Draft Special Publication 800-XXX

NIST Recommendation for X.509 Path Validation

Version 0.5

May 3, 2004

David A. Cooper
W. Timothy Polk

Draft: NIST Recommendation for X.509 Path Validation

1. INTRODUCTION

A Public Key Infrastructure (PKI) binds cryptographic public keys to physical entities through digital certificates. A PKI includes components that issue digital certificates and distribute certificate status information. PKI users select one or more certificate issuers as trust anchors, and establish security services based on certificates that may be validated using one of their trust anchors.

One critical component of PKI clients is the X.509 Path Validation Module (PVM). This module determines whether a certificate may be trusted for use by a particular application. Section 6 of the "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile" (RFC 3280) [3] describes one path validation algorithm. Client implementations that conform to RFC 3280 are required to implement a path validation algorithm functionally equivalent to the algorithm presented in Section 6. That is, the PVM is considered as a black box – if the PVM provides the same answer as the algorithm presented in Section 6 algorithm, it conforms to RFC 3280.

The path validation algorithm uses the information contained in certificates and CRLs to determine whether a certificate may be trusted for a particular application. The certificate and CRL information may appear as fields in the base certificate or CRL, or may appear in a number of standard certificate and CRL extensions. Implementations of the path validation algorithm are required to recognize and process all the fields in the base certificate and CRL. Implementations of the path validation algorithm are not required to recognize all the standard extensions, but must process recognized extensions according to the algorithm. The algorithm also establishes rules for processing unrecognized extensions.

Certificate and CRL extensions encountered in path validation depend upon the architecture and complexity of the public key infrastructure. For an overview of PKI architectures and their implications for path validation, see [1]. In the Federal government, PKI-enabled applications may expect to encounter two different levels of complexity: applications that serve users associated with a single agency will perform path validation in an *Enterprise PKI*; applications that serve users from multiple agencies or outside the Federal Government will perform path validation in a *Bridge-enabled PKI*.

A suite of tests, PKITS, has been developed for conformance testing of Path Validation Modules [2]. These tests may be used to objectively measure the conformance of implementations to the path validation algorithm as defined by RFC 3280.

Agencies are strongly encouraged to deploy path validation software that implements the Bridge-enabled functionality to support multi-agency applications. At a minimum, path validation software deployed by federal agencies must support the functionality specified for Enterprise PKIs. Agencies may determine whether or not additional functionality is required to meet agency requirements. Software that has demonstrated conformance using the PKITS test suite should be preferred over untested applications.

1.1. Scope

The scope of this document is limited to functional requirements for the path validation module. This specification assumes the use of a FIPS 140-1 or FIPS 140-2 Validated Module for all cryptographic operations. (For more information on Validated Modules, see http://csrc.nist.gov/cryptval). This specification does not define requirements for certification path building. NIST plans to publish a requirements document for path building, along with a corresponding test suite, in the future.

1.2. Document Overview

This document consists of six major sections and a normative appendix:

Section 2 provides an overview of path validation and explains how a path validation module fits into an overall system.

Section 3 identifies functional groupings of certificate and CRL extensions that are required to support Enterprise PKI architectures.

Section 4 identifies functional groupings of certificate and CRL extensions that are required to support Bridge-enabled PKI architectures.

Section 5 provides a naming scheme for path validation modules based on the functionality implemented by a PVM.

Section 6 contains the bibliographic references.

Appendix A identifies the subset of PKITS required to test a PVM based on the functionality implemented by the PVM.

2. Background

2.1. X.509 Certificates and CRLs

X.509 public key certificates are data structures that bind public key values to *subjects* (i.e., users or devices). The binding is asserted by having a CA, the *issuer*, digitally sign each certificate. A certificate has a limited valid lifetime which is indicated in its signed contents. Because a certificate's signature and timeliness can be independently checked by a certificate-using client, certificates can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

X.509 CRLs are data structures that list unexpired certificates that have been revoked by their issuer. The integrity of the list is protected by having the issuer digitally sign each CRL. In general, CRLs cover all unexpired certificates issued by the CA that signed the CRL. The CRL is issued periodically; the next expected update is indicated in its signed contents. Like certificates, CRLs can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

2.2. X.509 Certification Path Validation

PKI-enabled application protocols rely upon the trustworthiness of certificates to achieve security in their protocols. For example, the S/MIME secure electronic mail protocol depends upon X.509 certificates to verify digital signatures and exchange symmetric keying material. The binding between the subject name and the subject public key must be reliable, or the mail may not be secure! To ensure that certificates are trustworthy, applications request that the PVM determine if a certificate is (1) valid and (2) appropriate for the requesting application.

The X.509 certificate that the application wishes to use is called the *certificate of interest*, or the *end entity certificate*. The PVM confirms the trustworthiness of the certificate of interest by validating a sequence of certificates from a trust anchor or trust-point to the certificate of interest. This sequence of certificates is known as a certification path. Paths may be obtained from a Certification Path Constructor (CPC), or paths may be provided by other parties (via the application protocol). The PVM may process multiple paths before finding an acceptable path.

Section 6 of RFC 3280 describes a comprehensive algorithm for X.509 path processing, covering all the standard certificate and CRL extensions described in RFC 3280. A conformant implementation of a path validation module MUST include an X.509 path processing procedure that is functionally equivalent to the external behavior of this algorithm. Conformant implementations are *not* required to implement the algorithm as specified.

The trust anchor is an input to the algorithm. There is no requirement that the same trust anchor be used to validate all certification paths. Different trust anchors may be used to validate different paths. In particular, different applications may request validation with respect to different trust anchors.

The path validation process verifies, among other things, that a prospective certification path (a sequence of *n* certificates) satisfies the following conditions:

a) for all x in {1, ..., n-1}, the subject of certificate x is the issuer of certificate x+1;

b) certificate 1 is issued by the trust anchor;

c) certificate n is the certificate to be validated; and

d) for all x in {1, ..., n}, the certificate was valid at the time in question.

PVMs are commonly implemented so that the trust anchor is provided in the form of a self-signed certificate. This self-signed certificate is not included as part of the prospective certification path. Information about trust anchors are provided as inputs to the certification path validation algorithm.

A particular certification path may not be appropriate for all applications. In particular, the *certificate policy* that governs issuing certificates and maintaining their status can vary widely. The path validation process determines the set of certificate policies that are valid for this path, based on the certificatePolicies extension, policyMappings extension,

policyConstraints extension, and inhibitAnyPolicy extension. Applications may request validation of the certificate of interest with respect to an acceptable policy set.

2.3. PATH VALIDATION MODULE SPECIFICATION

Figure 1 shows the major functional components for an X.509 Path Validation System. The Path Validation Module is the core component but requires support from both the cryptographic module (to verify digital signatures) and path building module (to construct the set of certificates and CRLs). Note that these are functional components only, and do not imply any restrictions upon the architecture of a particular module or modules.

The scope of this document is limited to functional requirements for the path validation module. This specification assumes the use of a FIPS 140-1 or FIPS 140-2 Validated Module for all cryptographic operations. (For more information on Validated Modules, see http://csrc.nist.gov/cryptval). This specification does not define requirements for certification path building. NIST plans to publish a requirements document for path building, along with a corresponding test suite, in the future.

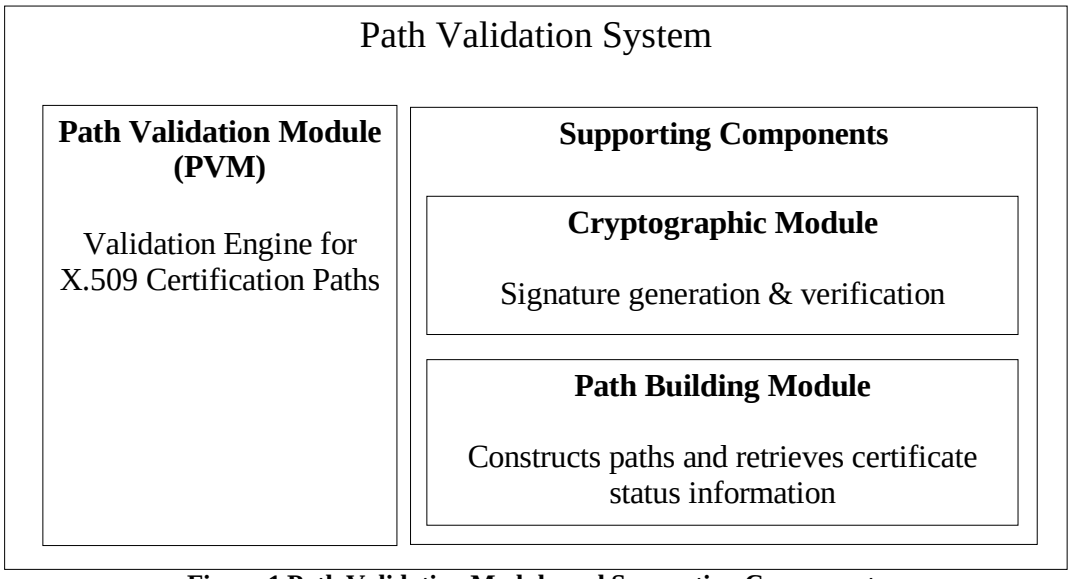

Figure 1 Path Validation Module and Supporting Components

The X.509 Path Validation Module will validate certificates with respect to one or more trusted CAs and verify the path with respect to a set of initial conditions and constraints imposed in the certificates themselves.

For the path validation module to operate properly, there are additional functional capabilities that must be provided by other functional components. This specification identifies two functional components providing these capabilities:

- Cryptographic module; and
- Path building module.

The cryptographic module functional component verifies digital signatures on certificates and CRLs. For Federal agencies, FIPS PUB 140 requires use of FIPS 140-1 or FIPS 140-2 Validated Modules. (For more information on Validated Modules, see http://csrc.nist.gov/cryptval).

The path building component retrieves certificates and CRLs to create a certification path linking a trust anchor and the certificate of interest. In some applications, certification paths are provided via the application, so the path building module may be omitted.

3. Enterprise Path Validation Module Functional Requirements

The Enterprise PVM component processes X.509 certification paths composed of X.509 v3 certificates and X.509 v2 CRLs. The PVM component MUST support the following features:

- Name chaining;
- Signature chaining;
- Certificate validity;
- Key usage, basic constraints, and certificate policies certificate extensions;
- Full CRLs; and
- CRLs segmented on names.

An Enterprise PVM may support additional path processing functionality.

3.1. Basic Path Processing

1. The PVM shall verify that digital signatures and public keys in the certification path chain in accordance with RFC 3280, using RSA PKCS#1 with SHA-1 in accordance with RFC 3279[1]. (That is, the PVM shall verify that the RSA with SHA-1 signature on each certificate in the path verifies using the RSA public key in the preceding certificate, and the RSA with SHA-1 signature on the first certificate in the path verifies using a trust anchor's RSA public key.)
2. The PVM shall verify that issuer and subject names in certification paths chain in accordance with RFC 3280. (That is, the PVM shall verify that the issuer of each certificate in the path was the subject of the preceding certificate, and the issuer of the first certificate in the path is the name associated with the trust anchor public key)

1 NIST is recommending that agencies transition to signing certificates and CRLs using RSA with SHA-256 by January 1, 2009. It is therefore recommended that path validation modules be able to verify digital signatures using RSA with SHA-256 in addition to RSA with SHA-1.

3. The PVM shall verify that subject of each intermediate version 3 certificate is a CA by verifying that the certificate contains the basicConstraints extension with cA asserted. The PVM may either:
    a. not accept version 1 or version 2 certificates as intermediate certificates; or
    b. verify that the subject of each intermediate version 1 or 2 certificate is a CA through local mechanisms.
3. The PVM shall verify that each intermediate certificate containing the keyUsage extension asserts the keyCertSign bit.
4. The PVM shall verify that each certificate that contains a keyUsage extension asserts the cRLSigning bit if the subject public key is used to validate a CRL.
5. The PVM shall verify that each certificate whose subject public key is used to validate a CRL is the end certificate in a valid certificate chain.
6. The PVM shall verify that the length of the chain does not violate constraints imposed by the issuer of any certificate in the chain by processing the pathLenConstraint field in the basicConstraints extension in accordance with RFC 3280.

3.2. Basic Policy Processing

1. The PVM shall be capable of processing the certificatePolicies extension, in accordance with RFC 3280, to determine the set of policies under which the path is valid.
2. The PVM shall recognize and process the requireExplicitPolicy field in the policyConstraints extension in accordance with RFC 3280.
3. The PVM shall include the capability to specify a set of acceptable policies, and indicate the path may only be accepted if valid under one of those policies. [Note: The capability may be supplied as a configuration setting, or as a parameter in an API call.]

3.3. Revocation Status

1. The PVM shall verify that each certificate in the path is valid at the specified time (e.g., the current time) in accordance with RFC 3280.
2. The PVM shall determine the revocation status of intermediate and end certificates in all paths. The PVM shall reject paths that include revoked certificates. The PVM shall return a warning or a rejection for paths containing certificates whose revocation status cannot be determined.
3. The PVM shall be capable of processing full CRLs issued by the certificate issuer to determine certificate status, in accordance with RFC 3280.

4. The PVM shall be capable of processing segmented CRLs, where the correct CRL is identified by matching the contents of the full names in the distributionPointName field in cRLDistributionPoints certificate extension and the issuingDistributionPoint CRL extension in accordance with RFC 3280.
5. The PVM shall process CRLs segmented by certificate type (e.g., CA certificate, user certificate, etc.) as specified in the issuingDistributionPoint CRL extension, in accordance with RFC 3280.

3.4. Critical Extensions

1. The PVM shall reject chains that include certificates containing unrecognized critical certificate extensions or critical certificate extensions with unrecognized fields.
2. The PVM shall not use CRLs containing unrecognized critical extensions or critical extensions with unrecognized fields as the basis for determining that a certificate is valid. However, the PVM shall assume that any certificates identified on such CRLs are revoked.

4. Bridge-Enabled Path Validation Module Requirements

In addition to meeting the requirements of an Enterprise PVM, a Bridge-enabled Path Validation Module must be capable of processing the types of certificate extensions that are typically found in CA certificates that cross enterprise boundaries. This section defines three packages of requirements for Bridge-enabled PVMs: Name Constraints, Policy Mapping, and anyPolicy. A PVM that meets all of the requirements for each of these packages, in addition to all of the requirements in section 3, is considered to be Bridge-enabled.

4.1. Name Constraints Processing Requirements

1. The PVM shall process name constraints as described in RFC 3280 for the following name forms:
   a) distinguished names, and
   b) RFC 822 names.

4.2. Policy Mapping Requirements

1. The PVM shall recognize and process the policyMappings extension, in accordance with RFC 3280.
2. The PVM shall recognize and process the inhibitPolicyMapping field in the policyConstraints extension, in accordance with RFC 3280.

3. The PVM shall include the capability to specify whether policy mapping is permitted or inhibited as an initial condition for path validation. [Note: The capability may be supplied as a configuration setting, or as a parameter in an API call.]

4.3. anyPolicy Processing Requirements

1. The PVM shall recognize and process the special policy anyPolicy in accordance with RFC 3280 when it appears in the certificate policies extension.
2. The PVM shall recognize and process the inhibitAnyPolicy certificate extension in accordance with RFC 3280.
3. The PVM shall include the capability to specify whether the special policy anyPolicy may be processed as an initial condition for path validation. [Note: The capability may be supplied as a configuration setting, or as a parameter in an API call.]

5. Supplementary Path Validation Functionality

The preceding sections identified common PKI requirements for enterprise PKIs and PKIs that cross organizational boundaries. Specific PKIs may use additional features, imposing additional requirements on PVMs. This document defines four independent functional packages in addition to the three packages defined in section 4. Three of the packages support additional mechanisms for certificate revocation,. The fourth package supports the use of the DSA cryptographic algorithm to sign certificates and CRLs.

The four supplementary path validation packages are:

1. **Indirect CRLs**: The ability to process indirect CRLs, including the ability to process:
   a) the cRLIssuer field of the cRLDistributionPoints extension;
   b) the indirectCRL field of the issuingDistributionPoint extension; and
   c) the certificateIssuer CRL entry extension.
2. **Reasons**: The ability to process CRLs that have been segmented by reason code, including the ability to process the onlySomeReasons field of the issuingDistributionPoint extension.
3. **Delta-CRLs**: The ability to process delta-CRLs.
4. **DSA**: The ability to process DSA public keys and signatures, including public keys in which the parameters have been inherited.

6. Path Validation Module Naming Scheme

To assist in comparison and succinct description of functional requirements for PVMs, this specification establishes a naming scheme that identifies PVMs that implement all of the functionality specified for Enterprise PVMs and, optionally, the functional packages specified in sections 4 and 5. While this naming scheme does not factor in partially supported packages or functionality not specified in a package, PVMs may implement any set of functionality as long as all functionality specified in section 3 is implemented and any functionality that is implemented is implemented correctly.

The following table is supplied to facilitate name construction, and is used as follows: working from the top to the bottom, find the first row where all required packages (indicated by an X) are implemented. Use the name from the topmost row that applies, filling in the optional packages that have been fully implemented as specified in the selected row.

Title of Protection Profile	Name Constraints	Policy Mapping	anyPolicy	Indirect CRLs	Reasons	Delta-CRLs	DSA
Bridge-enabled PVM with Advanced CRLs [, f, and g]	X	X	X	X	X	*f*	*g*
Bridge-enabled PVM [with d, e, f, and g]	X	X	X	*d*	*e*	*f*	*g*
Enterprise PVM with Advanced CRLs[, a, b, c, f, and g]	*a*	*b*	*c*	X	X	*f*	*g*
Enterprise PVM [with a, b, c, d, e, f, and g]	*a*	*b*	*c*	*d*	*e*	*f*	*g*

X: required package for this naming scheme

Optional packages:

a: Name Constraints

b: Policy Mapping

c: anyPolicy

d: Indirect CRLs

e: Reasons

f: Delta-CRLs

g: DSA

Example #1: An Path Validation Module that implements all of the functionality specified in sections 3 and 4 in addition to the requirements of the DSA package would be a *Bridge-enabled PVM with DSA*, based on the second row in the table above.

Example #2: An Path Validation Module that implements all of the functionality specified in section 3 in addition to the requirements for the Name Constraints and Delta-CRLs packages would be an *Enterprise PVM with Delta-CRLs and Name Constraints*, based on the fourth row in the table.

7. References

[1] William T. Polk, Nelson E. Hastings, and Ambarish Malpani. Public Key Infrastructures that Satisfy Security Goals. *IEEE Internet Computing*, 7(4): 60 – 67, July-August, 2003.

[2] Public Key Interoperability Test Suite (PKITS): Certification Path Validation, Version 1.0, May XX, 2004. http://csrc.nist.gov/pki/testing/x509paths.html.

[3] Russel Housley, Tim Polk, Warwick Ford, and David Solo. Internet Public Key Infrastructure: *X.509 Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280, April 2002.

A. Conformance Testing

The Public Key Interoperability Test suite (PKITS) [2] can be used to determine whether a Path Validation Module has been implemented in conformance with RFC 3280. Conformance testing of any given PVM will not require running every test in PKITS. The specific set of tests that need to be run for any given PVM, and the expected outcome for each test, will depend on what functionality that PVM has implemented. For example, in a test involving name constraints, a PVM that implements name constraints may be expected to process the name constraints extension and determine that the path is valid whereas a PVM that does not implement name constraints may be expected to reject the certification path as including a certificate with an unrecognized, critical extension.

In order to determine which tests to run and the expected outcome for each test, it is necessary to know whether or not the PVM being tested implements each of the following:

1. DSA signature verification: The ability to process DSA public keys and signatures when each certificate with a DSA subject public key includes parameters.
2. DSA parameter inheritance: The ability to process DSA public keys and signatures, including public keys in which the parameters have been inherited.
3. directoryName: The ability to process nameConstraints extensions that include a directoryName as a permitted and/or excluded subtree and to determine whether subsequent certificates satisfy the specified constraints.
4. rfc822Name: The ability to process nameConstraints extensions that include an rfc822Name as a permitted and/or excluded subtree and to determine whether subsequent certificates satisfy the specified constraints.

5. dNSName: The ability to process nameConstraints extensions that include a dNSName as a permitted and/or excluded subtree and to determine whether subsequent certificates satisfy the specified constraints.
6. uniformResourceIdentifier: The ability to process nameConstraints extensions that include a uniformResourceIdentifier as a permitted and/or excluded subtree and to determine whether subsequent certificates satisfy the specified constraints.
7. onlySomeReasons: The ability to process CRLs that have been segmented by reason code, including the ability to process the onlySomeReasons field of the issuingDistributionPoint extension.
8. nameRelativeToIssuer: The ability to process cRLDistributionPoints extensions and issuingDistributionPoint extensions that include a distributionPoint specified as nameRelativeToIssuer.
9. indirect CRLs: The ability to process indirect CRLs, including the ability to process:
   d) the cRLIssuer field of the cRLDistributionPoints extension;
   e) the indirectCRL field of the issuingDistributionPoint extension; and
   f) the certificateIssuer CRL entry extension.
10. delta-CRLs: The ability to process delta-CRLs.
11. anyPolicy OID: The ability to process the special policy anyPolicy in accordance with RFC 3280 when it appears in the certificate policies extension.
12. inhibitAnyPolicy: The ability to process the inhibitAnyPolicy extension.
13. initial-inhibit-any-policy: The ability to specify whether the special policy anyPolicy may be processed as an initial condition for path validation. [Note: The capability may be supplied as a configuration setting, or as a parameter in an API call.]
14. policyMappings: The ability to process the policyMappings extension.
15. inhibitPolicyMapping: The ability to process the inhibitPolicyMapping field of the policyConstraints extension.
16. initial-inhibit-policy-mapping: The ability to specify whether policy mapping is permitted or inhibited as an initial condition for path validation. [Note: The capability may be supplied as a configuration setting, or as a parameter in an API call.]

At a minimum, it is assumed that all PVMs can implement all of the functionality specified in section 3 for Enterprise PVMs.

Below is a list of each test in PKITS along with an indication of which applications need to run the test, based on the set of functionality implemented by the applications. Unless explicitly stated otherwise, the expected result for a PVM for a test that is to be run is the outcome specified in PKITS for that test.

Test	*Applications that need to run test*
4.1.1 Valid Signatures Test1	All.
4.1.2 Invalid CA Signature Test2	All.
4.1.3 Invalid EE Signature Test3	All.
4.1.4 Valid DSA Signatures Test4	All. Applications that can not verify DSA signatures must reject the path.
4.1.5 Valid DSA Parameter Inheritance Test5	Run only if application can verify DSA signatures and parameter inheritance.
4.1.6 Invalid DSA Signature Test6	Run only if application can verify DSA signatures
4.2.1 Invalid CA notBefore Date Test1	All.
4.2.2 Invalid EE notBefore Date Test2	All.
4.2.3 Valid pre2000 UTC notBefore Date Test3	All.
4.2.4 Valid GeneralizedTime notBefore Date Test4	All.
4.2.5 Invalid CA notAfter Date Test5	All.
4.2.6 Invalid EE notAfter Date Test6	All.
4.2.7 Invalid pre2000 UTC EE notAfter Date Test7	All.
4.2.8 Valid GeneralizedTime notAfter Date Test8	All.
4.3.1 Invalid Name Chaining EE Test1	All.
4.3.2 Invalid Name Chaining Order Test2	All.
4.3.3 Valid Name Chaining Whitespace Test3	All.
4.3.4 Valid Name Chaining Whitespace Test4	All.

Test	*Applications that need to run test*
4.3.5 Valid Name Chaining Capitalization Test5	All.
4.3.6 Valid Name Chaining UIDs Test6	All.
4.3.7 Valid RFC3280 Mandatory Attribute Types Test7	This test does not need to be run.
4.3.8 Valid RFC3280 Optional Attribute Types Test8	This test does not need to be run.
4.3.9 Valid UTF8String Encoded Names Test9	All.
4.3.10 Valid Rollover from PrintableString to UTF8String Test10	This test does not need to be run.
4.3.11 Valid UTF8String Case Insensitive Match Test11	This test does not need to be run.
4.4.1 Missing CRL Test1	All.
4.4.2 Invalid Revoked CA Test2	All.
4.4.3 Invalid Revoked EE Test3	All.
4.4.4. Invalid Bad CRL Signature Test4	All.
4.4.5 Invalid Bad CRL Issuer Name Test5	All.
4.4.6 Invalid Wrong CRL Test6	All.
4.4.7 Valid Two CRLs Test7	All.
4.4.8 Invalid Unknown CRL Entry Extension Test8	All.
4.4.9 Invalid Unknown CRL Extension Test9	All.
4.4.10 Invalid Unknown CRL Extension Test10	All.
4.4.11 Invalid Old CRL nextUpdate Test11	All.

Test	*Applications that need to run test*
4.4.12 Invalid pre2000 CRL nextUpdate Tesst12	All.
4.4.13 Valid GeneralizedTime CRL nextUpdate Test13	All.
4.4.14 Valid Negative Serial Number Test14	All.
4.4.15 Invalid Negative Serial Number Test15	All.
4.4.16 Valid Long Serial Number Test16	All.
4.4.17 Valid Long Serial Number Test17	All.
4.4.18 Invalid Long Serial Number Test18	All.
4.4.19 Valid Separate Certificate and CRL Keys Test19	All.
4.4.20 Invalid Separate Certificate and CRL Keys Test20	All.
4.4.21 Invalid Separate Certificate and CRL Keys Test21	All.
4.5.1 Valid Basic Self-Issued Old With New Test1	All.
4.5.2 Invalid Basic Self-Issued Old With New Test2	All.
4.5.3 Valid Basic Self-Issued New With Old Test3	All.
4.5.4 Valid Basic Self-Issued New With Old Test4	All.
4.5.5 Invalid Basic Self-Issued New With Old Test5	All.
4.5.6 Valid Basic Self-Issued CRL Signing Key Test6	All.

Test	*Applications that need to run test*
4.5.7 Invalid Basic Self-Issued CRL Signing Key Test7	All.
4.5.8 Invalid Basic Self-Issued CRL Signing Key Test8	All.
4.6.1 Invalid Missing basicConstraints Test1	All.
4.6.2 Invalid cA False Test2	All.
4.6.3 Invalid cA False Test3	All.
4.6.4 Valid basicConstraints Not Critical Test4	All.
4.6.5 Invalid pathLenConstraint Test5	All.
4.6.6 Invalid pathLenConstraint Test6	All.
4.6.7 Valid pathLenConstraint Test7	All.
4.6.8 Valid pathLenConstraint Test8	All.
4.6.9 Invalid pathLenConstraint Test9	All.
4.6.10 Invalid pathLenConstraint Test10	All.
4.6.11 Invalid pathLenConstraint Test11	All.
4.6.12 Invalid pathLenConstraint Test12	All.
4.6.13 Valid pathLenConstraint Test13	All.
4.6.14 Valid pathLenConstraint Test14	All.
4.6.15 Valid Self-Issued pathLenConstraint Test15	All.

Test	*Applications that need to run test*
4.6.16 Invalid Self-Issued pathLenConstraint Test16	All.
4.6.17 Valid Self-Issued pathLenConstraint Test17	All.
4.7.1 Invalid keyUsage Critical keyCertSign False Test1	All.
4.7.2 Invalid keyUsage Not Critical keyCertSign False Test2	All.
4.7.3 Valid keyUsage Not Critical Test3	All.
4.7.4 Invalid keyUsage Critical cRLSign False Test4	All.
4.7.5 Invalid keyUsage Not Critical cRLSign False Test5	All.
4.8.1 All Certificates Same Policy Test1, subtest 1	Run if application can be configured as specified (i.e., if *initial-policy-set* can be *any-policy* when *initial-explicit-policy* is set).
4.8.1 All Certificates Same Policy Test1, subtest 2	All.
4.8.1 All Certificates Same Policy Test1, subtest 3	All.
4.8.1 All Certificates Same Policy Test1, subtest 4	All.
4.8.2 All Certificates No Policies Test2, subtest 1	All.
4.8.2 All Certificates No Policies Test2, subtest 2	All. (*initial-policy-set* may be set to {NIST-test-policy-1, NIST-test-policy-2, NIST-test-policy-3, NIST-test-policy-4, NIST-test-policy-5, NIST-test-policy-6} if it can not be set to *any-policy*).
4.8.3 Different Policies Test3, subtest 1	All.
4.8.3 Different Policies Test3, subtest 2	Run if application can be configured as specified (i.e., if *initial-policy-set* can be *any-policy* when *initial-explicit-policy* is set).

Test	*Applications that need to run test*
4.8.3 Different Policies Test3, subtest 3	All.
4.8.4 Different Policies Test4	All.
4.8.5 Different Policies Test5	All.
4.8.6 Overlapping Policies Test6, subtest 1	All.
4.8.6 Overlapping Policies Test6, subtest 2	All.
4.8.6 Overlapping Policies Test6, subtest 3	All.
4.8.7 Different Policies Test7	All.
4.8.8 Different Policies Test8	All.
4.8.9 Different Policies Test9	All.
4.8.10 All Certificates Same Policies Test10, subtest 1	All.
4.8.10 All Certificates Same Policies Test10, subtest 2	All.
4.8.10 All Certificates Same Policies Test10, subtest 3	All.
4.8.11 All Certificates AnyPolicy Test11, subtest 1	This subtest does not need to be run.
4.8.11 All Certificates AnyPolicy Test11, subtest 2	Run if application can process the special policy **anyPolicy**.
4.8.12 Different Policies Test12	All.
4.8.13 All Certificates Same Policies Test13, subtest 1	All.
4.8.13 All Certificates Same Policies Test13, subtest 2	All.
4.8.13 All Certificates Same Policies Test13, subtest 3	All.
4.8.14 AnyPolicy Test14, subtest 1	Run if application can process the special policy **anyPolicy**.

Test	*Applications that need to run test*
4.8.14 AnyPolicy Test14, subtest 2	Run if application can process the special policy **anyPolicy.**
4.8.15 User Notice Qualifier Test15	This test does not need to be run.
4.8.16 User Notice Qualifier Test16	This test does not need to be run.
4.8.17 User Notice Qualifier Test17	This test does not need to be run.
4.8.18 User Notice Qualifier Test18, subtest 1	This subtest does not need to be run.
4.8.18 User Notice Qualifier Test18, subtest 2	This subtest does not need to be run.
4.8.19 User Notice Qualifier Test19	This test does not need to be run.
4.8.20 CPS Pointer Qualifier Test20	All. Test should be run with *initial-explicit-policy* set (*initial-policy-set* may be set to {NIST-test-policy-1, NIST-test-policy-2, NIST-test-policy-3, NIST-test-policy-4, NIST-test-policy-5, NIST-test-policy-6} if it can not be set to *any-policy*).
4.9.1 Valid RequireExplicitPolicy Test1	All.
4.9.2 Valid RequireExplicitPolicy Test2	All.
4.9.3 Invalid RequireExplicitPolicy Test3	All.
4.9.4 Valid RequireExplicitPolicy Test4	All.
4.9.5 Invalid RequireExplicitPolicy Test5	All.
4.9.6 Valid Self-Issued requireExplicitPolicy Test6	All.
4.9.7 Invalid Self-Issued requireExplicitPolicy Test7	All.

Test	*Applications that need to run test*
4.9.8 Invalid Self-Issued requireExplicitPolicy Test8	All.
4.10.1 Valid Policy Mapping Test1, subtest 1	All. Applications that can not process the **policyMappings** extension should reject the path (When testing an application that does not process the **policyMappings** extension, the default settings should be used (i.e., *initial-policy-set = any-policy*)).
4.10.1 Valid Policy Mapping Test1, subtest 2	Run if application can process the **policyMappings** extension.
4.10.1 Valid Policy Mapping Test1, subtest 3	Run if *initial-policy-mapping-inhibit* can be set.
4.10.2 Invalid Policy Mapping Test2, subtest 1	Run if application can process the **policyMappings** extension.
4.10.2 Invalid Policy Mapping Test2, subtest 2	Run if *initial-policy-mapping-inhibit* can be set.
4.10.3 Valid Policy Mapping Test3, subtest 1	Run if application can process the **policyMappings** extension.
4.10.3 Valid Policy Mapping Test3, subtest 2	Run if application can process the **policyMappings** extension.
4.10.4 Invalid Policy Mapping Test4	Run if application can process the **policyMappings** extension.
4.10.5 Valid Policy Mapping Test5, subtest 1	Run if application can process the **policyMappings** extension.
4.10.5 Valid Policy Mapping Test5, subtest 2	Run if application can process the **policyMappings** extension.
4.10.6 Valid Policy Mapping Test6, subtest 1	Run if application can process the **policyMappings** extension.
4.10.6 Valid Policy Mapping Test6, subtest 2	Run if application can process the **policyMappings** extension.
4.10.7 Invalid Mapping From anyPolicy Test7	Run if application can process the **policyMappings** extension and the special policy **anyPolicy**.
4.10.8 Invalid Mapping To anyPolicy Test8	Run if application can process the **policyMappings** extension and the special policy **anyPolicy**.

Test	*Applications that need to run test*
4.10.9 Valid Policy Mapping Test9	This test does not need to be run.
4.10.10 Invalid Policy Mapping Test10	Run if application can process the **policyMappings** extension and the special policy **anyPolicy**.
4.10.11 Valid Policy Mapping Test11	Run if application can process the **policyMappings** extension and the special policy **anyPolicy**.
4.10.12 Valid Policy Mapping Test12, subtest 1	Run if application can process the **policyMappings** extension. It is irrelevant whether the user notice is displayed.
4.10.12 Valid Policy Mapping Test12, subtest 2	Run if application can process the **policyMappings** extension and the special policy **anyPolicy**. It is irrelevant whether the user notice is displayed.
4.10.13 Valid Policy Mapping Test13	This test does not need to be run.
4.10.14 Valid Policy Mapping Test14	This test does not need to be run.
4.11.1 Invalid inhibitPolicyMapping Test1	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.2 Valid inhibitPolicyMapping Test2	All. Applications that can not process the **inhibitPolicyMapping** field in the **policyConstraints** extension should reject the path.
4.11.3 Invalid inhibitPolicyMapping Test3	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.4 Valid inhibitPolicyMapping Test4	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.5 Invalid inhibitPolicyMapping Test5	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.6 Invalid inhibitPolicyMapping Test6	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.7 Valid Self-Issued inhibitPolicyMapping Test7	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.8 Invalid Self-Issued inhibitPolicyMapping Test8	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.

Test	*Applications that need to run test*
4.11.9 Invalid Self-Issued inhibitPolicyMapping Test9	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.10 Invalid Self-Issued inhibitPolicyMapping Test10	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.11.11 Invalid Self-Issued inhibitPolicyMapping Test11	Run if application can process the **inhibitPolicyMapping** field in the **policyConstraints** extension.
4.12.1 Invalid inhibitAnyPolicy Test1	Run if application can process the **inhibitAnyPolicy** extension.
4.12.2 Valid inhibitAnyPolicy Test2	All. Applications that can not process the **inhibitAnyPolicy** extension should reject the path.
4.12.3 inhibitAnyPolicy Test3, subtest 1	Run if application can process the **inhibitAnyPolicy** extension.
4.12.3 inhibitAnyPolicy Test3, subtest 2	Run if *initial-inhibit-any-policy* can be set.
4.12.4 Invalid inhibitAnyPolicy Test4	Run if application can process the **inhibitAnyPolicy** extension.
4.12.5 Invalid inhibitAnyPolicy Test5	Run if application can process the **inhibitAnyPolicy** extension.
4.12.6 Invalid inhibitAnyPolicy Test6	Run if application can process the **inhibitAnyPolicy** extension.
4.12.7 Valid Self-Issued inhibitAnyPolicy Test7	Run if application can process the **inhibitAnyPolicy** extension.
4.12.8 Invalid Self-Issued inhibitAnyPolicy Test8	Run if application can process the **inhibitAnyPolicy** extension.
4.12.9 Valid Self-Issued inhibitAnyPolicy Test9	Run if application can process the **inhibitAnyPolicy** extension.
4.12.10 Invalid Self-Issued inhibitAnyPolicy Test10	Run if application can process the **inhibitAnyPolicy** extension.
4.13.1 Valid DN nameConstraints Test1	All. Applications that can not process name constraints for the **directoryName** form should reject the path.

Test	*Applications that need to run test*
4.13.2 Invalid DN nameConstraints Test2	Run if application can process name constraints for the **directoryName** form.
4.13.3 Invalid DN nameConstraints Test3	Run if application can process name constraints for the **directoryName** form.
4.13.4 Valid DN nameConstraints Test4	Run if application can process name constraints for the **directoryName** form.
4.13.5 Valid DN nameConstraints Test5	Run if application can process name constraints for the **directoryName** form.
4.13.6 Valid DN nameConstraints Test6	Run if application can process name constraints for the **directoryName** form.
4.13.7 Invalid DN nameConstraints Test7	Run if application can process name constraints for the **directoryName** form.
4.13.8 Invalid DN nameConstraints Test8	Run if application can process name constraints for the **directoryName** form.
4.13.9 Invalid DN nameConstraints Test9	Run if application can process name constraints for the **directoryName** form.
4.13.10 Invalid DN nameConstraints Test10	Run if application can process name constraints for the **directoryName** form.
4.13.11 Valid DN nameConstraints Test11	Run if application can process name constraints for the **directoryName** form.
4.13.12 Invalid DN nameConstraints Test12	Run if application can process name constraints for the **directoryName** form.
4.13.13 Invalid DN nameConstraints Test13	Run if application can process name constraints for the **directoryName** form.
4.13.14 Valid DN nameConstraints Test14	Run if application can process name constraints for the **directoryName** form.
4.13.15 Invalid DN nameConstraints Test15	Run if application can process name constraints for the **directoryName** form.
4.13.16 Invalid DN nameConstraints Test16	Run if application can process name constraints for the **directoryName** form.
4.13.17 Invalid DN nameConstraints Test17	Run if application can process name constraints for the **directoryName** form.

Test	*Applications that need to run test*
4.13.18 Valid DN nameConstraints Test18	Run if application can process name constraints for the **directoryName** form.
4.13.19 Valid Self-Issued DN nameConstraints Test19	Run if application can process name constraints for the **directoryName** form.
4.13.20 Invalid Self-Issued DN nameConstraints Test20	Run if application can process name constraints for the **directoryName** form.
4.13.21 Valid RFC822 nameConstraints Test21	All. Applications that can not process name constraints for the **rfc822Name** form should reject the path.
4.13.22 Invalid RFC822 nameConstraints Test22	Run if application can process name constraints for the **rfc822Name** form.
4.13.23 Valid RFC822 nameConstraints Test23	Run if application can process name constraints for the **rfc822Name** form.
4.13.24 Invalid RFC822 nameConstraints Test24	Run if application can process name constraints for the **rfc822Name** form.
4.13.25 Valid RFC822 nameConstraints Test25	Run if application can process name constraints for the **rfc822Name** form.
4.13.26 Invalid RFC822 nameConstraints Test26	Run if application can process name constraints for the **rfc822Name** form.
4.13.27 Valid DN and RFC822 nameConstraints Test27	Run if application can process name constraints for both the **directoryName** form and the **rfc822Name** form.
4.13.28 Invalid DN and RFC822 nameConstraints Test28	Run if application can process name constraints for both the **directoryName** form and the **rfc822Name** form.
4.13.29 Invalid DN and RFC822 nameConstraints Test29	Run if application can process name constraints for both the **directoryName** form and the **rfc822Name** form.
4.13.30 Valid DNS nameConstraints Test30	All. Applications that can not process name constraints for the **dNSName** form should reject the path.
4.13.31 Invalid DNS nameConstraints Test31	Run if application can process name constraints for the **dNSName** form.
4.13.32 Valid DNS nameConstraints Test32	Run if application can process name constraints for the **dNSName** form.

Test	*Applications that need to run test*
4.13.33 Invalid DNS nameConstraints Test33	Run if application can process name constraints for the **dNSName** form.
4.13.34 Valid URI nameConstraints Test34	All. Applications that can not process name constraints for the **uniformResourceIdentifier** name form should reject the path.
4.13.35 Invalid URI nameConstraints Test35	Run if application can process name constraints for the **uniformResourceIdentifier** name form.
4.13.36 Valid URI nameConstraints Test36	Run if application can process name constraints for the **uniformResourceIdentifier** name form.
4.13.37 Invalid URI nameConstraints Test37	Run if application can process name constraints for the **uniformResourceIdentifier** name form.
4.13.38 Invalid DNS nameConstraints Test38	Run if application can process name constraints for the **dNSName** form.
4.14.1 Valid distributionPoint Test1	All.
4.14.2 Invalid distributionPoint Test2	All.
4.14.3 Invalid distributionPoint Test3	All.
4.14.4 Valid distributionPoint Test4	All. Applications that can not process **cRLDistributionPoints** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer** should reject the path (or issue a warning that certificate status can not be determined).
4.14.5 Valid distributionPoint Test5	All. Applications that can not process **cRLDistributionPoints** extensions or **issuingDistributionPoint** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer** should reject the path (or issue a warning that certificate status can not be determined).
4.14.6 Invalid distributionPoint Test6	Run if application can process **cRLDistributionPoints** extensions and **issuingDistributionPoint** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer**.

Test	*Applications that need to run test*
4.14.7 Valid distributionPoint Test7	Run if application can process **issuingDistributionPoint** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer**.
4.14.8 Invalid distributionPoint Test8	Run if application can process **issuingDistributionPoint** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer**.
4.14.9 Invalid distributionPoint Test9	All.
4.14.10 Valid No issuingDistributionPoint Test10	All.
4.14.11 Invalid onlyContainsUserCerts CRL Test11	All.
4.14.12 Invalid onlyContainsCACerts CRL Test12	All.
4.14.13 Valid onlyContainsCACerts CRL Test13	All.
4.14.14 Invalid onlyContainsAttributeCerts Test14	All.
4.14.15 Invalid onlySomeReasons Test15	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.
4.14.16 Invalid onlySomeReasons Test16	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.
4.14.17 Invalid onlySomeReasons Test17	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.
4.14.18 Valid onlySomeReasons Test18	All. Applications that can not process the **onlySomeReasons** field of the **issuingDistributionPoint** extension should reject the path (or issue a warning that certificate status can not be determined).
4.14.19 Valid onlySomeReasons Test19	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.
4.14.20 Invalid onlySomeReasons Test20	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.

Test	*Applications that need to run test*
4.14.21 Invalid onlySomeReasons Test21	Run if application can process the **onlySomeReasons** field of the **issuingDistributionPoint** extension.
4.14.22 Valid IDP with indirectCRL Test22	Run if application can process indirect CRLs.
4.14.23 Invalid IDP with indirectCRL Test23	Run if application can process indirect CRLs.
4.14.24 Valid IDP with indirectCRL Test24	All. Applications that can not process indirect CRLs should reject the path (or issue a warning that certificate status can not be determined).
4.14.25 Valid IDP with indirectCRL Test25	Run if application can process indirect CRLs.
4.14.26 Invalid IDP with indirectCRL Test26	Run if application can process indirect CRLs.
4.14.27 Invalid cRLIssuer Test27	Run if application can process indirect CRLs.
4.14.28 Valid cRLIssuer Test28	Run if application can process indirect CRLs.
4.14.29 Valid cRLIssuer Test29	Run if application can process indirect CRLs and **cRLDistributionPoints** extensions that include a **distributionPoint** that is specified as **nameRelativeToCRLIssuer**.
4.14.30 Valid cRLIssuer Test30	This test does not need to be run.
4.14.31 Invalid cRLIssuer Test31	Run if application can process indirect CRLs.
4.14.32 Invalid cRLIssuer Test32	Run if application can process indirect CRLs.
4.14.33 Valid cRLIssuer Test33	Run if application can process indirect CRLs.
4.14.34 Invalid cRLIssuer Test34	Run if application can process indirect CRLs.
4.14.35 Invalid cRLIssuer Test35	Run if application can process indirect CRLs.
4.15.1 Invalid deltaCRLIndicator No Base Test1	All.
4.15.2 Valid delta-CRL Test2	Run if application can process delta-CRLs.
4.15.3 Invalid delta-CRL Test3	Run if application can process delta-CRLs.
4.15.4 Invalid delta-CRL Test4	Run if application can process delta-CRLs.
4.15.5 Valid delta-CRL Test5	Run if application can process delta-CRLs.

Test	*Applications that need to run test*
4.15.6 Invalid delta-CRL Test6	Run if application can process delta-CRLs.
4.15.7 Valid delta-CRL Test7	Run if application can process delta-CRLs.
4.15.8 Valid delta-CRL Test8	Run if application can process delta-CRLs.
4.15.9 Invalid delta-CRL Test9	Run if application can process delta-CRLs.
4.15.10 Invalid delta-CRL Test10	Run if application can process delta-CRLs.
4.16.1 Valid Unknown Not Critical Certificate Extension Test1	All.
4.16.2 Invalid Unknown Critical Certificate Extension Test2	All.