
Z(D)->EE(Z). A(R) will be found when the seventh node is added to
the path and the path terminated because one of the trust anchors has
been found.

In the event the first path fails to validate, the path builder will
still have the seven nodes and associated state information to work
with. On the next iteration, the path builder is able to traverse
back up the tree to a working decision point, such as A, and select
the next certificate in the sorted list at A. In this example, that
would be A(B). (A(R) has already been tested.) This would dead end,
and the builder traverse back up to the next decision point, E(2)

where it would try D(E). This process repeats until the traversal
backs all the way up to EE or a valid path is found. If the tree
traversal returns to EE, all possible paths have been exhausted and
the builder can conclude no valid path exists.

This approach of sorting certificates in order to optimize path
building will yield better results than not optimizing the tree
traversal. However, the path-building process can be further
streamlined by eliminating certificates, and entire branches of the
tree as a result, as paths are built.

3.2. Sorting vs. Elimination

Consider a situation when building a path in which three CA
certificates are found for a given target certificate and must be
prioritized. When the certificates are examined, as in the previous
example, one of the three has a name constraint present that will
invalidate the path built thus far. When sorting the three
certificates, that one would certainly go to the back of the line.
However, the path-building software could decide that this condition
eliminates the certificate from consideration at this point in the
graph, thereby reducing the number of certificate choices by 33% at
this point.

NOTE: It is important to understand that the elimination of a
certificate only applies to a single decision point during the tree
traversal. The same certificate may appear again at another point in
the tree; at that point it may or may not be eliminated. The
previous section details an example of this behavior.

Elimination of certificates could potentially eliminate the traversal
of a large, time-consuming infrastructure that will never lead to a
valid path. The question of whether to sort or eliminate is one that
pits the flexibility of the software interface against efficiency.

To be clear, if one eliminates invalid paths as they are built,
returning only likely valid paths, the end result will be an
efficient path-building module. The drawback to this is that unless
the software makes allowances for it, the calling application will
not be able to see what went wrong. The user may only see the
unrevealing error message: "No certification path found."

On the other hand, the path-building module could opt to not rule out
any certification paths. The path-building software could then
return any and all paths it can build from the certificate graph. It
is then up to the validation engine to determine which are valid and
which are invalid. The user or calling application can then have
complete details on why each and every path fails to validate. The

drawback is obviously one of performance, as an application or end
user may wait for an extended period of time while cross-certified
PKIs are navigated in order to build paths that will never validate.

Neither option is a very desirable approach. One option provides
good performance for users, which is beneficial. The other option
though allows administrators to diagnose problems with the PKI,
directory, or software. Below are some recommendations to reach a
middle ground on this issue.

First, developers are strongly encouraged to output detailed log
information from the path-building software. The log should
explicitly indicate every choice the builder makes and why. It
should clearly identify which certificates are found and used at each
step in building the path. If care is taken to produce a useful log,
PKI administrators and help desk personnel will have ample
information to diagnose a problem with the PKI. Ideally, there would
be a mechanism for turning this logging on and off, so that it is not
running all the time. Additionally, it is recommended that the log
contain information so that a developer or tester can recreate the
paths tried by the path-building software, to assist with diagnostics
and testing.

Secondly, it is desirable to return something useful to the user.
The easiest approach is probably to implement a "dual mode" path-
building module. In the first mode [mode 1], the software eliminates
any and all paths that will not validate, making it very efficient.
In the second mode [mode 2], all the sorting methods are still
applied, but no paths are eliminated based upon the sorting methods.
Having this dual mode allows the module to first fail to find a valid
path, but still return one invalid path (assuming one exists) by
switching over to the second mode long enough to generate a single
path. This provides a middle ground -- the software is very fast,
but still returns something that gives the user a more specific error
than "no path found".

Third, it may be useful to not rule out any paths, but instead limit
the number of paths that may be built given a particular input.
Assuming the path-building module is designed to return the "best
path first", the paths most likely to validate would be returned
before this limit is reached. Once the limit is reached the module
can stop building paths, providing a more rapid response to the
caller than one which builds all possible paths.

Ultimately, the developer determines how to handle the trade-off
between efficiency and provision of information. A developer could
choose the middle ground by opting to implement some optimizations as
elimination rules and others as not. A developer could validate

certificate signatures, or even check revocation status while
building the path, and then make decisions based upon the outcome of
those checks as to whether to eliminate the certificate in question.

This document suggests the following approach:

1) While building paths, eliminate any and all certificates that do
   not satisfy all path validation requirements with the following
   exceptions:

   a. Do not check revocation status if it requires a directory
      lookup or network access

   b. Do not check digital signatures (see Section 8.1, General
      Considerations for Building A Certification Path, for
      additional considerations).

   c. Do not check anything that cannot be checked as part of the
      iterative process of traversing the tree.

   d. Create a detailed log, if this feature is enabled.

   e. If a path cannot be found, the path builder shifts to "mode 2"
      and allows the building of a single bad path.

        i. Return the path with a failure indicator, as well as
           error information detailing why the path is bad.

2) If path building succeeds, validate the path in accordance with
   [X.509] and [RFC3280] with the following recommendations:

   a. For a performance boost, do not re-check items already checked
      by the path builder. (Note: if pre-populated paths are supplied
      to the path-building system, the entire path has to be fully
      re-validated.)

   b. If the path validation failed, call the path builder again to
      build another path.

        i. Always store the error information and path from the
           first iteration and return this to the user in the event
           that no valid path is found. Since the path-building
           software was designed to return the "best path first",
           this path should be shown to the user.

As stated above, this document recommends that developers do not
validate digital signatures or check revocation status as part of the
path-building process. This recommendation is based on two

assumptions about PKI and its usage. First, signatures in a working
PKI are usually good. Since signature validation is costly in terms
of processor time, it is better to delay signature checking until a
complete path is found and then check the signatures on each
certificate in the certification path starting with the trust anchor
(see Section 8.1). Second, it is fairly uncommon in typical
application environments to encounter a revoked certificate;
therefore, most certificates validated will not be revoked. As a
result, it is better to delay retrieving CRLs or other revocation
status information until a complete path has been found. This
reduces the probability of retrieving unneeded revocation status
information while building paths.

3.3. Representing the Decision Tree

   There are a multitude of ways to implement certification path
   building and as many ways to represent the decision tree in memory.

   The method described below is an approach that will work well with
   the optimization methods listed later in this document. Although
   this approach is the best the authors of this document have
   implemented, it is by no means the only way to implement it.
   Developers should tailor this approach to their own requirements or
   may find that another approach suits their environment, programming
   language, or programming style.

3.3.1. Node Representation for CA Entities

   A "node" in the certification graph is a collection of CA
   certificates with identical subject DNs. Minimally, for each node,
   in order to fully implement the optimizations to follow, the path-
   building module will need to be able to keep track of the following
   information:

   1. Certificates contained in the node

   2. Sorted order of the certificates

   3. "Current" certificate indicator

   4. The current policy set (It may be split into authority and user
      constrained sets, if desired.)

      - It is suggested that encapsulating the policy set in an object
        with logic for manipulating the set such as performing
        intersections, mappings, etc., will simplify implementation.

   5. Indicators (requireExplicitPolicy, inhibitPolicyMapping,
      anyPolicyInhibit) and corresponding skipCert values

   6. A method for indicating which certificates are eliminated or
      removing them from the node.

      - If nodes are recreated from the cache on demand, it may be
        simpler to remove eliminated certificates from the node.

   7. A "next" indicator that points to the next node in the current
      path

   8. A "previous" indicator that points to the previous node in the
      current path

3.3.2. Using Nodes to Iterate Over All Paths

   In simplest form, a node is created, the certificates are sorted, the
   next subject DN required is determined from the first certificate,
   and a new node is attached to the certification path via the next
   indicator (Number 7 above). This process continues until the path
   terminates. (Note: end entity certificates may not contain subject
   DNs as allowed by [RFC3280]. Since end entity certificates by
   definition do not issue certificates, this has no impact on the
   process.)

   Keeping in mind that the following algorithm is designed to be
   implemented using recursion, consider the example in Figure 12 and
   assume that the only path in the diagram is valid for E is TA->A->
   B->E:

   If our path-building module is building a path in the forward
   direction for E, a node is first created for E. There are no
   certificates to sort because only one certificate exists, so all
   initial values are loaded into the node from E. For example, the
   policy set is extracted from the certificate and stored in the node.

   Next, the issuer DN (B) is read from E, and new node is created for B
   containing both certificates issued to B -- B(A) and B(C). The
   sorting rules are applied to these two certificates and the sorting
   algorithm returns B(C);B(A). This sorted order is stored and the
   current indicator is set to B(C). Indicators are set and the policy
   sets are calculated to the extent possible with respect to B(C). The
   following diagram illustrates the current state with the current
   certificate indicated with a "*".

```
+-------------+    +--------------+
| Node 1      |    | Node 2       |
| Subject: E  |--->| Subject: B   |
| Issuers: B* |    | Issuers: C*,A |
+-------------+    +--------------+
```

Next, a node is created for C and all three certificates are added to it. The sorting algorithm happens to return the certificates sorted in the following order: C(TA);C(A);C(B)

```
+-------------+    +--------------+    +-----------------+
| Node 1      |    | Node 2       |    | Node 3          |
| Subject: E  |--->| Subject: B   |--->| Subject: C      |
| Issuers: B  |    | Issuers: C*,A |    | Issuers: TA*,A,B |
+-------------+    +--------------+    +-----------------+
```

Recognizing that the trust anchor has been found, the path (TA->C->B->E) is validated but fails. (Remember that the only valid path happens to be TA->A->B->E.) The path-building module now moves the current certificate indicator in node 3 to C(A), and adds the node for A.

```
  +-------------+    +--------------+    +-----------------+
  | Node 1      |    | Node 2       |    | Node 3          |
  | Subject: E  |--->| Subject: B   |--->| Subject: C      |
  | Issuers: B  |    | Issuers: C*,A |    | Issuers: TA,A*,B |
  +-------------+    +--------------+    +-----------------+
                                                |
                                                v
                                        +-----------------+
                                        | Node 4          |
                                        | Subject: A      |
                                        | Issuers: TA*,C,B |
                                        +-----------------+
```

The path TA->A->C->B->E is validated and it fails. The path-building module now moves the current indicator in node 4 to A(C) and adds a node for C.

```
+-------------+    +--------------+    +-----------------+
| Node 1      |    | Node 2       |    | Node 3          |
| Subject: E  |--->| Subject: B   |--->| Subject: C      |
| Issuers: B  |    | Issuers: C*,A |    | Issuers: TA,A*,B |
+-------------+    +--------------+    +-----------------+
                                               |
                                               v
                  +-----------------+    +-----------------+
                  | Node 5          |    | Node 4          |
                  | Subject: C      |<---| Subject: A      |
                  | Issuers: TA*,A,B |    | Issuers: TA,C*,B |
                  +-----------------+    +-----------------+
```
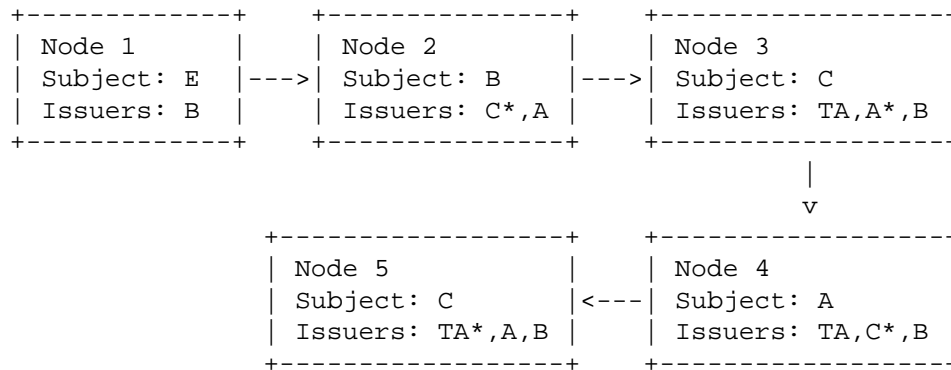
At this juncture, the decision of whether to allow repetition of name
and key comes to the forefront. If the certification path-building
module will NOT allow repetition of name and key, there are no
certificates in node 5 that can be used. (C and the corresponding
public key is already in the path at node 3.) At this point, node 5
is removed from the current path and the current certificate
indicator on node 4 is moved to A(B).

If instead, the module is only disallowing repetition of
certificates, C(A) is eliminated from node 5 since it is in use in
node 3, and path building continues by first validating TA->C->A->
C->B->E, and then continuing to try to build paths through C(B).
After this also fails to provide a valid path, node 5 is removed from
the current path and the current certificate indicator on node 4 is
moved to A(B).

```
+-------------+    +--------------+    +-----------------+
| Node 1      |    | Node 2       |    | Node 3          |
| Subject: E  |--->| Subject: B   |--->| Subject: C      |
| Issuers: B  |    | Issuers: C*,A |    | Issuers: TA,A*,B |
+-------------+    +--------------+    +-----------------+
                                               |
                                               v
                                     +-----------------+
                                     | Node 4          |
                                     | Subject: A      |
                                     | Issuers: TA,C,B* |
                                     +-----------------+
```
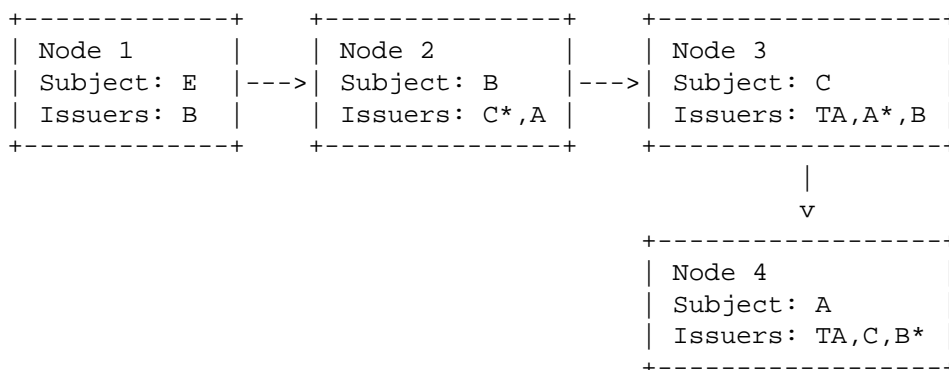
Now a new node 5 is created for B. Just as with the prior node 5, if
not repeating name and key, B also offers no certificates that can be
used (B and B's public key is in use in node 2) so the new node 5 is
also removed from the path. At this point all certificates in node 4
have now been tried, so node 4 is removed from the path, and the
current indicator on node 3 is moved to C(B).

Also as above, if allowing repetition of name and key, B(C) is
removed from the new node 5 (B(C) is already in use in node 3) and
paths attempted through the remaining certificate B(A). After this
fails, it will lead back to removing node 5 from the path. At this
point all certificates in node 4 have now been tried, so node 4 is
removed from the path, and the current indicator on node 3 is moved
to C(B).

This process continues until all certificates in node 1 (if there
happened to be more than one) have been tried, or until a valid path
has been found. Once the process ends and in the event no valid path
was found, it may be concluded that no path can be found from E to
TA.

3.4. Implementing Path-Building Optimization

The following section describes methods that may be used for
optimizing the certification path-building process by sorting
certificates. Optimization as described earlier seeks to prioritize
a list of certificates, effectively prioritizing (weighting) branches
of the graph/tree. The optimization methods can be used to assign a
cumulative score to each certificate. The process of scoring the
certificates amounts to testing each certificate against the
optimization methods a developer chooses to implement, and then
adding the score for each test to a cumulative score for each
certificate. After this is completed for each certificate at a given
branch point in the builder's decision tree, the certificates can be
sorted so that the highest scoring certificate is selected first, the
second highest is selected second, etc.

For example, suppose the path builder has only these two simple
sorting methods:

1) If the certificate has a subject key ID, +5 to score.
2) If the certificate has an authority key ID, +10 to score.

And it then examined three certificates:

1) Issued by CA 1; has authority key ID; score is 10.
2) Issued by CA 2; has subject key ID; score is 5.
3) Issued by CA 1; has subject key ID and authority key ID; score is
   15.

The three certificates are sorted in descending order starting with
the highest score: 3, 1, and 2. The path-building software should
first try building the path through certificate 3. Failing that, it
should try certificate 1. Lastly, it should try building a path
through certificate 2.

The following optimization methods specify tests developers may
choose to perform, but does not suggest scores for any of the
methods. Rather, developers should evaluate each method with respect
to the environment in which the application will operate, and assign
weights to each accordingly in the path-building software.
Additionally, many of the optimization methods are not binary in
nature. Some are tri-valued, and some may be well suited to sliding
or exponential scales. Ultimately, the implementer decides the
relative merits of each optimization with respect to his or her own
software or infrastructure.

Over and above the scores for each method, many methods can be used
to eliminate branches during the tree traversal rather than simply
scoring and weighting them. All cases where certificates could be
eliminated based upon an optimization method are noted with the
method descriptions.

Many of the sorting methods described below are based upon what has
been perceived by the authors as common in PKIs. Many of the methods
are aimed at making path building for the common PKI fast, but there
are cases where most any sorting method could lead to inefficient
path building. The desired behavior is that although one method may
lead the algorithm in the wrong direction for a given situation or
configuration, the remaining methods will overcome the errant
method(s) and send the path traversal down the correct branch of the
tree more often than not. This certainly will not be true for every
environment and configuration, and these methods may need to be
tweaked for further optimization in the application's target
operating environment.

As a final note, the list contained in this document is not intended
to be exhaustive. A developer may desire to define additional
sorting methods if the operating environment dictates the need.

3.5. Selected Methods for Sorting Certificates

The reader should draw no specific conclusions as to the relative
merits or scores for each of the following methods based upon the
order in which they appear. The relative merit of any sorting
criteria is completely dependent on the specifics of the operating
environment. For most any method, an example can be created to
demonstrate the method is effective and a counter-example could be
designed to demonstrate that it is ineffective.

Each sorting method is independent and may (or may not) be used to
assign additional scores to each certificate tested. The implementer
decides which methods to use and what weights to assign them. As
noted previously, this list is also not exhaustive.

In addition, name chaining (meaning the subject name of the issuer
certificate matches the issuer name of the issued certificate) is not
addressed as a sorting method since adherence to this is required in
order to build the decision tree to which these methods will be
applied. Also, unaddressed in the sorting methods is the prevention
of repeating certificates. Path builders should handle name chaining
and certificate repetition irrespective of the optimization approach.

Each sorting method description specifies whether the method may be
used to eliminate certificates, the number of possible numeric values
(sorting weights) for the method, components from Section 2.6 that
are required for implementing the method, forward and reverse methods
descriptions, and finally a justification for inclusion of the
method.

With regard to elimination of certificates, it is important to
understand that certificates are eliminated only at a given decision
point for many methods. For example, the path built up to
certificate X may be invalidated due to name constraints by the
addition of certificate Y. At this decision point only, Y could be
eliminated from further consideration. At some future decision
point, while building this same path, the addition of Y may not
invalidate the path.

For some other sorting methods, certificates could be eliminated from
the process entirely. For example, certificates with unsupported
signature algorithms could not be included in any path and validated.
Although the path builder may certainly be designed to operate in
this fashion, it is sufficient to always discard certificates only
for a given decision point regardless of cause.

3.5.1. basicConstraints Is Present and cA Equals True

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: None

   Forward Method: Certificates with basicConstraints present and
   cA=TRUE, or those designated as CA certificates out-of-band have
   priority. Certificates without basicConstraints, with
   basicConstraints and cA=FALSE, or those that are not designated as CA
   certificates out-of-band may be eliminated or have zero priority.

   Reverse Method: Same as forward except with regard to end entity
   certificates at the terminus of the path.

   Justification: According to [RFC3280], basicConstraints is required
   to be present with cA=TRUE in all CA certificates, or must be

   verified via an out-of-band mechanism. A valid path cannot be built
   if this condition is not met.

3.5.2. Recognized Signature Algorithms

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: None

   Forward Method: Certificates containing recognized signature and
   public key algorithms [PKIXALGS] have priority.

   Reverse Method: Same as forward.

   Justification: If the path-building software is not capable of
   processing the signatures associated with the certificate, the
   certification path cannot be validated.

3.5.3. keyUsage Is Correct

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: None

   Forward Method: If keyUsage is present, certificates with
   keyCertSign set have 100% priority. If keyUsage is present and
   keyCertSign is not set, the certificate may be eliminated or have
   zero priority. All others have zero priority.

   Reverse Method: Same as forward except with regard to end entity
   certificates at the terminus of the path.

   Justification: A valid certification path cannot be built through a
   CA certificate with inappropriate keyUsage. Note that
   digitalSignature is not required to be set in a CA certificate.

3.5.4. Time (T) Falls within the Certificate Validity

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: None

   Forward Method: Certificates that contain the required time (T)
   within their validity period have 100% priority. Otherwise, the
   certificate is eliminated or has priority zero.

   Reverse Method: Same as forward.

   Justification: A valid certification path cannot be built if T falls
   outside of the certificate validity period.

   NOTE: Special care should be taken to return a meaningful error to
   the caller, especially in the event the target certificate does not
   meet this criterion, if this sorting method is used for elimination.
   (e.g., the certificate is expired or is not yet valid).

3.5.5. Certificate Was Previously Validated

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Certification Path Cache

   Forward Method: A certificate that is present in the certification
   path cache has priority.

   Reverse Method: Does not apply. (The validity of a certificate vs.
   unknown validity does not infer anything about the correct direction
   in the decision tree. In other words, knowing the validity of a CA
   certificate does not indicate that the target is more likely found
   through that path than another.)

   Justification: Certificates in the path cache have been validated
   previously. Assuming the initial constraints have not changed, it is
   highly likely that the path from that certificate to a trust anchor
   is still valid. (Changes to the initial constraints may cause a
   certificate previously considered valid to no longer be considered
   valid.)

   Note: It is important that items in the path cache have appropriate
   life times. For example, it could be inappropriate to cache a
   relationship beyond the period the related CRL will be trusted by the
   application. It is also critical to consider certificates and CRLs
   farther up the path when setting cache lifetimes. For example, if
   the issuer certificate expires in ten days, but the issued
   certificate is valid for 20 days, caching the relationship beyond 10
   days would be inappropriate.

3.5.6. Previously Verified Signatures

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: Path Cache

   Forward Method: If a previously verified relationship exists in the
   path cache between the subject certificate and a public key present
   in one or more issuer certificates, all the certificates containing

said public key have higher priority. Other certificates may be
eliminated or set to zero priority.

Reverse Method: If known bad signature relationships exist between
certificates, these relationships can be used to eliminate potential
certificates from the decision tree. Nothing can be concluded about
the likelihood of finding a given target certificate down one branch
versus another using known good signature relationships.

Justification: If the public key in a certificate (A) was previously
used to verify a signature on a second certificate (B), any and all
certificates containing the same key as (A) may be used to verify the
signature on (B). Likewise, any certificates that do not contain the
same key as (A) cannot be used to verify the signature on (B). This
forward direction method is especially strong for multiply cross-
certified CAs after a key rollover has occurred.

3.5.7. Path Length Constraints

May be used to eliminate certificates: Yes
Number of possible values: Binary
Components required: None

Forward Method: Certificates with basic constraints present and
containing a path length constraint that would invalidate the current
path (the current length is known since the software is building from
the target certificate) may be eliminated or set to zero priority.
Otherwise, the priority is 100%.

Reverse Method: This method may be applied in reverse. To apply it,
the builder keeps a current path length constraint variable and then
sets zero priority for (or eliminates) certificates that would
violate the constraint.

Justification: A valid path cannot be built if the path length
constraint has been violated.

3.5.8. Name Constraints

May be used to eliminate certificates: Yes
Number of possible values: Binary
Components required: None

Forward Method: Certificates that contain nameConstraints that would
be violated by certificates already in the path to this point are
given zero priority or eliminated.

Reverse Method: Certificates that will allow successful processing
of any name constraints present in the path to this point are given
higher priority.

Justification: A valid path cannot be built if name constraints are
violated.

3.5.9. Certificate Is Not Revoked

May be used to eliminate certificates: No
Number of possible values: Three
Components required: CRL Cache

Forward Method: If a current CRL for a certificate is present in the
CRL cache, and the certificate serial number is not on the CRL, the
certificate has priority. If the certificate serial number is
present on the CRL, it has zero priority. If an (acceptably fresh)
OCSP response is available for a certificate, and identifies the
certificate as valid, the certificate has priority. If an OCSP
response is available for a certificate, and identifies the
certificate as invalid, the certificate has zero priority.

Reverse Method: Same as Forward.

Alternately, the certificate may be eliminated if the CRL or OCSP
response is verified. That is, fully verify the CRL or OCSP response
signature and relationship to the certificate in question in
accordance with [RFC3280]. While this is viable, the signature
verification required makes it less attractive as an elimination
method. It is suggested that this method only be used for sorting
and that CRLs and OCSP responses are validated post path building.

Justification: Certificates known to be not revoked can be
considered more likely to be valid than certificates for which the
revocation status is unknown. This is further justified if CRL or
OCSP response validation is performed post path validation - CRLs or
OCSP responses are only retrieved when complete paths are found.

NOTE: Special care should be taken to allow meaningful errors to
propagate to the caller, especially in cases where the target
certificate is revoked. If a path builder eliminates certificates
using CRLs or OCSP responses, some status information should be
preserved so that a meaningful error may be returned in the event no
path is found.

3.5.10. Issuer Found in the Path Cache

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Certification Path Cache

   Forward Method: A certificate whose issuer has an entry (or entries)
   in the path cache has priority.

   Reverse Method: Does not apply.

   Justification: Since the path cache only contains entries for
   certificates that were previously validated back to a trust anchor,
   it is more likely than not that the same or a new path may be built
   from that point to the (or one of the) trust anchor(s). For
   certificates whose issuers are not found in the path cache, nothing
   can be concluded.

   NOTE: This method is not the same as the method named "Certificate
   Was Previously Validated". It is possible for this sorting method to
   evaluate to true while the other method could evaluate to zero.

3.5.11. Issuer Found in the Application Protocol

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Certification Path Cache

   Forward Method: If the issuer of a certificate sent by the target
   through the application protocol (SSL/TLS, S/MIME, etc.), matches the
   signer of the certificate you are looking at, then that certificate
   has priority.

   Reverse Method: If the subject of a certificate matches the issuer
   of a certificate sent by the target through the application protocol
   (SSL/TLS, S/MIME, etc.), then that certificate has priority.

   Justification: The application protocol may contain certificates
   that the sender considers valuable to certification path building,
   and are more likely to lead to a path to the target certificate.

3.5.12. Matching Key Identifiers (KIDs)

   May be used to eliminate certificates: No
   Number of possible values: Three
   Components required: None

   Forward Method: Certificates whose subject key identifier (SKID)

matches the current certificate's authority key identifier (AKID)
have highest priority. Certificates without a SKID have medium
priority. Certificates whose SKID does not match the current
certificate's AKID (if both are present) have zero priority. If the
current certificate expresses the issuer name and serial number in
the AKID, certificates that match both these identifiers have highest
priority. Certificates that match only the issuer name in the AKID
have medium priority.

Reverse Method: Certificates whose AKID matches the current
certificate's SKID have highest priority. Certificates without an
AKID have medium priority. Certificates whose AKID does not match
the current certificate's SKID (if both are present) have zero
priority. If the certificate expresses the issuer name and serial
number in the AKID, certificates that match both these identifiers in
the current certificate have highest priority. Certificates that
match only the issuer name in the AKID have medium priority.

Justification: Key Identifier (KID) matching is a very useful
mechanism for guiding path building (that is their purpose in the
certificate) and should therefore be assigned a heavy weight.

NOTE: Although required to be present by [RFC3280], it is extremely
important that KIDs be used only as sorting criteria or as hints
during certification path building. KIDs are not required to match
during certification path validation and cannot be used to eliminate
certificates. This is of critical importance for interoperating
across domains and multi-vendor implementations where the KIDs may
not be calculated in the same fashion.

3.5.13. Policy Processing

May be used to eliminate certificates: Yes
Number of possible values: Three
Components required: None

Forward Method: Certificates that satisfy Forward Policy Chaining
have priority. (See Section 4 entitled "Forward Policy Chaining" for
details.) If the caller provided an initial-policy-set and did not
set the initial-require-explicit flag, the weight of this sorting
method should be increased. If the initial-require-explicit-policy
flag was set by the caller or by a certificate, certificates may be
eliminated.

Reverse Method: Certificates that contain policies/policy mappings
that will allow successful policy processing of the path to this
point have priority. If the caller provided an initial-policy-set
and did not set the initial-require-explicit flag, the weight of this

sorting method should be increased. Certificates may be eliminated
only if initial-require-explicit was set by the caller or if
require-explicit-policy was set by a certificate in the path to this
point.

Justification: In a policy-using environment, certificates that
successfully propagate policies are more likely part of an intended
certification path than those that do not.

When building in the forward direction, it is always possible that a
certificate closer to the trust anchor will set the require-
explicit-policy indicator; so giving preference to certification
paths that propagate policies may increase the probability of finding
a valid path first. If the caller (or a certificate in the current
path) has specified or set the initial-require-explicit-policy
indicator as true, this sorting method can also be used to eliminate
certificates when building in the forward direction.

If building in reverse, it is always possible that a certificate
farther along the path will set the require-explicit-policy
indicator; so giving preference to those certificates that propagate
policies will serve well in that case. In the case where require-
explicit-policy is set by certificates or the caller, certificates
can be eliminated with this method.

3.5.14. Policies Intersect the Sought Policy Set

May be used to eliminate certificates: No
Number of possible values: Additive
Components required: None

Forward Method: Certificates that assert policies found in the
initial-acceptable-policy-set have priority. Each additional
matching policy could have an additive affect on the total score.

Alternately, this could be binary; it matches 1 or more, or matches
none.

Reverse Method: Certificates that assert policies found in the
target certificate or map policies to those found in the target
certificate have priority. Each additional matching policy could
have an additive affect on the total score. Alternately, this could
be binary; it matches 1 or more, or matches none.

Justification: In the forward direction, as the path draws near to
the trust anchor in a cross-certified environment, the policies
asserted in the CA certificates will match those in the caller's
domain. Since the initial acceptable policy set is specified in the

caller's domain, matches may indicate that the path building is
drawing nearer to a desired trust anchor. In the reverse direction,
finding policies that match those of the target certificate may
indicate that the path is drawing near to the target's domain.

3.5.15. Endpoint Distinguished Name (DN) Matching

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: None

   Forward Method: Certificates whose issuer exactly matches a trust
   anchor subject DN have priority.

   Reverse Method: Certificates whose subject exactly matches the
   target entity issuer DN have priority.

   Justification: In the forward direction, if a certificate's issuer
   DN matches a trust anchor's DN [X.501], then it may complete the
   path. In the reverse direction, if the certificate's subject DN
   matches the issuer DN of the target certificate, it may be the last
   certificate required to complete the path.

3.5.16. Relative Distinguished Name (RDN) Matching

   May be used to eliminate certificates: No
   Number of possible values: Sliding Scale
   Components required: None

   Forward Method: Certificates that match more ordered RDNs between
   the issuer DN and a trust anchor DN have priority. When all the RDNs
   match, this yields the highest priority.

   Reverse Method: Certificates with subject DNs that match more RDNs
   with the target's issuer DN have higher priority. When all the RDNs
   match, this yields the highest priority.

   Justification: In PKIs the DNs are frequently constructed in a tree
   like fashion. Higher numbers of matches may indicate that the trust
   anchor is to be found in that direction within the tree. Note that
   in the case where all the RDNs match [X.501], this sorting method
   appears to mirror the preceding one. However, this sorting method
   should be capable of producing a 100% weight even if the issuer DN
   has more RDNs than the trust anchor. The Issuer DN need only contain
   all the RDNs (in order) of the trust anchor.

   NOTE: In the case where all RDNs match, this sorting method mirrors
   the functionality of the preceding one. This allows for partial

matches to be weighted differently from exact matches. Additionally,
this method can require a lot of processing if many trust anchors are
present.

3.5.17. Certificates are Retrieved from cACertificate Directory
         Attribute

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Certificate Cache with flags for the attribute
   from where the certificate was retrieved and Remote Certificate
   Storage/Retrieval using a directory

   Forward Method: Certificates retrieved from the cACertificate
   directory attribute have priority over certificates retrieved from
   the crossCertificatePair attribute. (See [RFC2587].)

   Reverse Method: Does not apply.

   Justification: The cACertificate directory attribute contains
   certificates issued from local sources and self issued certificates.
   By using the cACertificate directory attribute before the
   crossCertificatePair attribute, the path-building algorithm will
   (depending on the local PKI configuration) tend to demonstrate a
   preference for the local PKI before venturing to external cross-
   certified PKIs. Most of today's PKI applications spend most of their
   time processing information from the local (user's own) PKI, and the
   local PKI is usually very efficient to traverse due to proximity and
   network speed.

3.5.18. Consistent Public Key and Signature Algorithms

   May be used to eliminate certificates: Yes
   Number of possible values: Binary
   Components required: None

   Forward Method: If the public key in the issuer certificate matches
   the algorithm used to sign the subject certificate, then it has
   priority. (Certificates with unmatched public key and signature
   algorithms may be eliminated.)

   Reverse Method: If the public key in the current certificate matches
   the algorithm used to sign the subject certificate, then it has
   priority. (Certificates with unmatched public key and signature
   algorithms may be eliminated.)

   Justification: Since the public key and signature algorithms are not
   consistent, the signature on the subject certificate will not verify

successfully. For example, if the issuer certificate contains an RSA
public key, then it could not have issued a subject certificate
signed with the DSA-with-SHA-1 algorithm.

3.5.19. Similar Issuer and Subject Names

May be used to eliminate certificates: No
Number of possible values: Sliding Scale
Components required: None

Forward Method: Certificates encountered with a subject DN that
matches more RDNs with the issuer DN of the target certificate have
priority.

Reverse Method: Same as forward.

Justification: As it is generally more efficient to search the local
domain prior to branching to cross-certified domains, using
certificates with similar names first tends to make a more efficient
path builder. Cross-certificates issued from external domains will
generally match fewer RDNs (if any), whereas certificates in the
local domain will frequently match multiple RDNs.

3.5.20. Certificates in the Certification Cache

May be used to eliminate certificates: No
Number of possible values: Three
Components required: Local Certificate Cache and Remote Certificate
Storage/Retrieval (e.g., LDAP directory as the repository)

Forward Method: A certificate whose issuer certificate is present in
the certificate cache and populated with certificates has higher
priority. A certificate whose issuer's entry is fully populated with
current data (all certificate attributes have been searched within
the timeout period) has higher priority.

Reverse Method: If the subject of a certificate is present in the
certificate cache and populated with certificates, then it has higher
priority. If the entry is fully populated with current data (all
certificate attributes have been searched within the timeout period)
then it has higher priority.

Justification: The presence of required directory values populated
in the cache increases the likelihood that all the required
certificates and CRLs needed to complete the path from this
certificate to the trust anchor (or target if building in reverse)
are present in the cache from a prior path being developed, thereby

eliminating the need for directory access to complete the path. In
the event no path can be found, the performance cost is low since the
certificates were likely not retrieved from the network.

3.5.21. Current CRL Found in Local Cache

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components Required: CRL Cache

   Forward Method: Certificates have priority if the issuer's CRL entry
   exists and is populated with current data in the CRL cache.

   Reverse Method: Certificates have priority if the subject's CRL
   entry exists and is populated with current data in the CRL cache.

   Justification: If revocation is checked only after a complete path
   has been found, this indicates that a complete path has been found
   through this entity at some past point, so a path still likely
   exists. This also helps reduce remote retrievals until necessary.

3.6. Certificate Sorting Methods for Revocation Signer Certification
      Paths

   Unless using a locally-configured OCSP responder or some other
   locally-configured trusted revocation status service, certificate
   revocation information is expected to be provided by the PKI that
   issued the certificate. It follows that when building a
   certification path for a Revocation Signer certificate, it is
   desirable to confine the building algorithm to the PKI that issued
   the certificate. The following sorting methods seek to order
   possible paths so that the intended Revocation Signer certification
   path is found first.

   These sorting methods are not intended to be used in lieu of the ones
   described in the previous section; they are most effective when used
   in conjunction with those in Section 3.5. Some sorting criteria below
   have identical names as those in the preceding section. This
   indicates that the sorting criteria described in the preceding
   section are modified slightly when building the Revocation Signer
   certification path.

3.6.1. Identical Trust Anchors

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Is-revocation-signer indicator and the
   Certification Authority's trust anchor

Forward Method: Not applicable.

Reverse Method: Path building should begin from the same trust
anchor used to validate the Certification Authority before trying any
other trust anchors. If any trust anchors exist with a different
public key but an identical subject DN to that of the Certification
Authority's trust anchor, they should be tried prior to those with
mismatched names.

Justification: The revocation information for a given certificate
should be produced by the PKI that issues the certificate.
Therefore, building a path from a different trust anchor than the
Certification Authority's is not desirable.

3.6.2. Endpoint Distinguished Name (DN) Matching

May be used to eliminate certificates: No
Number of possible values: Binary
Components required: Is-revocation-signer indicator and the
Certification Authority's trust anchor

Forward Method: Operates identically to the sorting method described
in 3.5.15, except that instead of performing the matching against all
trust anchors, the DN matching is performed only against the trust
anchor DN used to validate the CA certificate.

Reverse Method: No change for Revocation Signer's certification
path.

Justification: The revocation information for a given certificate
should be produced by the PKI that issues the certificate.
Therefore, building a path to a different trust anchor than the CA's
is not desirable. This sorting method helps to guide forward
direction path building toward the trust anchor used to validate the
CA certificate.

3.6.3. Relative Distinguished Name (RDN) Matching

May be used to eliminate certificates: No
Number of possible values: Sliding Scale
Components required: Is-revocation-signer indicator and the
Certification Authority's trust anchor

Forward Method: Operates identically to the sorting method described
in 3.5.16 except that instead of performing the RDN matching against
all trust anchors, the matching is performed only against the trust
anchor DN used to validate the CA certificate.

   Reverse Method: No change for Revocation Signer's certification
   path.

   Justification: The revocation information for a given certificate
   should be produced by the PKI that issues the certificate.
   Therefore, building a path to a different trust anchor than the CA's
   is not desirable. This sorting method helps to guide forward
   direction path building toward the trust anchor used to validate the
   CA certificate.

3.6.4. Identical Intermediate Names

   May be used to eliminate certificates: No
   Number of possible values: Binary
   Components required: Is-revocation-signer indicator and the
   Certification Authority's complete certification path

   Forward Method: If the issuer DN in the certificate matches the
   issuer DN of a certificate in the Certification Authority's path, it
   has higher priority.

   Reverse Method: If the subject DN in the certificate matches the
   subject DN of a certificate in the Certification Authority's path, it
   has higher priority.

   Justification: Following the same path as the Certificate should
   deter the path-building algorithm from wandering in an inappropriate
   direction. Note that this sorting method is indifferent to whether
   the certificate is self-issued. This is beneficial in this situation
   because it would be undesirable to lower the priority of a re-key
   certificate.

4. Forward Policy Chaining

   It is tempting to jump to the conclusion that certificate policies
   offer little assistance to path building when building from the
   target certificate. It's easy to understand the "validate as you go"
   approach from the trust anchor, and much less obvious that any value
   can be derived in the other direction. However, since policy
   validation consists of the intersection of the issuer policy set with
   the subject policy set and the mapping of policies from the issuer
   set to the subject set, policy validation can be done while building
   a path in the forward direction as well as the reverse. It is simply
   a matter of reversing the procedure. That is not to say this is as
   ideal as policy validation when building from the trust anchor, but
   it does offer a method that can be used to mostly eliminate what has
   long been considered a weakness inherent to building in the forward
   (from the target certificate) direction.

4.1. Simple Intersection

   The most basic form of policy processing is the intersection of the
   policy sets from the first CA certificate through the target
   certificate. Fortunately, the intersection of policy sets will
   always yield the same final set regardless of the order of
   intersection. This allows processing of policy set intersections in
   either direction. For example, if the trust anchor issues a CA
   certificate (A) with policies {X,Y,Z}, and that CA issues another CA
   certificate (B) with policies {X,Y}, and CA B then issues a third CA
   certificate (C) with policy set {Y,G}, one normally calculates the
   policy set from the trust anchor as follows:

   1) Intersect A{X,Y,Z} with B{X,Y} to yield the set {X,Y}

   2) Intersect that result, {X,Y} with C{Y,G} to yield the final set
      {Y}

   Now it has been shown that certificate C is good for policy Y.

   The other direction is exactly the same procedure, only in reverse:

   1) Intersect C{Y,G} with B{X,Y} to yield the set {Y}

   2) Intersect that result, {Y} with A{X,Y,Z} to yield the final set
      {Y}

   Just like in the reverse direction, it has been shown that
   certificate C is good for policy Y, but this time in the forward
   direction.

   When building in the forward direction, policy processing is handled
   much like it is in reverse -- the software lends preference to
   certificates that propagate policies. Neither approach guarantees
   that a path with valid policies will be found, but rather both
   approaches help guide the path in the direction it should go in order
   for the policies to propagate.

   If the caller has supplied an initial-acceptable-policy set, there is
   less value in using it when building in the forward direction unless
   the caller also set inhibit-policy-mapping. In that case, the path
   builder can further constrain the path building to propagating
   policies that exist in the initial-acceptable-policy-set. However,
   even if the inhibit-policy-mapping is not set, the initial-policy-set
   can still be used to guide the path building toward the desired trust
   anchor.

4.2. Policy Mapping

   When a CA issues a certificate into another domain, an environment
   with disparate policy identifiers to its own, the CA may make use of
   policy mappings to map equivalence from the local domain's policy to
   the non-local domain's policy. If in the prior example, A had
   included a policy mapping that mapped X to G in the certificate it
   issued to B, C would be good for X and Y:

   1) Intersect A{X,Y,Z} with B{X,Y} to yield the set {X,Y}

   2) Process Policy Mappings in B's certificate (X maps to G) to yield
      {G,Y} (same as {Y,G})

   3) Intersect that result, {G,Y} with C{Y,G} to yield the final set
      {G,Y}

   Since policies are always expressed in the relying party's domain,
   the certificate C is said to be good for {X, Y}, not {Y, G}. This is
   because "G" doesn't mean anything in the context of the trust anchor
   that issued A without the policy mapping.

   When building in the forward direction, policies can be "unmapped" by
   reversing the mapping procedure. This procedure is limited by one
   important aspect: if policy mapping has occurred in the forward
   direction, there is no mechanism by which it can be known in advance
   whether or not a future addition to the current path will invalidate
   the policy chain (assuming one exists) by setting inhibit-policy-
   mapping. Fortunately, it is uncommon practice to set this flag. The
   following is the procedure for processing policy mapping in the
   forward direction:

   1) Begin with C's policy set {Y,G}

   2) Apply the policy mapping in B's certificate (X maps to G) in
      reverse to yield {Y,X} (same as {X,Y})

   3) Intersect the result {X,Y} with B{X,Y} to yield the set {X,Y}

   4) Intersect that result, {X,Y}, with A{X,Y,Z} to yield the final set
      {X,Y}

   Just like in the reverse direction, it is determined in the forward
   direction that certificate C is good for policies {X,Y}. If during
   this procedure, an inhibit-policy-mapping flag was encountered, what
   should be done? This is reasonably easy to keep track of as well.
   The software simply maintains a flag on any policies that were
   propagated as a result of a mapping; just a simple Boolean kept with

the policies in the set. Imagine now that the certificate issued to
A has the inhibit-policy-mapping constraint expressed with a skip
certificates value of zero.

1) Begin with C's policy set {Y,G}

2) Apply the policy mapping in B's certificate and mark X as
   resulting from a mapping. (X maps to G) in reverse to yield {Y,Xm}
   (same as {Xm,Y})

3) Intersect the result {Xm,Y} with B{X,Y} to yield the set {Xm,Y}

4) A's certificate expresses the inhibit policy mapping constraint,
   so eliminate any policies in the current set that were propagated
   due to mapping (which is Xm) to yield {Y}

5) Intersect that result, {Y} with A{X,Y,Z} to yield the final set
   {Y}

If in our example, the policy set had gone to empty at any point (and
require-explicit-policy was set), the path building would back up and
try to traverse another branch of the tree. This is analogous to the
path-building functionality utilized in the reverse direction when
the policy set goes to empty.

4.3. Assigning Scores for Forward Policy Chaining

   Assuming the path-building module is maintaining the current forward
   policy set, weights may be assigned using the following procedure:

   1) For each CA certificate being scored:

      a. Copy the current forward policy set.

      b. Process policy mappings in the CA certificate in order to
         "un-map" policies, if any.

      c. Intersect the resulting set with CA certificate's policies.

   The larger the policy set yielded, the larger the score for that CA
   certificate.

   2) If an initial acceptable set was supplied, intersect this set with
      the resulting set for each CA certificate from (1).

   The larger the resultant set, the higher the score is for this
   certificate.

Other scoring schemes may work better if the operating environment
dictates.

5. Avoiding Path-Building Errors

This section defines some errors that may occur during the path-
building process, as well as ways to avoid these errors when
developing path-building functions.

5.1. Dead Ends

When building certification paths in a non-hierarchical PKI
structure, a simple path-building algorithm could fail prematurely
without finding an existing path due to a "dead end". Consider the
example in Figure 14.

```
         +----+        +---+
         | TA |        | Z |
         +----+        +---+
            |             |
            |             |
            V             V
         +---+         +---+
         | C |<-----| Y |
         +---+         +---+
            |
            |
            V
        +--------+
        | Target |
        +--------+
```
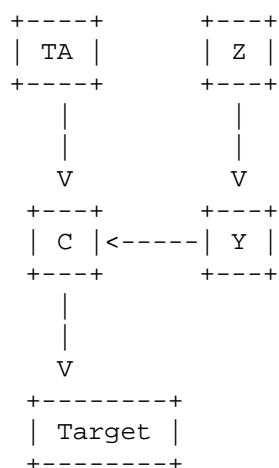
        Figure 14 - Dead End Example

Note that in the example, C has two certificates: one issued by Y,
and the other issued by the Trust Anchor. Suppose that a simple
"find issuer" algorithm is used, and the order in which the path
builder found the certificates was Target(C), C(Y), Y(Z), Z(Z). In
this case, Z has no certificates issued by any other entities, and so
the simplistic path-building process stops. Since Z is not the
relying party's trust anchor, the certification path is not complete,
and will not validate. This example shows that in anything but the
simplest PKI structure, additional path-building logic will need to
handle the cases in which entities are issued multiple certificates
from different issuers. The path-building algorithm will also need
to have the ability to traverse back up the decision tree and try
another path in order to be robust.

5.2. Loop Detection

   In a non-hierarchical PKI structure, a path-building algorithm may
   become caught in a loop without finding an existing path. Consider
   the example below:

                +----+
                | TA |
                +----+
                  |
                  |
                +---+   +---+
                | A | ->| Z |
                +---+   /   +---+
                  | / |
                  | / |
                  V /   V
                +---+   +---+
                | B |<-----| Y |
                +---+   +---+
                  |
                  |
                  V
                +--------+
                | Target |
                +--------+
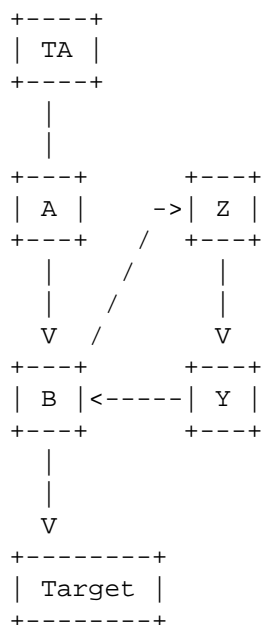
        Figure 15 - Loop Example

   Let us suppose that in this example the simplest "find issuer"
   algorithm is used, and the order in which certificates are retrieved
   is Target(B), B(Y), Y(Z), Z(B), B(Y), Y(Z), Z(B), B(Y), ... A loop
   has formed that will cause the correct path (Target, B, A) to never
   be found. The certificate processing system will need to recognize
   loops created by duplicate certificates (which are prohibited in a
   path by [X.509]) before they form to allow the certification path-
   building process to continue and find valid paths. The authors of
   this document recommend that the loop detection not only detect the
   repetition of a certificate in the path, but also detect the presence
   of the same subject name / subject alternative name/ subject public
   key combination occurring twice in the path. A name/key pair should
   only need to appear once in the path. (See Section 2.4.2 for more
   information on the reasoning behind this recommendation.)

5.3. Use of Key Identifiers

   Inconsistent and/or incompatible approaches to computing the subject
   key identifier and authority key identifier in public key
   certificates can cause failures in certification path-building
   algorithms that use those fields to identify certificates, even
   though otherwise valid certification paths may exist. Path-building
   implementations should use existing key identifiers and not attempt
   to re-compute subject key identifiers. It is extremely important
   that Key Identifiers be used only as sorting criteria or hints. KIDs
   are not required to match during certification path validation and
   cannot be used to eliminate certificates. This is of critical
   importance for interoperating across domains and multi-vendor
   implementations where the KIDs may not be calculated in the same
   fashion.

   Path-building and processing implementations should not rely on the
   form of authority key identifier that uses the authority DN and
   serial number as a restrictive matching rule, because cross-
   certification can lead to this value not being matched by the cross-
   certificates.

5.4. Distinguished Name Encoding

   Certification path-building software should not rely on DNs being
   encoded as PrintableString. Although frequently encoded as
   PrintableString, DNs may also appear as other types, including
   BMPString or UTF8String. As a result, software systems that are
   unable to process BMPString and UTF8String encoded DNs may be unable
   to build and validate some certification paths.

   Furthermore, [RFC3280] compliant certificates are required to encode
   DNs as UTF8String as of January 1, 2004. Certification path-building
   software should be prepared to handle "name rollover" certificates as
   described in [RFC3280]. Note that the inclusion of a "name rollover"
   certificate in a certification path does not constitute repetition of
   a DN and key. Implementations that include the "name rollover"
   certificate in the path should ensure that the DNs with differing
   encoding are regarded as dissimilar. (Implementations may instead
   handle matching DNs of different encodings and will therefore not
   need to include "name rollover" certificates in the path.)

6. Retrieval Methods

   Building a certification path requires the availability of the
   certificates and CRLs that make up the path. There are many
   different methods for obtaining these certificates and CRLs. This
   section lists a few of the common ways to perform this retrieval, as
   well as some suggested approaches for improving performance. This
   section is not intended to provide a complete reference for
   certificate and CRL retrieval methods or optimizations that would be
   useful in certification path building.

6.1. Directories Using LDAP

   Most applications utilize the Lightweight Directory Access Protocol
   (LDAP) when retrieving data from directories following the X.500
   model. Applications may encounter directories which support either
   LDAP v2 [RFC1777] or LDAP v3 [RFC3377].

   The LDAP v3 specification defines one attribute retrieval option, the
   "binary" option. When specified in an LDAP retrieval request, this
   option was intended to force the directory to ignore any string-based
   representations of BER-encoded directory information, and send the
   requested attribute(s) in BER format. Since all PKI objects of
   concern are BER-encoded objects, the "binary" option should be used.
   However, not all directories support the "binary" option. Therefore,
   applications should be capable of requesting attributes with and
   without the "binary" option. For example, if an application wishes
   to retrieve the userCertificate attribute, the application should
   request "userCertificate;binary". If the desired information is not
   returned, robust implementations may opt to request "userCertificate"
   as well.

   The following attributes should be considered by PKI application
   developers when performing certificate retrieval from LDAP sources:

   userCertificate: contains certificates issued by one or more
      certification authorities with a subject DN that matches that of
      the directory entry. This is a multi-valued attribute and all
      values should be received and considered during path building.
      Although typically it is expected that only end entity
      certificates will be stored in this attribute, (e.g., this is the
      attribute an application would request to find a person's
      encryption certificate) implementers may opt to search this
      attribute when looking in CA entries to make their path builder
      more robust. If it is empty, the overhead added by including this
      attribute when already requesting one or both of the two below is
      marginal.

cACertificate: contains self-issued certificates (if any) and any
   certificates issued to this certification authority by other
   certification authorities in the same realm. (Realm is dependent
   upon local policy.) This is a multi-valued attribute and all
   values should be received and considered during path building.

crossCertificatePair: in conformant implementations, the
   crossCertificatePair is used to contain all, except self-issued
   certificates issued to this certification authority, as well as
   certificates issued by this certification authority to other
   certification authorities. Each attribute value is a structure
   containing two elements. The issuedToThisCA element contains
   certificates issued to this certification authority by other
   certification authorities. The issuedByThisCA element contains
   certificates issued by this certification authority to other
   certification authorities. Both elements of the
   crossCertificatePair are labeled optional in the ASN.1 definition.
   If both elements are present in a single value, the issuer name in
   one certificate is required to match the subject name in the other
   and vice versa, and the subject public key in one certificate
   shall be capable of verifying the digital signature on the other
   certificate and vice versa. As this technology has evolved,
   different standards have had differing requirements on where
   information could be found. For example, the LDAP v2 schema
   [RFC2587] states that the issuedToThisCA (once called 'forward')
   element of the crossCertificatePair attribute is mandatory and the
   issuedByThisCA (once called 'reverse') element is optional. In
   contrast, Section 11.2.3 of [X.509] requires the issuedByThisCA
   element to be present if the CA issues a certificate to another CA
   if the subject is not a subordinate CA in a hierarchy. Conformant
   directories behave as required by [X.509], but robust path-
   building implementations may want to retrieve all certificates
   from the cACertificate and crossCertificatePair attributes to
   ensure all possible certification authority certificates are
   obtained.

certificateRevocationList: the certificateRevocationList attribute
   contains a certificate revocation list (CRL). A CRL is defined in
   [RFC3280] as a time stamped list identifying revoked certificates,
   which is signed by a CA or CRL issuer and made freely available in
   a public repository. Each revoked certificate is identified in a
   CRL by its certificate serial number. There may be one or more
   CRLs in this attribute, and the values should be processed in
   accordance with [RFC3280].

   authorityRevocationList: the authorityRevocationList attribute also
      contains CRLs. These CRLs contain revocation information
      regarding certificates issued to other CAs. There may be one or
      more CRLs in this attribute, and the values should be processed in
      accordance with [RFC3280].

   Certification path processing systems that plan to interoperate with
   varying PKI structures and directory designs should at a minimum be
   able to retrieve and process the userCertificate, cACertificate,
   crossCertificatePair, certificateRevocationList, and
   authorityRevocationList attributes from directory entries.

6.2. Certificate Store Access via HTTP

   Another possible method of certificate retrieval is using HTTP as an
   interface mechanism for retrieving certificates and CRLs from PKI
   repositories. A current PKIX document [CERTSTORE] provides a
   protocol for a general-purpose interface capability for retrieving
   certificates and CRLs from PKI repositories. Since the [CERTSTORE]
   document is a work in progress as of the writing of this document, no
   details are given here on how to utilize this mechanism for
   certificate and CRL retrieval. Instead, refer to the [CERTSTORE]
   document or its current version. Certification path processing
   systems may wish to implement support for this interface capability,
   especially if they will be used in environments that will provide
   HTTP-based access to certificates and CRLs.

6.3. Authority Information Access

   The authority information access (AIA) extension, defined within
   [RFC3280], indicates how to access CA information and services for
   the issuer of the certificate in which the extension appears. If a
   certificate with an AIA extension contains an accessMethod defined
   with the id-ad-caIssuers OID, the AIA may be used to retrieve one or
   more certificates for the CA that issued the certificate containing
   the AIA extension. The AIA will provide a uniform resource
   identifier (URI) [RFC3986] when certificates can be retrieved via
   LDAP, HTTP, or FTP. The AIA will provide a directoryName when
   certificates can be retrieved via directory access protocol (DAP).
   The AIA will provide an rfc822Name when certificates can be retrieved
   via electronic mail. Additionally, the AIA may specify the location
   of an OCSP [RFC2560] responder that is able to provide revocation
   information for the certificate.

   If present, AIA may provide forward path-building implementations
   with a direct link to a certificate for the issuer of a given
   certificate. Therefore, implementations may wish to provide support
   for decoding the AIA extension and processing the LDAP, HTTP, FTP,

   DAP, or e-mail locators. Support for AIA is optional; [RFC3280]
   compliant implementations are not required to populate the AIA
   extension. However, implementers of path-building and validation
   modules are strongly encouraged to support AIA, especially the HTTP
   transport; this will provide for usability and interoperability with
   many existing PKIs.

6.4. Subject Information Access

   The subject information access (SIA) extension, defined within
   [RFC3280], indicates how to access information and services for the
   subject of the certificate in which the extension appears. If a
   certificate with an SIA extension contains an accessMethod defined
   with the id-ad-caRepository OID, the SIA may be used to locate one or
   more certificates (and possibly CRLs) for entities issued
   certificates by the subject. The SIA will provide a uniform resource
   identifier (URI) [RFC3986] when data can be retrieved via LDAP, HTTP,
   or FTP. The SIA will provide a directoryName when data can be
   retrieved via directory access protocol (DAP). The SIA will provide
   an rfc822Name when data can be retrieved via electronic mail.

   If present, the SIA extension may provide reverse path-building
   implementations with the certificates required to continue building
   the path. Therefore, implementations may wish to provide support for
   decoding the SIA extension and processing the LDAP, HTTP, FTP, DAP,
   or e-mail locators. Support for SIA is optional; [RFC3280] compliant
   implementations are not required to populate the SIA extension.
   However, implementers of path-building and validation modules are
   strongly encouraged to support SIA, especially the HTTP transport;
   this will provide for usability and interoperability with many
   existing PKIs.

6.5. CRL Distribution Points

   The CRL distribution points (CRLDP) extension, defined within
   [RFC3280], indicates how to access CRL information. If a CRLDP
   extension appears within a certificate, the CRL(s) to which the CRLDP
   refer are generally the CRLs that would contain revocation
   information for the certificate. The CRLDP extension may point to
   multiple distribution points from which the CRL information may be
   obtained; the certificate processing system should process the CRLDP
   extension in accordance with [RFC3280]. The most common distribution
   points contain URIs from which the appropriate CRL may be downloaded,
   and directory names, which can be queried in a directory to retrieve
   the CRL attributes from the corresponding entry.

If present, CRLDP can provide certificate processing implementations
with a link to CRL information for a given certificate. Therefore,
implementations may wish to provide support for decoding the CRLDP
extension and using the information to retrieve CRLs. Support for
CRLDP is optional and [RFC3280] compliant implementations need not
populate the CRLDP extension. However, implementers of path-building
and validation modules are strongly encouraged to support CRLDPs. At
a minimum, developers are encouraged to consider supporting the LDAP
and HTTP transports; this will provide for interoperability across a
wide range of existing PKIs.

6.6. Data Obtained via Application Protocol

Many application protocols, such as SSL/TLS and S/MIME, allow one
party to provide certificates and CRLs to another. Data provided in
this method is generally very valuable to path-building software
(will provide direction toward valid paths), and should be stored and
used accordingly. Note: self-signed certificates obtained via
application protocol are not trustworthy; implementations should only
consider the relying party's trust anchors when building paths.

6.7. Proprietary Mechanisms

Some certificate issuing systems and certificate processing systems
may utilize proprietary retrieval mechanisms, such as network mapped
drives, databases, or other methods that are not directly referenced
via the IETF standards. Certificate processing systems may wish to
support other proprietary mechanisms, but should only do so in
addition to supporting standard retrieval mechanisms such as LDAP,
AIA, and CRLDP (unless functioning in a closed environment).

7. Improving Retrieval Performance

Retrieval performance can be improved through a few different
mechanisms, including the use of caches and setting a specific
retrieval order. This section discusses a few methods by which the
performance of a certificate processing system may be improved during
the retrieval of PKI objects. Certificate processing systems that
are consistently very slow during processing will be disliked by
users and will be slow to be adopted into organizations. Certificate
processing systems are encouraged to do whatever possible to reduce
the delays associated with requesting and retrieving data from
external sources.

7.1. Caching

   Certificate processing systems operating in a non-hierarchical PKI
   will often need to retrieve certificates and certificate revocation
   lists (CRLs) from a source outside the application protocol.
   Typically, these objects are retrieved from an X.500 or LDAP
   repository, an Internet URI [RFC3986], or some other non-local
   source. Due to the delays associated with establishing connections
   as well as network transfers, certificate processing systems ought to
   be as efficient as possible when retrieving data from external
   sources. Perhaps the best way to improve retrieval efficiency is by
   using a caching mechanism. Certificate processing systems can cache
   data retrieved from external sources for some period of time, but not
   to exceed the useful period of the data (i.e., an expired certificate
   need not be cached). Although this comes at a cost of increased
   memory/disk consumption by the system, the cost and performance
   benefit of reducing network transmissions is great. Also, CRLs are
   often issued and available in advance of the nextUpdate date in the
   CRL. Implementations may wish to obtain these "fresher" CRLs before
   the nextUpdate date has passed.

   There are a number of different ways in which caching can be
   implemented; the specifics of these methods can be used as
   distinguishing characteristics between certificate processing
   systems. However, some things that implementers may wish to consider
   when developing caching systems are as follows:

      - If PKI objects are cached, the certification path-building
        mechanism should be able to examine and retrieve from the cache
        during path building. This will allow the certificate
        processing system to find or eliminate one or more paths quickly
        without requiring external contact with a directory or other
        retrieval mechanism.

      - Sharing caches between multiple users (via a local area network
        or LAN) may be useful if many users in one organization
        consistently perform PKI operations with another organization.

      - Caching not only PKI objects (such as certificates and CRLs) but
        also relationships between PKI objects (storing a link between a
        certificate and the issuer's certificate) may be useful. This
        linking may not always lead to the most correct or best
        relationship, but could represent a linking that worked in
        another scenario.

      - Previously built paths and partial paths are quite useful to
        cache, because they will provide information on previous
        successes or failures. Additionally, if the cache is safe from

         unauthorized modifications, caching validation and signature
         checking status for certificates, CRLs, and paths can also be
         stored.

7.2. Retrieval Order

   To optimize efficiency, certificate processing systems are encouraged
   to also consider the order in which different PKI objects are
   retrieved, as well as the mechanism from which they are retrieved.
   If caching is utilized, the caches can be consulted for PKI objects
   before attempting other retrieval mechanisms. If multiple caches are
   present (such as local disk and network), the caches can be consulted
   in the order in which they can be expected to return their result
   from fastest to slowest. For example, if a certificate processing
   system wishes to retrieve a certificate with a particular subject DN,
   the system might first consult the local cache, then the network
   cache, and then attempt directory retrieval. The specifics of the
   types of retrieval mechanisms and their relative costs are left to
   the implementer.

   In addition to ordering retrieval mechanisms, the certificate
   processing system ought to order the relative merits of the different
   external sources from which a PKI object can be retrieved. If the
   AIA is present within a certificate, with a URI [RFC3986] for the
   issuer's certificate, the certificate processing system (if able) may
   wish to attempt to retrieve the certificate first from local cache
   and then by using that URI (because it is expected to point directly
   to the desired certificate) before attempting to retrieve the
   certificates that may exist within a directory.

   If a directory is being consulted, it may be desirable to retrieve
   attributes in a particular order. A highly cross-certified PKI
   structure will lead to multiple possibilities for certification
   paths, which may mean multiple validation attempts before a
   successful path is retrieved. Therefore, cACertificate and
   userCertificate (which typically contain certificates from within the
   same 'realm') could be consulted before attempting to retrieve the
   crossCertificatePair values for an entry. Alternately, all three
   attributes could be retrieved in one query, but cross-certificates
   then tagged as such and used only after exhausting the possibilities
   from the cACertificate attribute. The best approach will depend on
   the nature of the application and PKI environment.

7.3. Parallel Fetching and Prefetching

   Much of this document has focused on a path-building algorithm that
   minimizes the performance impact of network retrievals, by preventing
   those retrievals and utilization of caches. Another way to improve

performance would be to allow network retrievals to be performed in
advance (prefetching) or at the same time that other operations are
performed (parallel fetching). For example, if an email application
receives a signed email message, it could download the required
certificates and CRLs prior to the recipient viewing (or attempting
to verify) the message. Implementations that provide the capability
of parallel fetching and/or prefetching, along with a robust cache,
can lead to greatly improved performance or user experience.

8. Security Considerations

8.1. General Considerations for Building a Certification Path

   Although certification path building deals directly with security
   relevant PKI data, the PKI data itself needs no special handling
   because its integrity is secured with the digital signature applied
   to it. The only exception to this is the appropriate protection of
   the trust anchor public keys. These are to be kept safe and obtained
   out of band (e.g., not from an electronic mail message or a
   directory) with respect to the path-building module.

   The greatest security risks associated with this document revolve
   around performing certification path validation while certification
   paths are built. It is therefore noted here that fully implemented
   certification path validation in accordance with [RFC3280] and
   [X.509] is required in order for certification path building,
   certification path validation, and the certificate using application
   to be properly secured. All of the Security Considerations listed in
   Section 9 of [RFC3280] apply equally here.

   In addition, as with any application that consumes data from
   potentially untrusted network locations, certification path-building
   components should be carefully implemented so as to reduce or
   eliminate the possibility of network based exploits. For example, a
   poorly implemented path-building module may not check the length of
   the CRLDP URI [RFC3986] before using the C language strcpy() function
   to place the address in a 1024 byte buffer. A hacker could use such
   a flaw to create a buffer overflow exploit by encoding malicious
   assembly code into the CRLDP of a certificate and then use the
   certificate to attempt an authentication. Such an attack could yield
   system level control to the attacker and expose the sensitive data
   the PKI was meant to protect.

   Path building may be used to mount a denial of service (DOS) attack.
   This might occur if multiple simple requests could be performed that
   cause a server to perform a number of path developments, each taking
   time and resources from the server. Servers can help avoid this by
   limiting the resources they are willing to devote to path building,

and being able to further limit those resources when the load is
heavy. Standard DOS protections such as systems that identify and
block attackers can also be useful.

A DOS attack can be also created by presenting spurious CA
certificates containing very large public keys. When the system
attempts to use the large public key to verify the digital signature
on additional certificates, a long processing delay may occur. This
can be mitigated by either of two strategies. The first strategy is
to perform signature verifications only after a complete path is
built, starting from the trust anchor. This will eliminate the
spurious CA certificate from consideration before the large public
key is used. The second strategy is to recognize and simply reject
keys longer than a certain size.

A similar DOS attack can occur with very large public keys in end
entity certificates. If a system uses the public key in a
certificate before building and validating that certificate's
certification path, long processing delays may occur. To mitigate
this threat, the public key in an end entity certificate should not
be used for any purpose until a complete certification path for that
certificate is built and validated.

8.2. Specific Considerations for Building Revocation Signer
      Certification Paths

If the CRL Signer certificate (and certification path) is not
identical to the Certification Authority certificate (and
certification path), special care should be exercised when building
the CRL Signer certification path.

If special consideration is not given to building a CRL Signer
certification path, that path could be constructed such that it
terminates with a different root or through a different certification
path to the same root. If this behavior is not prevented, the
relying party may end up checking the wrong revocation data, or even
maliciously substituted data, resulting in denial of service or
security breach.

For example, suppose the following certification path is built for E
and is valid for an example "high assurance" policy.

    A->B->C->E

When the building/validation routine attempts to verify that E is not
revoked, C is referred to as the Certification Authority certificate.
The path builder finds that the CRL for checking the revocation
status of E is issued by C2; a certificate with the subject name "C",

but with a different key than the key that was used to sign E. C2 is
referred to as the CRL Signer. An unrestrictive certification path
builder might then build a path such as the following for the CRL
Signer C2 certificate:

    X->Y->Z->C2

If a path such as the one above is permitted, nothing can be
concluded about the revocation status of E since C2 is a different CA
from C.

Fortunately, preventing this security problem is not difficult and
the solution also makes building CRL Signer certification paths very
efficient. In the event the CRL Signer certificate is identical to
the Certification Authority certificate, the Certification Authority
certification path should be used to verify the CRL; no additional
path building is required. If the CRL Signer certificate is not
identical to the Certification Authority certificate, a second path
should be built for the CRL Signer certificate in exactly the same
fashion as for any certificate, but with the following additional
guidelines:

1. Trust Anchor: The CRL Signer's certification path should start
    with the same trust anchor as the Certification Authority's
    certification path. Any trust anchor certificate with a subject
    DN matching that of the Certification Authority's trust anchor
    should be considered acceptable though lower in priority than the
    one with a matching public key and subject DN. While different
    trust anchor public keys are acceptable at the beginning of the
    CRL signer's certification path and the Certification Authority's
    certification path, both keys must be trusted by the relying
    party per the recommendations in Section 8.1.

2. CA Name Matching: The subject DNs for all CA certificates in the
    two certification paths should match on a one-to-one basis
    (ignoring self-issued certificates) for the entire length of the
    shorter of the two paths.

3. CRL Signer Certification Path Length: The length of the CRL
    Signer certification path (ignoring self-issued certificates)
    should be equal to or less than the length of the Certification
    Authority certification path plus (+) one. This allows a given
    Certification Authority to issue a certificate to a
    delegated/subordinate CRL Signer. The latter configuration
    represents the maximum certification path length for a CRL Signer
    certificate.

The reasoning behind the first guideline is readily apparent.
Lacking this and the second guideline, any trusted CA could issue
CRLs for any other CA, even if the PKIs are not related in any
fashion. For example, one company could revoke certificates issued
by another company if the relying party trusted the trust anchors
from both companies. The two guidelines also prevent erroneous CRL
checks since Global uniqueness of names is not guaranteed.

The second guideline prevents roaming certification paths such as the
previously described example CRL Signer certification path for
A->B->C->E. It is especially important that the "ignoring self-
issued certificates" is implemented properly. Self-issued
certificates are cast out of the one-to-one name comparison in order
to allow for key rollover. The path-building algorithm may be
optimized to only consider certificates with the acceptable subject
DN for the given point in the CRL Signer certification path while
building the path.

The third and final guideline ensures that the CRL used is the
intended one. Without a restriction on the length of the CRL Signer
certification path, the path could roam uncontrolled into another
domain and still meet the first two guidelines. For example, again
using the path A->B->C->E, the Certification Authority C, and a CRL
Signer C2, a CRL Signer certification path such as the following
could pass the first two guidelines:

    A->B->C->D->X->Y->RogueCA->C2

In the preceding example, the trust anchor is identical for both
paths and the one-to-one name matching test passes for A->B->C.
However, accepting such a path has obvious security consequences, so
the third guideline is used to prevent this situation. Applying the
second and third guideline to the certification path above, the path
builder could have immediately detected this path was not acceptable
(prior to building it) by examining the issuer DN in C2. Given the
length and name guidelines, the path builder could detect that
"RogueCA" is not in the set of possible names by comparing it to the
set of possible CRL Signer issuer DNs, specifically, A, B, or C.

Similar consideration should be given when building the path for the
OCSP Responder certificate when the CA is the OCSP Response Signer or
the CA has delegated the OCSP Response signing to another entity.

9. Acknowledgements

   The authors extend their appreciation to David Lemire for his efforts
   coauthoring "Managing Interoperability in Non-Hierarchical Public Key
   Infrastructures" from which material was borrowed heavily for use in
   the introductory sections.

   This document has also greatly benefited from the review and
   additional technical insight provided by Dr. Santosh Chokhani, Carl
   Wallace, Denis Pinkas, Steve Hanna, Alice Sturgeon, Russ Housley, and
   Tim Polk.

10. Normative References

   [RFC3280]   Housley, R., Polk, W., Ford, W., and D. Solo, "Internet
               X.509 Public Key Infrastructure Certificate and
               Certificate Revocation List (CRL) Profile", RFC 3280,
               April 2002.

11. Informative References

   [MINHPKIS] Hesse, P., and D. Lemire, "Managing Interoperability in
               Non-Hierarchical Public Key Infrastructures", 2002
               Conference Proceedings of the Internet Society Network
               and Distributed System Security Symposium, February 2002.

   [RFC1777]   Yeong, W., Howes, T., and S. Kille, "Lightweight
               Directory Access Protocol", RFC 1777, March 1995.

   [RFC2560]   Myers, M., Ankney, R., Malpani, A., Galperin, S., and C.
               Adams, "X.509 Internet Public Key Infrastructure Online
               Certificate Status Protocol - OCSP", RFC 2560, June 1999.

   [RFC2587]   Boeyen, S., Howes, T., and P. Richard, "Internet X.509
               Public Key Infrastructure LDAPv2 Schema", RFC 2587, June
               1999.

   [RFC3377]   Hodges, J. and R. Morgan, "Lightweight Directory Access
               Protocol (v3): Technical Specification", RFC 3377,
               September 2002.

   [RFC3820]   Tuecke, S., Welch, V., Engert, D., Pearlman, L., and M.
               Thompson, "Internet X.509 Public Key Infrastructure (PKI)
               Proxy Certificate Profile", RFC 3820, June 2004.

   [RFC3986]   Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
               Resource Identifier (URI): Generic Syntax", STD 66, RFC
               3986, January 2005.

   [X.501]   ITU-T Recommendation X.501: Information Technology - Open
                Systems Interconnection - The Directory: Models, 1993.

   [X.509]   ITU-T Recommendation X.509 (2000 E): Information
                Technology - Open Systems Interconnection - The
                Directory: Authentication Framework, March 2000.

   [PKIXALGS]   Bassham, L., Polk, W. and R. Housley, "Algorithms and
                Identifiers for the Internet X.509 Public Key
                Infrastructure Certificate and Certificate Revocation
                Lists (CRL) Profile", RFC 3279, April 2002.

   [CERTSTORE] P. Gutmann, "Internet X.509 Public Key Infrastructure
                Operational Protocols: Certificate Store Access via
                HTTP", Work in Progress, August 2004.

Authors' Addresses

   Matt Cooper
   Orion Security Solutions, Inc.
   1489 Chain Bridge Rd, Ste. 300
   McLean, VA 22101, USA

   Phone: +1-703-917-0060
   EMail: mcooper@orionsec.com

   Yuriy Dzambasow
   A&N Associates, Inc.
   999 Corporate Blvd Ste. 100
   Linthicum, MD 21090, USA

   Phone: +1-410-859-5449 x107
   EMail: yuriy@anassoc.com

   Peter Hesse
   Gemini Security Solutions, Inc.
   4451 Brookfield Corporate Dr. Ste. 200
   Chantilly, VA 20151, USA

   Phone: +1-703-378-5808 x105
   EMail: pmhesse@geminisecurity.com

   Susan Joseph
   Van Dyke Technologies
   6716 Alexander Bell Drive
   Columbia, MD 21046

   EMail: susan.joseph@vdtg.com

   Richard Nicholas
   BAE Systems Information Technology
   141 National Business Parkway, Ste. 210
   Annapolis Junction, MD 20701, USA

   Phone: +1-301-939-2722
   EMail: richard.nicholas@it.baesystems.com

Full Copyright Statement

Intellectual Property

Acknowledgement