Network Working Group
Request for Comments: 5055
Category: Standards Track

T. Freeman
Microsoft Corp
R. Housley
Vigil Security
A. Malpani
Malpani Consulting Services
D. Cooper
W. Polk
NIST
December 2007

Server-Based Certificate Validation Protocol (SCVP)

Status of This Memo

Abstract

The Server-Based Certificate Validation Protocol (SCVP) allows a client to delegate certification path construction and certification path validation to a server. The path construction or validation (e.g., making sure that none of the certificates in the path are revoked) is performed according to a validation policy, which contains one or more trust anchors. It allows simplification of client implementations and use of a set of predefined validation policies.

Table of Contents

1. Introduction

Certificate validation is complex. If certificate handling is to be widely deployed in a variety of applications and environments, the amount of processing an application needs to perform before it can accept a certificate needs to be reduced. There are a variety of applications that can make use of public key certificates, but these applications are burdened with the overhead of constructing and validating the certification paths. SCVP reduces this overhead for two classes of certificate-using applications.

The first class of applications wants just two things: confirmation that the public key belongs to the identity named in the certificate and confirmation that the public key can be used for the intended purpose. Such clients can completely delegate certification path construction and validation to the SCVP server. This is often referred to as delegated path validation (DPV).

The second class of applications can perform certification path validation, but they lack a reliable or efficient method of constructing a valid certification path. Such clients delegate certification path construction to the SCVP server, but not validation of the returned certification path. This is often referred to as delegated path discovery (DPD).

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [STDWORDS].

1.2. SCVP Overview

The primary goals of SCVP are to make it easier to deploy Public Key Infrastructure (PKI)-enabled applications by delegating path discovery and/or validation processing to a server, and to allow central administration of validation policies within an organization. SCVP can be used by clients that do much of the certificate processing themselves but simply want an untrusted server to collect information for them. However, when the client has complete trust in the SCVP server, SCVP can be used to delegate the work of certification path construction and validation, and SCVP can be used to ensure that policies are consistently enforced throughout an organization.

Untrusted SCVP servers can provide clients the certification paths. They can also provide clients the revocation information, such as Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP) responses, that the clients need to validate the certification paths constructed by the SCVP server. These services can be valuable to clients that do not implement the protocols needed to find and download intermediate certificates, CRLs, and OCSP responses.

Trusted SCVP servers can perform certification path construction and validation for the client. For a client that uses these services, the client inherently trusts the SCVP server as much as it would its own certification path validation software (if it contained such software). There are two main reasons that a client may want to trust such an SCVP server:

1. The client does not want to incur the overhead of including certification path validation software and running it for each certificate it receives.

2. The client is in an organization or community that wants to centralize management of validation policies. These policies might dictate that particular trust anchors are to be used and the types of policy checking that are to be performed during certification path validation.

1.3. SCVP Requirements

SCVP meets the mandatory requirements documented in [RQMTS] for DPV and DPD.

Note that RFC 3379 states the following requirement:

> The DPD response MUST indicate one of the following status alternatives:
>
> 1) one or more certification paths was found according to the path discovery policy, with all of the requested revocation information present.
>
> 2) one or more certification paths was found according to the path discovery policy, with a subset of the requested revocation information present.
>
> 3) one or more certification paths was found according to the path discovery policy, with none of the requested revocation information present.
>
> 4) no certification path was found according to the path discovery policy.
>
> 5) path construction could not be performed due to an error.

DPD responses constructed by SCVP servers do not differentiate between states 2) and 3). This property was discussed on the PKIX working group list and determined to be conformant with the intent of [RQMTS].

1.4. Validation Policies

A validation policy (as defined in RFC 3379 [RQMTS]) specifies the rules and parameters to be used by the SCVP server when validating a certificate. In SCVP, the validation policy to be used by the server either can be fully referenced in the request by the client (and thus no additional parameters are necessary) or can be referenced in the request by the client with additional parameters.

Policy definitions can be quite long and complex, and some policies may allow for the setting of a few parameters. The request can therefore be very simple if an object identifier (OID) is used to specify both the algorithm to be used and all the associated parameters of the validation policy. The request can be more complex if the validation policy fixes many of the parameters but allows the client to specify some of them. When the validation policy defines every parameter necessary, an SCVP request needs only to contain the certificate to be validated, the referenced validation policy, and any run-time parameters for the request.

A server publishes the references of the validation policies it supports. When these policies have parameters that may be overridden, the server communicates the default values for these parameters as well. The client can simplify the request by omitting a parameter from a request if the default value published by the server for a given validation policy reference is acceptable. However, if there is a desire to demonstrate to someone else that a specific validation policy with all its parameters has been used, the client will need to ask the server for the inclusion of the full validation policy with all the parameters in the response.

The inputs to the basic certification path processing algorithm used by SCVP are defined by [PKIX-1] in Section 6.1.1 and comprise:

Certificate to be validated (by value or by reference);

Validation time;

The initial policy set;

Initial inhibit policy mapping setting;

Initial inhibit anyPolicy setting; and

Initial require explicit policy setting.

The basic certification path processing algorithm also supports specification of one or more trust anchors (by value or reference) as an input. Where the client demands a certification path originating with a specific Certification Authority (CA), a single trust anchor is specified. Where the client is willing to accept paths beginning with any of several CAs, a set of trust anchors is specified.

The basic certification path processing algorithm also supports the following parameters, which are defined in [PKIX-1], Section 4:

The usage of the key contained in the certificate (e.g., key encipherment, key agreement, signature); and

Other application-specific purposes for which the certified public key may be used.

1.5. Validation Algorithm

The validation algorithm is determined by agreement between the client and the server and is represented as an OID. The algorithm defines the checking that will be performed by the server to determine whether the certificate is valid. A validation algorithm

is one of the parameters to a validation policy. SCVP defines a basic validation algorithm that implements the basic path validation algorithm as defined in [PKIX-1], and it permits the client to request additional information about the certificate to be validated. New validation algorithms can be specified that define additional checks if needed. These new validation algorithms may specify additional parameters. The values for these parameters may be defined by any validation policy that uses the algorithm or may be included by the client in the request.

Application-specific validation algorithms, in addition to those defined in this document, can be defined to meet specific requirements not covered by the basic validation algorithm. The validation algorithms documented here should serve as a guide for the development of further application-specific validation algorithms. For example, a new application-specific validation algorithm might require the presence of a particular name form in the subject alternative name extension of the certificate.

1.6. Validation Requirements

For a certification path to be considered valid under a particular validation policy, it MUST be a valid certification path as defined in [PKIX-1], and all validation policy constraints that apply to the certification path MUST be verified.

Revocation checking is one aspect of certification path validation defined in [PKIX-1]. However, revocation checking is an optional feature in [PKIX-1], and revocation information is distributed in multiple formats. Clients specify in requests whether revocation checking should be performed and whether revocation information should be returned in the response.

Servers MUST be capable of indicating the sources of revocation information that they are capable of processing:

1. full CRLs (or full Authority Revocation Lists);

2. OCSP responses, using [OCSP];

3. delta CRLs; and

4. indirect CRLs.

2. Protocol Overview

SCVP uses a simple request-response model. That is, the SCVP client creates a request and sends it to the SCVP server, and then the SCVP server creates a single response and sends it to the client. The typical use of SCVP is expected to be over HTTP [HTTP], but it can also be used with email or any other protocol that can transport digitally signed objects. Appendices A and B provide the details necessary to use SCVP with HTTP.

SCVP includes two request-response pairs. The primary request-response pair handles certificate validation. The secondary request-response pair is used to determine the list of validation policies and default parameters supported by a specific SCVP server.

Section 3 defines the certificate validation request.

Section 4 defines the corresponding certificate validation response.

Section 5 defines the validation policies request.

Section 6 defines the corresponding validation policies response.

Appendix A registers MIME types for SCVP requests and responses, and Appendix B describes the use of these MIME types with HTTP.

3. Validation Request

An SCVP client request to the server MUST be a single CVRequest item. When a CVRequest is encapsulated in a MIME body part, application/scvp-cv-request MUST be used. There are two forms of SCVP request: unprotected and protected. A protected request is used to authenticate the client to the server or to provide anonymous client integrity over the request-response pair. The protection is provided by a digital signature or message authentication code (MAC). In the later case, the MAC key is derived using a key agreement algorithm, such as Diffie-Hellman. If the client’s public key is contained in a certificate, then it may be used to authenticate the client. More commonly, the client’s key agreement public key will be ephemeral, supporting anonymous client integrity.

A server MAY require all requests to be protected, and a server MAY discard all unprotected requests. Alternatively, a server MAY choose to process unprotected requests.

The unprotected request consists of a CVRequest encapsulated in a Cryptographic Message Syntax (CMS) ContentInfo [CMS]. An overview of this structure is provided below and is only intended as

illustrative. The definitive ASN.1 is found in [CMS]. Many details are not shown, but the way that SCVP makes use of CMS is clearly illustrated.

```
ContentInfo {
  contentType        id-ct-scvp-certValRequest,
                              -- (1.2.840.113549.1.9.16.1.10)
  content            CVRequest }
```

The protected request consists of a CVRequest encapsulated in either a SignedData or AuthenticatedData, which is in turn encapsulated in a ContentInfo. That is, the EncapsulatedContentInfo field of either SignedData or AuthenticatedData consists of an eContentType field with a value of id-ct-scvp-certValRequest and an eContent field that contains a Distinguished Encoding Rules (DER)-encoded CVRequest. SignedData is used when the request is digitally signed. AuthenticatedData is used with a message authentication code (MAC).

All SCVP clients and servers MUST support SignedData for signed requests and responses. SCVP clients and servers SHOULD support AuthenticatedData for MAC-protected requests and responses.

If the client uses SignedData, it MUST have a public key that has been bound to a subject identity by a certificate that conforms to the PKIX profile [PKIX-1], and that certificate MUST be suitable for signing the SCVP request. That is:

1. If the key usage extension is present, either the digital signature or the non-repudiation bit MUST be asserted.

2. If the extended key usage extension is present, it MUST contain either the SCVP client OID (see Section 3.11), the anyExtendedKeyUsage OID, or another OID acceptable to the SCVP server.

The client MUST put an unambiguous reference to its certificate in the SignedData that encapsulates the request. The client SHOULD include its certificate in the request, but MAY omit the certificate to reduce the size of the request. The client MAY include other certificates in the request to aid the validation of its certificates by the SCVP server. The signerInfos field of SignedData MUST include exactly one SignerInfo. The SignedData MUST NOT include the unsignedAttrs field.

The client MUST put its key agreement public key, or an unambiguous reference to a certificate that contains its key agreement public key, in the AuthenticatedData that encapsulates the request. If an ephemeral key agreement key pair is used, then the ephemeral key agreement public key is carried in the originatorKey field of KeyAgreeRecipientInfo, which requires the client to obtain the server's key agreement public key before computing the message authentication code (MAC). An SCVP server's key agreement key is included in its validation policy response message (see Section 6). The recipientInfos field of AuthenticatedData MUST include exactly one RecipientInfo, which contains information for the SCVP server. The AuthenticatedData MUST NOT include the unauthAttrs field.

The syntax and semantics for SignedData, AuthenticatedData, and ContentInfo are defined in [CMS]. The syntax and semantics for CVRequest are defined below. The CVRequest item contains the client request. The CVRequest contains the cvRequestVersion and query items; the CVRequest MAY also contain the requestorRef, requestNonce, requestorName, responderName, requestExtensions, signatureAlg, and hashAlg items.

The CVRequest MUST have the following syntax:

```
CVRequest ::= SEQUENCE {
  cvRequestVersion        INTEGER DEFAULT 1,
  query                   Query,
  requestorRef        [0] GeneralNames OPTIONAL,
  requestNonce        [1] OCTET STRING OPTIONAL,
  requestorName       [2] GeneralName OPTIONAL,
  responderName       [3] GeneralName OPTIONAL,
  requestExtensions   [4] Extensions OPTIONAL,
  signatureAlg        [5] AlgorithmIdentifier OPTIONAL,
  hashAlg             [6] OBJECT IDENTIFIER OPTIONAL,
  requestorText       [7] UTF8String (SIZE (1..256)) OPTIONAL }
```

Conforming clients MUST be able to construct requests with cvRequestVersion and query. Conforming clients MUST DER encode the CVRequest in both protected and unprotected messages to facilitate unambiguous hash-based referencing in the corresponding response message. SCVP clients that insist on creation of a fresh response (e.g., to protect against a replay attack or ensure information is up to date) MUST support requestNonce. Support for the remaining items is optional in client implementations.

Conforming servers MUST be able to parse CVRequests that contain any or all of the optional items.

Each of the items within the CVRequest is described in the following sections.

3.1. cvRequestVersion

The cvRequestVersion item defines the version of the SCVP CVRequest used in a request. The subsequent response MUST use the same version number. The value of the cvRequestVersion item MUST be one (1) for a client implementing this specification. Future updates to this specification must specify other values if there are any changes to syntax or semantics. However, new extensions may be defined without changing the version number.

SCVP clients MUST support asserting this value and SCVP servers MUST be capable of processing this value.

3.2. query

The query item specifies one or more certificates that are the subject of the request; the certificates can be either public key certificates [PKIX-1] or attribute certificates [PKIX-AC]. A query MUST contain a queriedCerts item as well as one checks item, and one validationPolicy item; a query MAY also contain wantBack, responseFlags, serverContextInfo, validationTime, intermediateCerts, revInfos, producedAt, and queryExtensions items.

A Query MUST have the following syntax:

```
Query ::= SEQUENCE {
  queriedCerts            CertReferences,
  checks                  CertChecks,
   -- Note: tag [0] not used --
  wantBack            [1] WantBack OPTIONAL,
  validationPolicy        ValidationPolicy,
  responseFlags           ResponseFlags OPTIONAL,
  serverContextInfo   [2] OCTET STRING OPTIONAL,
  validationTime      [3] GeneralizedTime OPTIONAL,
  intermediateCerts   [4] CertBundle OPTIONAL,
  revInfos            [5] RevocationInfos OPTIONAL,
  producedAt          [6] GeneralizedTime OPTIONAL,
  queryExtensions     [7] Extensions OPTIONAL }
```

The list of certificate references in the queriedCerts item tells the server the certificate(s) for which the client wants information. The checks item specifies the checking that the client wants performed. The wantBack item specifies the objects that the client wants the server to return in the response. The validationPolicy item specifies the validation policy that the client wants the server

to employ. The responseFlags item allows the client to request optional features for the response. The serverContextInfo item tells the server that additional information from a previous request-response is desired. The validationTime item tells the date and time relative to which the client wants the server to perform the checks. The intermediateCerts and revInfos items provide context for the client request. The queryExtensions item provides for future expansion of the query syntax. The syntax and semantics of each of these items are discussed in the following sections.

Conforming clients MUST be able to construct a Query with a queriedCerts item that specifies at least one certificate, checks, and validationPolicy. Conforming SCVP clients MAY support specification of multiple certificates and MAY support the optional items in the Query structure.

SCVP clients that support delegated path discovery (DPD) as defined in [RQMTS] MUST support wantBack and responseFlags. SCVP clients that insist on creation of a fresh response (e.g., to protect against a replay attack or ensure information is up to date) MUST support responseFlags.

Conforming servers MUST be able to process a Query that contains any of the optional items, and MUST be able to process a Query that specifies multiple certificates.

3.2.1. queriedCerts

The queriedCerts item is a SEQUENCE of one or more certificates, each of which is a subject of the request. The specified certificates are either public key certificates or attribute certificates; if more than one certificate is specified, all must be of the same type. Each certificate is either directly included, or it is referenced. When referenced, a hash value of the referenced item is included to ensure that the SCVP client and the SCVP server both obtain the same certificate when the referenced certificate is fetched. Certificate references use the SCVPCertID type, which is described below. A single request MAY contain both directly included and referenced certificates.

CertReferences has the following syntax:

```
CertReferences ::= CHOICE {
  pkcRefs     [0] SEQUENCE SIZE (1..MAX) OF PKCReference,
  acRefs      [1] SEQUENCE SIZE (1..MAX) OF ACReference }
```

```
PKCReference ::= CHOICE {
  cert        [0] Certificate,
  pkcRef      [1] SCVPCertID }

ACReference ::= CHOICE {
  attrCert    [2] AttributeCertificate,
  acRef       [3] SCVPCertID }

SCVPCertID ::= SEQUENCE {
  certHash        OCTET STRING,
  issuerSerial    SCVPIssuerSerial,
  hashAlgorithm   AlgorithmIdentifier DEFAULT { algorithm sha-1 } }
```

The ASN.1 definition of Certificate is imported from [PKIX-1] and the definition of AttributeCertificate is imported from [PKIX-AC].

When creating a SCVPCertID, the certHash is computed over the entire DER-encoded certificate including the signature. The hash algorithm used to compute certHash is specified in hashAlgorithm. The hash algorithm used to compute certHash SHOULD be one of the hash algorithms specified in the hashAlgorithms item of the server's validation policy response message.

When encoding SCVPIssuerSerial, serialNumber is the serial number that uniquely identifies the certificate. For public key certificates, the issuer MUST contain only the issuer name from the certificate encoded in the directoryName choice of GeneralNames. For attribute certificates, the issuer MUST contain the issuer name field from the attribute certificate.

Conforming clients MUST be able to reference a certificate by direct inclusion. Clients SHOULD be able to specify a certificate using the SCVPCertID. Conforming clients MAY be able to reference multiple certificates and MAY be able to reference both public key and attribute certificates.

Conforming SCVP Server implementations MUST be able to process CertReferences with multiple certificates. Conforming SCVP server implementations MUST be able to parse CertReferences that contain either public key or attribute certificates. Conforming SCVP server implementations MUST be able to parse both the cert and pkcRef choices in PKCReference. Conforming SCVP server implementations that process attribute certificates MUST be able to parse both the attrCert and acRef choices in ACReference.

3.2.2. checks

The checks item describes the checking that the SCVP client wants the SCVP server to perform on the certificate(s) in the queriedCerts item. The checks item contains a sequence of object identifiers (OIDs). Each OID tells the SCVP server what checking the client expects the server to perform. For each check specified in the request, the SCVP server MUST perform the requested check, or return an error. A server may choose to perform additional checks (e.g., a server that is only asked to build a validated certification path may choose to also perform revocation status checks), although the server cannot indicate in the response that the additional checks have been performed, except in the case of an error response.

The checks item uses the CertChecks type, which has the following syntax:

```
CertChecks ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER
```

For public key certificates, the following checks are defined in this document:

- id-stc-build-pkc-path: Build a prospective certification path to a trust anchor (as defined in Section 6.1 of [PKIX-1]);

- id-stc-build-valid-pkc-path: Build a validated certification path to a trust anchor (revocation checking not required);

- id-stc-build-status-checked-pkc-path: Build a validated certification path to a trust anchor and perform revocation status checks on the certification path.

Conforming SCVP server implementations that support delegated path discovery (DPD) as defined in [RQMTS] MUST support the id-stc-build-pkc-path check. Conforming SCVP server implementations that support delegated path validation (DPV) as defined in [RQMTS] MUST support the id-stc-build-valid-pkc-path and id-stc-build-status-checked-pkc-path checks.

For attribute certificates, the following checks are defined in this document:

- id-stc-build-aa-path: Build a prospective certification path to a trust anchor for the Attribute Certificate (AC) issuer;

- id-stc-build-valid-aa-path: Build a validated certification path to a trust anchor for the AC issuer;

- id-stc-build-status-checked-aa-path: Build a validated certification path to a trust anchor for the AC issuer and perform revocation status checks on the certification path for the AC issuer;

- id-stc-status-check-ac-and-build-status-checked-aa-path: Build a validated certification path to a trust anchor for the AC issuer and perform revocation status checks on the AC as well as the certification path for the AC issuer.

Conforming SCVP server implementations MAY support the attribute certificates checks.

For these purposes, the following OIDs are defined:

```
id-stc OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
        dod(6) internet(1) security(5) mechanisms(5) pkix(7) 17 }

id-stc-build-pkc-path         OBJECT IDENTIFIER ::= { id-stc 1 }
id-stc-build-valid-pkc-path   OBJECT IDENTIFIER ::= { id-stc 2 }
id-stc-build-status-checked-pkc-path
                              OBJECT IDENTIFIER ::= { id-stc 3 }
id-stc-build-aa-path          OBJECT IDENTIFIER ::= { id-stc 4 }
id-stc-build-valid-aa-path    OBJECT IDENTIFIER ::= { id-stc 5 }
id-stc-build-status-checked-aa-path
                              OBJECT IDENTIFIER ::= { id-stc 6 }
id-stc-status-check-ac-and-build-status-checked-aa-path
                              OBJECT IDENTIFIER ::= { id-stc 7 }
```

Other specifications may define additional checks.

Conforming client implementations MUST support assertion of at least one of the standard checks. Conforming clients MAY support assertion of multiple checks. Conforming clients need not support all of the checks defined in this section.

3.2.3. wantBack

The optional wantBack item describes any information the SCVP client wants from the SCVP server for the certificate(s) in the queriedCerts item in addition to the results of the checks specified in the checks item. If present, the wantBack item MUST contain a sequence of object identifiers (OIDs). Each OID tells the SCVP server what the client wants to know about the queriedCerts item. For each type of information specified in the request, the server MUST return information regarding its finding (in a successful response).

For example, a request might include a checks item that only specifies certification path building and include a wantBack item that requests the return of the certification path built by the server. In this case, the response would not include a status for the validation of the certification path, but it would include a prospective certification path. A client that wants to perform its own certification path validation might use a request of this form.

Alternatively, a request might include a checks item that requests the server to build a certification path and validate it, including revocation checking, and not include a wantBack item. In this case, the response would include only a status for the validation of the certification path. A client that completely delegates certification path validation might use a request of this form.

The wantBack item uses the WantBack type, which has the following syntax:

```
WantBack ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER
```

For public key certificates, the following wantBacks are defined in this document:

- id-swb-pkc-cert: The certificate that was the subject of the request;

- id-swb-pkc-best-cert-path: The certification path built for the certificate including the certificate that was validated;

- id-swb-pkc-revocation-info: Proof of revocation status for each certificate in the certification path;

- id-swb-pkc-public-key-info: The public key from the certificate that was the subject of the request;

- id-swb-pkc-all-cert-paths: A set of certification paths for the certificate that was the subject of the request;

- id-swb-pkc-ee-revocation-info: Proof of revocation status for the end entity certificate in the certification path; and

- id-swb-pkc-CAs-revocation-info: Proof of revocation status for each CA certificate in the certification path.

All conforming SCVP server implementations MUST support the id-swb-pkc-cert and id-swb-pkc-public-key-info wantBacks. Conforming SCVP server implementations that support delegated path discovery (DPD) as defined in [RQMTS] MUST support the id-swb-pkc-best-cert-path and id-swb-pkc-revocation-info wantBacks.

SCVP provides two methods for a client to obtain multiple certification paths for a certificate. The client could use serverContextInfo to request one path at a time (see Section 3.2.6). After obtaining each path, the client could submit the serverContextInfo from the previous request to obtain another path until either the client found a suitable path or the server indicated (by not returning a serverContextInfo) that no more paths were available. Alternatively, the client could send a single request with an id-swb-pkc-all-cert-paths wantBack, in which case the server would return all of the available paths in a single response.

The server may, at its discretion, limit the number of paths that it returns in response to the id-swb-pkc-all-cert-paths. When the request includes an id-swb-pkc-all-cert-paths wantBack, the response SHOULD NOT include a serverContextInfo.

For attribute certificates, the following wantBacks are defined in this document:

- id-swb-ac-cert: The attribute certificate that was the subject of the request;

- id-swb-aa-cert-path: The certification path built for the AC issuer certificate;

- id-swb-ac-revocation-info: Proof of revocation status for each certificate in the AC issuer certification path; and

- id-swb-aa-revocation-info: Proof of revocation status for the attribute certificate.

Conforming SCVP server implementations MAY support the attribute certificate wantBacks.

The following wantBack can be used for either public key or attribute certificates:

- id-swb-relayed-responses: Any SCVP responses received by the server that were used to generate the response to this query.

Conforming SCVP servers MAY support the id-swb-relayed-responses wantBack.

For these purposes, the following OIDs are defined:

```
id-swb OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
        dod(6) internet(1) security(5) mechanisms(5) pkix(7) 18 }

id-swb-pkc-best-cert-path      OBJECT IDENTIFIER ::= { id-swb 1 }
id-swb-pkc-revocation-info     OBJECT IDENTIFIER ::= { id-swb 2 }
id-swb-pkc-public-key-info     OBJECT IDENTIFIER ::= { id-swb 4 }
id-swb-aa-cert-path            OBJECT IDENTIFIER ::= { id-swb 5 }
id-swb-aa-revocation-info      OBJECT IDENTIFIER ::= { id-swb 6 }
id-swb-ac-revocation-info      OBJECT IDENTIFIER ::= { id-swb 7 }
id-swb-relayed-responses       OBJECT IDENTIFIER ::= { id-swb 9 }
id-swb-pkc-cert                OBJECT IDENTIFIER ::= { id-swb 10}
id-swb-ac-cert                 OBJECT IDENTIFIER ::= { id-swb 11}
id-swb-pkc-all-cert-paths      OBJECT IDENTIFIER ::= { id-swb 12}
id-swb-pkc-ee-revocation-info  OBJECT IDENTIFIER ::= { id-swb 13}
id-swb-pkc-CAs-revocation-info OBJECT IDENTIFIER ::= { id-swb 14}
```

Other specifications may define additional wantBacks.

Conforming client implementations that support delegated path validation (DPV) as defined in [RQMTS] SHOULD support assertion of at least one wantBack. Conforming client implementations that support delegated path discovery (DPD) as defined in [RQMTS] MUST support assertion of at least one wantBack. Conforming clients MAY support assertion of multiple wantBacks. Conforming clients need not support all of the wantBacks defined in this section.

3.2.4. validationPolicy

The validationPolicy item defines the validation policy that the client wants the SCVP server to use during certificate validation. If this policy cannot be used for any reason, then the server MUST return an error response.

A validation policy MUST define default values for all parameters necessary for processing an SCVP request. For each parameter, a validation policy may either allow the client to specify a non-default value or forbid the use of a non-default value. If the client wishes to use the default values for all of the parameters, then the client need only supply a reference to the policy in this item. If the client wishes to use non-default values for one or more parameters, then the client supplies a reference to the policy plus whatever parameters are necessary to complete the request in this item. If there are any conflicts between the policy referenced in the request and any supplied parameter values in the request, then the server MUST return an error response.

The syntax of the validationPolicy item is:

```
ValidationPolicy ::= SEQUENCE {
  validationPolRef          ValidationPolRef,
  validationAlg         [0] ValidationAlg OPTIONAL,
  userPolicySet         [1] SEQUENCE SIZE (1..MAX) OF OBJECT
                              IDENTIFIER OPTIONAL,
  inhibitPolicyMapping  [2] BOOLEAN OPTIONAL,
  requireExplicitPolicy [3] BOOLEAN OPTIONAL,
  inhibitAnyPolicy      [4] BOOLEAN OPTIONAL,
  trustAnchors          [5] TrustAnchors OPTIONAL,
  keyUsages             [6] SEQUENCE OF KeyUsage OPTIONAL,
  extendedKeyUsages     [7] SEQUENCE OF KeyPurposeId OPTIONAL,
  specifiedKeyUsages    [8] SEQUENCE OF KeyPurposeId OPTIONAL }
```

The validationPolRef item is required, but the remaining items are optional. The optional items are used to provide validation policy parameters. When the client uses the validation policy's default values for all parameters, all of the optional items are absent.

At a minimum, conforming SCVP client implementations MUST support the validationPolRef item. Conforming client implementations MAY support any or all of the optional items in ValidationPolicy.

Conforming SCVP servers MUST support processing of a ValidationPolicy that contains any or all of the optional items.

The validationAlg item specifies the validation algorithm. The userPolicySet item provides an acceptable set of certificate policies. The inhibitPolicyMapping item inhibits certificate policy mapping during certification path validation. The requireExplicitPolicy item requires at least one valid certificate policy in the certificate policies extension. The inhibitAnyPolicy item indicates whether the anyPolicy certificate policy OID is processed or ignored when evaluating certificate policy. The trustAnchors item indicates the trust anchors that are acceptable to the client. The keyUsages item indicates the technical usage of the public key that is to be confirmed by the server as acceptable. The extendedKeyUsages item indicates the application-specific usage of the public key that is to be confirmed by the server as acceptable. The syntax and semantics of each of these items are discussed in the following sections.

3.2.4.1. validationPolRef

The reference to the validation policy is an OID that the client and server have agreed represents a particular validation policy.

The syntax of the validationPolRef item is:

```
ValidationPolRef::= SEQUENCE {
  valPolId              OBJECT IDENTIFIER,
  valPolParams          ANY DEFINED BY valPolId OPTIONAL }
```

Where a validation policy supports additional policy-specific parameter settings, these values are specified using the valPolParams item. The syntax and semantics of the parameters structure are defined by the object identifier encoded as the valPolId. Where a validation policy has no parameters, such as the default validation policy (see Section 3.2.4.1.1), this item MUST be omitted.

Parameters specified in this item are independent of the validation algorithm and the validation algorithm's parameters (see Section 3.2.4.2). For example, a server may support a validation policy where it validates a certificate using the name validation algorithm and also makes a determination regarding the creditworthiness of the subject. In this case, the validation policy parameters could be used to specify the value of the transaction. The validation algorithm parameters are used to specify the application identifier and name for the name validation algorithm.

Conforming SCVP client implementations MUST support specification of a validation policy. Conforming SCVP client implementations MAY be able to specify parameters for a validation policy. Conforming SCVP server implementations MUST be able to process valPolId and MAY be able to process valPolParams.

3.2.4.1.1. Default Validation Policy

The client can request the SCVP server's default validation policy or another validation policy. The default validation policy corresponds to standard certification path processing as defined in [PKIX-1] with server-chosen default values (e.g., with a server-determined policy set and trust anchors). The default values can be distributed out of band or using the policy request mechanism (see Section 5). This mechanism permits the deployment of an SCVP server without obtaining a new object identifier.

The object identifier that identifies the default validation policy is:

```
id-svp OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
       dod(6) internet(1) security(5) mechanisms(5) pkix(7) 19 }

id-svp-defaultValPolicy OBJECT IDENTIFIER ::= { id-svp 1 }
```

The default validation policy MUST use the basic validation algorithm as its default validation algorithm (see Section 3.2.4.2.1), and has no validation policy parameters (see Section 3.2.4.1).

When using the default validation policy, the client can override any of the default parameter values by supplying a specific value in the request. The SCVP server MUST make use of the provided parameter values or return an error response.

Conforming implementations of SCVP servers MUST support the default policy. However, an SCVP server may be configured to send an error response to all requests using the default policy to meet local security requirements.

3.2.4.2. validationAlg

The optional validationAlg item defines the validation algorithm to be used by the SCVP server during certificate validation. The value of this item can be determined by agreement between the client and the server. The validation algorithm is represented by an object identifier.

The syntax of the validationAlg item is:

```
ValidationAlg ::= SEQUENCE {
  valAlgId              OBJECT IDENTIFIER,
  parameters            ANY DEFINED BY valAlgId OPTIONAL }
```

The following section specifies the basic validation algorithm and the name validation algorithm.

SCVP servers MUST recognize and support both validation algorithms defined in this section. SCVP clients that support explicit assertion of the validation algorithm MUST support the basic validation algorithm and SHOULD support the name validation algorithm. Other validation algorithms can be specified in other documents for use with specific applications. SCVP clients and servers MAY support any such validation algorithms.

3.2.4.2.1. Basic Validation Algorithm

The client can request use of the SCVP basic validation algorithm or another algorithm. For identity certificates, the basic validation algorithm MUST implement the certification path validation algorithm as defined in Section 6 of [PKIX-1]. For attribute certificates, the basic validation algorithm MUST implement certification path validation as defined in Section 5 of [PKIX-AC]. Other validation algorithms MAY implement functions over and above those in the basic

algorithm, but validation algorithms MUST generate results compliant with the basic validation algorithm. That is, none of the validation requirements in the basic algorithm may be omitted from any newly defined validation algorithms. However, other validation algorithms MAY reject paths that are valid using the basic validation algorithm. The object identifier to identify the basic validation algorithm is:

```
id-svp-basicValAlg OBJECT IDENTIFIER ::= { id-svp 3 }
```

When id-svp-basicValAlg appears in valAlgId, the parameters item MUST be absent.

3.2.4.2.2. Basic Validation Algorithm Errors

The following errors are defined for the basic validation algorithm for inclusion in the validationErrors item in the response (see Section 4.9.6). These errors can be used by any other validation algorithm since all validation algorithms MUST implement the functionality of the basic validation algorithm.

```
id-bvae OBJECT IDENTIFIER ::= id-svp-basicValAlg

id-bvae-expired              OBJECT IDENTIFIER ::= { id-bvae 1 }
id-bvae-not-yet-valid        OBJECT IDENTIFIER ::= { id-bvae 2 }
id-bvae-wrongTrustAnchor     OBJECT IDENTIFIER ::= { id-bvae 3 }
id-bvae-noValidCertPath      OBJECT IDENTIFIER ::= { id-bvae 4 }
id-bvae-revoked              OBJECT IDENTIFIER ::= { id-bvae 5 }
id-bvae-invalidKeyPurpose    OBJECT IDENTIFIER ::= { id-bvae 9 }
id-bvae-invalidKeyUsage      OBJECT IDENTIFIER ::= { id-bvae 10 }
id-bvae-invalidCertPolicy    OBJECT IDENTIFIER ::= { id-bvae 11 }
```

The id-bvae-expired value means that the validation time used for the request was later than the notAfter time in the end certificate (the certificate specified in the queriedCerts item).

The id-bvae-not-yet-valid value means that the validation time used for the request was before the notBefore time in the end certificate.

The id-bvae-wrongTrustAnchor value means that a certification path could not be constructed for the client-specified trust anchor(s), but a path exists for one of the trust anchors specified in the server's default validation policy.

The id-bvae-noValidCertPath value means that the server could not construct a sequence of intermediate certificates between the trust anchor and the target certificate that satisfied the request.

The id-bvae-revoked value means that the end certificate has been revoked.

The id-bvae-invalidKeyPurpose value means that the extended key usage extension ([PKIX-1], Section 4.2.1.13) in the end certificate does not satisfy the validation policy.

The id-bvae-invalidKeyUsage value means that the keyUsage extension ([PKIX-1], Section 4.2.1.3) in the end certificate does not satisfy the validation policy. For example, the keyUsage extension in the certificate may assert only the keyEncipherment bit, but the validation policy specifies in the keyUsages item that digitalSignature is required.

The id-bvae-invalidCertPolicy value means that the path is not valid under any of the policies specified in the user policy set and explicit policies are required. That is, the valid_policy_tree is NULL and the explicit_policy variable is zero ([PKIX-1], Section 6.1.5).

3.2.4.2.3. Name Validation Algorithm

The name validation algorithm allows the client to specify one or more subject names that MUST appear in the end certificate in addition to the requirements specified for the basic validation algorithm. The name validation algorithm allows the client to supply an application identifier and a name to the server. The application identifier defines the name matching rules to use in comparing the name supplied in the request with the names in the certificate.

```
id-svp-nameValAlg OBJECT IDENTIFIER ::= { id-svp 2 }
```

When the id-svp-nameValAlg appears as a valAlgId, the parameters MUST use the NameValidationAlgParms syntax:

```
NameValidationAlgParms ::= SEQUENCE {
  nameCompAlgId     OBJECT IDENTIFIER,
  validationNames   GeneralNames }
```

GeneralNames is defined in [PKIX-1].

If more than one name is supplied in the validationNames value, all names MUST be of the same type. The certificate must contain a matching name for each of the names supplied in validationNames according to the name matching rules associated with the nameCompAlgId. This specification defines three sets of name matching rules.

If the nameCompAlgId supplied in the request is id-nva-dnCompAlg, then GeneralNames supplied in the request MUST be a directoryName, and the matching rules to be used are defined in [PKIX-1]. The certificate must contain a matching name in either the subject field or a directoryName in the subjectAltName extension. This specification defines the OID for id-nva-dnCompAlg as follows:

```
id-nva-dnCompAlg   OBJECT IDENTIFIER ::= { id-svp 4 }
```

If the nameCompAlgId supplied in the request is id-kp-serverAuth [PKIX-1], then GeneralNames supplied in the request MUST be a dNSName, and the matching rules to be used are defined in [PKIX-1].

If a subjectAltName extension is present and includes one or more names of type dNSName, a match in any one of the set is considered acceptable. If the subjectAltName extension is omitted, or does not include any names of type dNSName, the (most specific) Common Name field in the subject field of the certificate MUST be used.

Names may contain the wildcard character *, which is considered to match any single domain name component. That is, *.a.com matches foo.a.com but not bar.foo.a.com.

If the nameCompAlgId supplied in the request is id-kp-mailProtection [PKIX-1], then GeneralNames supplied in the request MUST be an rfc822Name, and the matching rules are defined in [SMIME-CERT].

Conforming SCVP servers MUST support the name validation algorithm and the matching rules associated with id-nva-dnCompAlg, id-kp-serverAuth, and id-kp-mailProtection. SCVP servers MAY support other name matching rules.

3.2.4.2.4. Name Validation Algorithm Errors

The following errors are defined for the name validation algorithm:

```
id-nvae OBJECT IDENTIFIER ::= id-svp-nameValAlg

id-nvae-name-mismatch    OBJECT IDENTIFIER ::= { id-nvae 1 }
id-nvae-no-name          OBJECT IDENTIFIER ::= { id-nvae 2 }
id-nvae-unknown-alg      OBJECT IDENTIFIER ::= { id-nvae 3 }
id-nvae-bad-name         OBJECT IDENTIFIER ::= { id-nvae 4 }
id-nvae-bad-name-type    OBJECT IDENTIFIER ::= { id-nvae 5 }
id-nvae-mixed-names      OBJECT IDENTIFIER ::= { id-nvae 6 }
```

The id-nvae-name-mismatch value means the client supplied a name with the request, which the server recognized and the server found a corresponding name type in the certificate, but was unable to find a

match to the name supplied. For example, the client supplied a DNS name of example1.com, and the certificate contained a DNS name of example.com.

The id-nvae-no-name value means the client supplied a name with the request, which the server recognized, but the server could not find the corresponding name type in the certificate. For example, the client supplied a DNS name of example1.com, and the certificate only contained a rfc822Name of user@example.com.

The id-nvae-unknown-alg value means the client supplied a nameCompAlgId that the server does not recognize.

The id-nvae-bad-name value means the client supplied either an empty or malformed name in the request.

The id-nvae-bad-name-type value means the client supplied an inappropriate name type for the application identifier. For example, the client specified a nameCompAlgId of id-kp-serverAuth, and an rfc822Name of user@example.com.

The id-nvae-mixed-names value means the client supplied multiple names in the request of different types.

3.2.4.3. userPolicySet

The userPolicySet item specifies a list of certificate policy identifiers that the SCVP server MUST use when constructing and validating a certification path. The userPolicySet item specifies the user-initial-policy-set as defined in Section 6 of [PKIX-1]. A userPolicySet containing the anyPolicy OID indicates a user-initial-policy-set of any-policy.

SCVP clients SHOULD support the userPolicySet item in requests, and SCVP servers MUST support the userPolicySet item in requests.

3.2.4.4. inhibitPolicyMapping

The inhibitPolicyMapping item specifies an input to the certification path validation algorithm, and it controls whether policy mapping is allowed during certification path validation (see [PKIX-1], Section 6.1.1). If the client wants the server to inhibit policy mapping, inhibitPolicyMapping is set to TRUE in the request. SCVP clients MAY support inhibiting policy mapping. SCVP servers SHOULD support inhibiting policy mapping.

3.2.4.5. requireExplicitPolicy

The requireExplicitPolicy item specifies an input to the certification path validation algorithm, and it controls whether there must be at least one valid policy in the certificate policies extension (see [PKIX-1], Section 6.1.1). If the client wants the server to require at least one policy, requireExplicitPolicy is set to TRUE in the request.

SCVP clients MAY support requiring explicit policies. SCVP servers SHOULD support requiring explicit policies.

3.2.4.6. inhibitAnyPolicy

The inhibitAnyPolicy item specifies an input to the certification path validation algorithm (see [PKIX-1], Section 6.1.1), and it controls whether the anyPolicy OID is processed or ignored when evaluating certificate policy. If the client wants the server to ignore the anyPolicy OID, inhibitAnyPolicy MUST be set to TRUE in the request.

SCVP clients MAY support ignoring the anyPolicy OID. SCVP servers SHOULD support ignoring the anyPolicy OID.

3.2.4.7. trustAnchors

The trustAnchors item specifies the trust anchors at which the certification path must terminate if the path is to be considered valid by the SCVP server for the request. If a trustAnchors item is present, the server MUST NOT consider any certification paths ending in other trust anchors as valid.

The TrustAnchors type contains one or more trust anchor specifications. A certificate reference can be used to identify the trust anchor by certificate hash and distinguished name with serial number. Alternatively, trust anchors can be provided directly. The order of trust anchor specifications within the sequence is not important. Any CA certificate that meets the requirements of [PKIX-1] for signing certificates can be provided as a trust anchor. If a trust anchor is supplied that does not meet these requirements, the server MUST return an error response.

The trust anchor itself, regardless of its form, MUST NOT be included in any certification path returned by the SCVP server.

TrustAnchors has the following syntax:

```
TrustAnchors ::= SEQUENCE SIZE (1..MAX) OF PKCReference
```

SCVP servers MUST support trustAnchors. SCVP clients SHOULD support trustAnchors.

3.2.4.8. keyUsages

The key usage extension ([PKIX-1], Section 4.2.1.3) in the certificate defines the technical purpose (such as encipherment, signature, and CRL signing) of the key contained in the certificate. If the client wishes to confirm the technical usage, then it can communicate the usage it wants to validate by the same structure using the same semantics as defined in [PKIX-1]. For example, if the client obtained the certificate in the context of a digital signature, it can confirm this use by including a keyUsage structure with the digital signature bit set.

If the keyUsages item is present and contains an empty sequence, it indicates that the client does not require any particular key usage.

If the keyUsages item contains one or more keyUsage definitions, then the certificate MUST satisfy at least one of the specified keyUsage definitions. If the client is willing to accept multiple possibilities, then the client passes in a sequence of possible patterns. Each keyUsage can contain a set of one or more bits set in the request, all bits MUST be set in the certificate to match against an instance of the keyUsage in the SCVP request. The certificate key usage extension may contain more usages than requested. For example, if a client wishes to check for either digital signature or non-repudiation, then the client provides two keyUsage values, one with digital signature set and the other with non-repudiation set. If the key usage extension is absent from the certificate, the certificate MUST be considered good for all usages and therefore any pattern in the SCVP request will match.

SCVP clients SHOULD support keyUsages, and SCVP servers MUST support keyUsages.

3.2.4.9. extendedKeyUsages

The extended key usage extension ([PKIX-1], Section 4.2.1.13) defines more specific technical purposes, in addition to, or in place of, the purposes indicated in the key usage extension, for which the certified public key may be used. If the client will accept certificates that are consistent with a particular value (or values) in the extended key usage extension, then it can communicate the appropriate usages using the same semantics as defined in [PKIX-1].

For example, if the client obtained the certificate in the context of a Transport Layer Security (TLS) server, it can confirm the certificate is consistent with this usage by including the extended key usage structure with the id-kp-serverAuth object identifier.

If the extension is absent, or is present and asserts the anyExtendedKeyUsage OID, then all usages specified in the request are a match. If the extension is present and does not assert the anyExtendedKeyUsage OID, all usages in the request MUST be present in the certificate. The certificate extension may contain more usages than requested.

Where the client does not require any particular extended key usage, the client can specify an empty SEQUENCE. This may be used to override extended key usage requirements imposed in the validation policy specified by valPolId.

SCVP clients SHOULD support extendedKeyUsages, and SCVP servers MUST support extendedKeyUsages.

3.2.4.10. specifiedKeyUsages

The extended key usage extension ([PKIX-1], Section 4.2.1.13) defines more specific technical purposes, in addition to or in place of the purposes indicated in the key usage extension, for which the certified public key may be used. If the client requires that a particular value (or values) appear in the extended key usage extension, then it can specify the required usage(s) using the same semantics as defined in [PKIX-1]. For example, if the client obtained the certificate in the context of a TLS server, it might require that the server certificate include the extended key usage structure with the id-kp-serverAuth object identifier. In this case, the client would include a specifiedKeyUsages item in the request and assert the id-kp-serverAuth object identifier.

If one or more specified usages are included in the request, the certificate MUST contain the extended key usage extension, and all usages specified in the request MUST be present in the certificate extension. The certificate extension may contain more usages than specified in the request. Specified key usages are not satisfied by the presence of the anyExtendedKeyUsage OID.

Where the client does not require any particular extended key usage, the client can specify an empty SEQUENCE. This may be used to override specified key usage requirements imposed in the validation policy specified by valPolId.

SCVP clients SHOULD support specifiedKeyUsages, and SCVP servers MUST support specifiedKeyUsages.

3.2.5. responseFlags

The optional responseFlags item allows the client to indicate which optional features in the CVResponse it wants the server to include. If the default values for all of the flags are used, then the responseFlags item MUST NOT be included in the request.

The syntax of the responseFlags item is:

```
ResponseFlags ::= SEQUENCE {
  fullRequestInResponse      [0] BOOLEAN DEFAULT FALSE,
  responseValidationPolByRef [1] BOOLEAN DEFAULT TRUE,
  protectResponse            [2] BOOLEAN DEFAULT TRUE,
  cachedResponse             [3] BOOLEAN DEFAULT TRUE }
```

Each of the response flags is described in the following sections.

3.2.5.1. fullRequestInResponse

By default, the server includes a hash of the request in non-cached responses to allow the client to identify the response. If the client wants the server to include the full request in the non-cached response, fullRequestInResponse is set to TRUE. The main reason a client would request the server to include the full request in the response is to archive the request-response exchange in a single object. That is, the client wants to archive a single object that includes both request and response.

SCVP clients and servers MUST support the default behavior. SCVP clients MAY support requesting and processing the full request. SCVP servers SHOULD support returning the full request.

3.2.5.2. responseValidationPolByRef

The responseValidationPolByRef item controls whether the response includes just a reference to the policy or a reference to the policy plus all the parameters by value of the policy used to process the request. The response MUST contain a reference to the validation policy. If the client wants the validation policy parameters to be included by value also, then responseValidationPolByRef is set to FALSE. The main reason a client would request the server to include validation policy to be included by value is to archive the request-response exchange in a single object. That is, the client wants to archive the CVResponse and have it include every aspect of the validation policy.

SCVP clients MUST support requesting and processing the validation policy by reference, and SCVP servers MUST support returning the validation policy by reference. SCVP clients MAY support requesting and processing the validation policy by values. SVCP servers SHOULD support returning the validation policy by values.

3.2.5.3. protectResponse

The protectResponse item indicates whether the client requires the server to protect the response. If the client is performing full certification path validation on the response and it is not concerned about the source of the response, then the client does not benefit from a digital signature or MAC on the response. In this case, the client can indicate to the server that protecting the message is unnecessary. However, the server is always permitted to return a protected response.

SCVP clients that support delegated path discovery (DPD) as defined in [RQMTS] MUST support setting this value to FALSE.

SCVP clients that support delegated path validation (DPV) as defined in [RQMTS] require an authenticated response. Unless a protected transport mechanism (such as TLS) is used, such clients MUST always set this value to TRUE or omit the responseFlags item entirely, which requires the server to return a protected response.

SCVP servers MUST support returning protected responses, and SCVP servers SHOULD support returning unprotected responses. Based on local policy, the server can be configured to return protected or unprotected responses if this value is set to FALSE. If, based on local policy, the server is unable to return protected responses, then the server MUST return an error if this value is set to TRUE.

3.2.5.4. cachedResponse

The cachedResponse item indicates whether the client will accept a cached response. To enhance performance and limit the exposure of signing keys, an SCVP service may be designed to cache responses until new revocation information is expected. Where cachedResponse is set to TRUE, the client will accept a previously cached response.

Clients may insist on creation of a fresh response to protect against a replay attack and ensure that information is up to date. Where cachedResponse is FALSE, the client will not accept a cached response. To ensure that a response is fresh, the client MUST also include the requestNonce as defined in Section 3.4.

Servers MUST process the cachedResponse flag. Where cachedResponse is FALSE, servers that cannot produce fresh responses MUST reply with an error message. Servers MAY choose to provide fresh responses even where cachedResponse is set to TRUE.

3.2.6. serverContextInfo

The optional serverContextInfo item, if present, contains context from a previous request-response exchange with the same SCVP server. It allows the server to return more than one certification path for the same certificate to the client. For example, if a server constructs a particular certification path for a certificate, but the client finds it unacceptable, the client can then send the same query back to the server with the serverContextInfo from the first response, and the server will be able to provide a different certification path (if another one can be found).

Contents of the serverContextInfo are opaque to the SCVP client. That is, the client only knows that it needs to return the value provided by the server with the subsequent request to get a different certification path. Note that the subsequent query needs to be identical to the previous query with the exception of the following:

- requestNonce,

- serverContextInfo, and

- the client's digital signature or MAC on the request.

SCVP clients MAY support serverContextInfo, and SCVP servers SHOULD support serverContextInfo.

3.2.7. validationTime

The optional validationTime item, if present, tells the date and time relative to which the SCVP client wants the server to perform the checks. If the validationTime is not present, the server MUST perform the validation using the date and time at which the server processes the request. If the validationTime is present, it MUST be encoded as GeneralizedTime. The validationTime provided MUST be a retrospective time since the server can only perform a validity check using the current time (default) or previous time. A server can ignore the validationTime provided in the request if the time is within the clock skew of the server's current time.

The revocation status information is obtained with respect to the validation time. When specifying a validation time other than the current time, the validation time should not necessarily be identical to the time when the private key was used. The validation time specified by the client may be adjusted to compensate for:

1) time for the end-entity to realize that its private key has been, or could possibly be, compromised, and/or

2) time for the end-entity to report the key compromise, and/or

3) time for the revocation authority to process the revocation request from the end-entity, and/or

4) time for the revocation authority to update and distribute the revocation status information.

GeneralizedTime values MUST be expressed in Universal Coordinated Time (UTC) (which is also known as Greenwich Mean Time and Zulu time) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even when the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds.

The information in the corresponding CertReply item in the response MUST be formatted as if the server created the response at the time indicated in the validationTime. However, if the server does not have appropriate historical information, the server MUST return an error response.

SCVP servers MUST apply a clock skew to the validation time to allow for minor time synchronization errors. The default value is 10 minutes. If the server uses a value other than the default, it MUST include the clock skew value in the validation policy response.

SCVP clients MAY support validationTime other than the current time. SCVP servers MUST support using its current time, and SHOULD support the client setting the validationTime in the request.

3.2.8. intermediateCerts

The optional intermediateCerts item may help the SCVP server create valid certification paths. The intermediateCerts item, when present, provides certificates that the server MAY use when forming a certification path. When building certification paths, the server MAY use the certificates in the intermediateCerts item in addition to any other certificates that the server can access. When present, the intermediateCerts item MUST contain at least one certificate, and

the intermediateCerts item MUST be structured as a CertBundle. The certificates in the intermediateCerts item MUST NOT be considered as valid by the server just because they are present in this item.

The CertBundle type contains one or more certificates. The order of the entries in the bundle is not important. CertBundle has the following syntax:

```
CertBundle ::= SEQUENCE SIZE (1..MAX) OF Certificate
```

SCVP clients SHOULD support intermediateCerts, and SCVP servers MUST support intermediateCerts.

3.2.9. revInfos

The optional revInfos item specifies revocation information such as CRLs, delta CRLs [PKIX-1], and OCSP responses [OCSP] that the SCVP server MAY use when validating certification paths. The purpose of the revInfos item is to provide revocation information to which the server might not otherwise have access, such as an OCSP response that the client received along with the certificate. Note that the information in the revInfos item might not be used by the server. For example, the revocation information might be associated with certificates that the server does not use in the certification path that it constructs.

Clients SHOULD be courteous to the SCVP server by separating CRLs and delta CRLs. However, since the two share a common syntax, SCVP servers SHOULD accept delta CRLs even if they are identified as regular CRLs by the SCVP client.

CRLs, delta CRLs, and OCSP responses can be provided as revocation information. If needed, additional object identifiers can be assigned for additional revocation information types in the future.

The revInfos item uses the RevocationInfos type, which has the following syntax:

```
RevocationInfos ::= SEQUENCE SIZE (1..MAX) OF RevocationInfo

RevocationInfo ::= CHOICE {
  crl                   [0] CertificateList,
  delta-crl             [1] CertificateList,
  ocsp                  [2] OCSPResponse,
  other                 [3] OtherRevInfo }
```

```
OtherRevInfo ::= SEQUENCE {
  riType                    OBJECT IDENTIFIER,
  riValue                   ANY DEFINED BY riType }
```

3.2.10. producedAt

The client MAY allow the server to use a cached SCVP response. When doing so, the client MAY use the producedAt item to express requirements on the freshness of the cached response. The producedAt item tells the earliest date and time at which an acceptable cached response could have been produced. The producedAt item represents the date and time in UTC, using the GeneralizedTime type. The value in the producedAt item is independent of the validation time.

GeneralizedTime value MUST be expressed in UTC, as defined in Section 3.2.7.

SCVP clients MAY support using producedAt values in the request. SCVP servers MAY support the producedAt values in the request. SCVP servers that support cached responses SHOULD support the producedAt value in requests.

3.2.11. queryExtensions

The optional queryExtensions item contains extensions. If present, each extension in the sequence extends the query. This specification does not define any extensions; the facility is provided to allow future specifications to extend SCVP. The syntax for Extensions is imported from [PKIX-1]. The queryExtensions item, when present, MUST contain a sequence of Extension items, and each of the extensions MUST contain extnID, critical, and extnValue items. Each of these is described in the following sections.

3.2.11.1. extnID

The extnID item is an identifier for the extension. It contains the object identifier that names the extension.

3.2.11.2. critical

The critical item is a BOOLEAN. Each extension is designated as either critical (with a value of TRUE) or non-critical (with a value of FALSE). By default, the extension is non-critical. An SCVP server MUST reject the query if it encounters a critical extension that it does not recognize; however, a non-critical extension MAY be ignored if it is not recognized, but MUST be processed if it is recognized.

3.2.11.3. extnValue

The extnValue item contains an OCTET STRING. Within the OCTET STRING is the extension value. An ASN.1 type is specified for each extension, identified by the associated extnID object identifier.

3.3. requestorRef

The optional requestorRef item contains a list of names identifying SCVP servers, and it is intended for use in environments where SCVP relay is employed. Although requestorRef is encoded as a SEQUENCE, no order is implied. The requestorRef item is used to detect looping in some configurations. The value and use of requestorRef are described in Section 7.

Conforming SCVP clients MAY support specification of the requestorRef value. Conforming SCVP server implementations MUST process the requestorRef value if present. If the SCVP client includes a requestorRef value in the request, then the SCVP server MUST return the same value in a non-cached response. The SCVP server MAY omit the requestorRef value from cached SCVP responses.

The requestorRef item MUST be a sequence of GeneralName. No provisions are made to ensure uniqueness of the requestorRef GeneralName values.

3.4. requestNonce

The optional requestNonce item contains a request identifier generated by the SCVP client. If the client includes a requestNonce value in the request, it is expressing a preference that the SCVP server SHOULD return a non-cached response. If the server returns a non-cached response, it MUST include the value of requestNonce from the request in the response as the respNonce item; however, the server MAY return a cached response which MUST NOT have a respNonce.

SCVP clients that insist on creation of a fresh response (e.g., to protect against a replay attack or ensure information is up to date) MUST support requestNonce. Conforming SCVP server implementations MUST process the requestNonce value if present.

If the client includes a requestNonce and also sets the cachedResponse flag to FALSE as described in Section 3.2.5.4, the client is indicating that the SCVP server MUST return either a non-cached response including the respNonce or an error response. The client SHOULD include a requestNonce item in every request to prevent

an attacker from acting as a man-in-the-middle by replaying old responses from the server. The requestNonce value SHOULD change with every request sent by the client.

The client MUST NOT set the cachedResponse flag to FALSE without also including a requestNonce. A server receiving such a request SHOULD return an invalidRequest error response.

The requestNonce item, if present, MUST be an OCTET STRING that was generated exclusively for this request.

3.5. requestorName

The optional requestorName item is used by the client to include an identifier in the request. The client MAY include this information for the DPV server to copy into the response.

Conforming SCVP clients MAY support specification of this item in requests. SCVP servers MUST be able to process requests that include this item.

3.6. responderName

The optional responderName item is used by the client to indicate the identity of the SCVP server that the client expects to sign the SCVP response if the response is digitally signed. The responderName item SHOULD only be included if:

1. the request is either unprotected or digitally signed (i.e., is not protected using a MAC), and

2. the responseFlags item is either absent or present with the protectResponse set to TRUE.

Conforming SCVP clients MAY support specification of this item in requests. SCVP servers MUST be able to process requests that include this item. SCVP servers that maintain a single private key for signing SCVP responses or that are unable to return digitally signed responses MAY ignore the value in this item. SCVP servers that maintain more than one private key for signing SCVP responses SHOULD either (a) digitally sign the response using a private key that corresponds to a certificate that includes the name specified in responderName in either subject field or subjectAltName extension or (b) return a error indicating that the server does not possess a certificate that asserts the specified name.

3.7. requestExtensions

The OPTIONAL requestExtensions item contains extensions. If present, each extension in the sequence extends the request. This specification does not define any extensions; the facility is provided to allow future specifications to extend SCVP. The syntax for Extensions is imported from [PKIX-1]. The requestExtensions item, when present, MUST contain a sequence of Extension items, and each of the extensions MUST contain extnID, critical, and extnValue items. Each of these is described in the following sections.

3.7.1. extnID

The extnID item is an identifier for the extension. It contains the object identifier that names the extension.

3.7.2. critical

The critical item is a BOOLEAN. Each extension is designated as either critical (with a value of TRUE) or non-critical (with a value of FALSE). By default, the extension is non-critical. An SCVP server MUST reject the query if it encounters a critical extension it does not recognize. A non-critical extension MAY be ignored if it is not recognized, but MUST be processed if it is recognized.

3.7.3. extnValue

The extnValue item contains an OCTET STRING. Within the OCTET STRING is the extension value. An ASN.1 type is specified for each extension, identified by the associated extnID object identifier.

3.8. signatureAlg

The signatureAlg item contains an AlgorithmIdentifier indicating which algorithm the server should use to sign the response message. The signatureAlg item SHOULD only be included if:

1. the request is either unprotected or digitally signed (i.e., is not protected using a MAC), and

2. the responseFlags item is either absent or present with the protectResponse set to TRUE.

If included, the signatureAlg item SHOULD specify one of the signature algorithms specified in the signatureGeneration item of the server’s validation policy response message.

SCVP servers MUST be able to process requests that include this item. If the server is returning a digitally signed response to this message, then:

1. If the signatureAlg item is present and specifies an algorithm that is included in the signatureGeneration item of the server's validation policy response message, the server MUST sign the response using the signature algorithm specified in signatureAlg.

2. Otherwise, if the signatureAlg item is absent or is present but specifies an algorithm that is not supported by the server, the server MUST sign the response using the server's default signature algorithm as specified in the signatureGeneration item of the server's validation policy response message.

3.9. hashAlg

The hashAlg item contains an object identifier indicating which hash algorithm the server should use to compute the hash value for the requestHash item in the response. SCVP clients SHOULD NOT include this item if fullRequestInResponse is set to TRUE. If included, the hashAlg item SHOULD specify one of the hash algorithms specified in the hashAlgorithms item of the server's validation policy response message.

SCVP servers MUST be able to process requests that include this item. If the server is returning a response to this message that includes a requestHash, then:

1. If the hashAlg item is present and specifies an algorithm that is included in the hashAlgorithms item of the server's validation policy response message, the server MUST use the algorithm specified in hashAlg to compute the requestHash.

2. Otherwise, if the hashAlg item is absent or is present but specifies an algorithm that is not supported by the server, the server MUST compute the requestHash using the server's default hash algorithm as specified in the hashAlgorithms item of the server's validation policy response message.

3.10. requestorText

SCVP clients MAY use the requestorText item to provide text for inclusion in the corresponding response. For example, this field may describe the nature or reason for the request.

Conforming SCVP client implementations MAY support inclusion of this item in requests. Conforming SCVP server implementations MUST accept requests that include this item. When generating non-cached responses, conforming SCVP server implementations MUST copy the contents of this item into the requestorText item in the corresponding response (see Section 4.13).

3.11. SCVP Request Authentication

It is a matter of local policy what validation policy the server uses when authenticating requests. When authenticating protected SCVP requests, the SCVP servers SHOULD use the validation algorithm defined in Section 6 of [PKIX-1].

If the certificate used to validate a SignedData validation request includes the key usage extension ([PKIX-1], Section 4.2.1.3), it MUST have either the digital signature bit set, the non-repudiation bit set, or both bits set.

If the certificate used to validate an AuthenticatedData validation request includes the key usage extension, it MUST have the key agreement bit set.

If the certificate used on a validation request contains the extended key usage extension ([PKIX-1], Section 4.2.1.13), the server SHALL verify that it contains the SCVP client OID, the anyExtendedKeyUsage OID, or another OID acceptable to the server. The SCVP client OID is defined as follows:

```
id-kp OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
          dod(6) internet(1) security(5) mechanisms(5) pkix(7) 3 }

id-kp-scvpClient             OBJECT IDENTIFIER ::= { id-kp 16 }
```

If a protected request fails to meet the validation policy of the server, it MUST be treated as an unauthenticated request.

4. Validation Response

An SCVP server response to the client MUST be a single CVResponse item. When a CVResponse is encapsulated in a MIME body part, application/scvp-cv-response MUST be used.

There are a number of forms of an SCVP response:

1. A success response to a request that has protectResponse set to FALSE. These responses SHOULD NOT be protected by the server.

2. The server MUST protect all other success responses. If the server is unable to return a protected success response due to local policy, then it MUST return an error response.

3. An error response to a request made over a protected transport such as TLS. These responses SHOULD NOT be protected by the server.

4. An error response to a request that has protectResponse set to FALSE. These responses SHOULD NOT be protected by the server.

5. An error response to an authenticated request. The server SHOULD protect these responses.

6. An error response to an AuthenticatedData request where MAC is valid. The server MUST protect these responses.

7. All other error responses MUST NOT be protected by the server.

Successful responses are made when the server has fully complied with the request. That is, the server was able to attempt to build a certification path using the referenced or supplied validation policy, and it was able to comply with all the requested parameters. If the server is unable to perform validations using the required validation policy or the request contains an unsupported option, then the server MUST return an error response.

For protected requests and responses, SCVP servers MUST support SignedData and SHOULD support AuthenticatedData. It is a matter of local policy which types are used. Where a protected response is required, SCVP servers MUST use SignedData or AuthenticatedData, even if the transaction is performed using a protected transport (e.g., TLS).

If the server is making a protected response to a protected request, then the server MUST use the same protection mechanism (SignedData or AuthenticatedData) as in the request.

An overview of the structure used for an unprotected response is provided below. Many details are not shown, but the way that SCVP makes use of CMS is clearly illustrated.

```
ContentInfo {
  contentType        id-ct-scvp-certValResponse,
                              -- (1.2.840.113549.1.9.16.1.11)
  content            CVResponse }
```

The protected response consists of a CVResponse encapsulated in either a SignedData or an AuthenticatedData, which is in turn encapsulated in a ContentInfo. That is, the EncapsulatedContentInfo field of either SignedData or AuthenticatedData consists of an eContentType field with a value of id-ct-scvp-certValResponse and an eContent field that contains a DER-encoded CVResponse.

The SCVP server MUST include its own certificate in the certificates field within SignedData. Other certificates MAY also be included.

The SCVP server MAY also provide one or more CRLs in the crls field within SignedData. The signerInfos field of SignedData MUST include exactly one SignerInfo. The SignedData MUST NOT include the unsignedAttrs field.

The signedAttrs field within SignerInfo MUST include the content-type and message-digest attributes defined in [CMS], and it SHOULD include the signing-certificate attribute as defined in [ESS]. Within the signing-certificate attribute, the first certificate identified in the sequence of certificate identifiers MUST be the certificate of the SCVP server. The inclusion of other certificate identifiers in the signing-certificate attribute is OPTIONAL. The inclusion of policies in the signing-certificate is OPTIONAL.

The recipientInfos field of AuthenticatedData MUST include exactly one RecipientInfo, which contains information for the client that sent the request. The AuthenticatedData MUST NOT include the unauthAttrs field.

The CVResponse item contains the server’s response. The CVResponse MUST contain the cvResponseVersion, serverConfigurationID, producedAt, and responseStatus items. The CVResponse MAY also contain the respValidationPolicy, requestRef, requestorRef, requestorName, replyObjects, respNonce, serverContextInfo, and cvResponseExtensions items. The replyObjects item MUST contain exactly one CertReply item for each certificate requested. The requestorRef item MUST be included if the request included a requestorRef item and a non-cached response is provided. The respNonce item MUST be included if the request included a requestNonce item and a non-cached response is provided.

The CVResponse MUST have the following syntax:

```
CVResponse ::= SEQUENCE {
  cvResponseVersion         INTEGER,
  serverConfigurationID     INTEGER,
  producedAt                GeneralizedTime,
  responseStatus            ResponseStatus,
  respValidationPolicy  [0] RespValidationPolicy OPTIONAL,
  requestRef            [1] RequestReference OPTIONAL,
  requestorRef          [2] GeneralNames OPTIONAL,
  requestorName         [3] GeneralNames OPTIONAL,
  replyObjects          [4] ReplyObjects OPTIONAL,
  respNonce             [5] OCTET STRING OPTIONAL,
  serverContextInfo     [6] OCTET STRING OPTIONAL,
  cvResponseExtensions  [7] Extensions OPTIONAL,
  requestorText         [8] UTF8String (SIZE (1..256)) OPTIONAL }
```

Conforming SCVP servers MAY be capable of constructing a CVResponse that includes the serverContextInfo or cvResponseExtensions items. Conforming SCVP servers MUST be capable of constructing a CVResponse with any of the remaining optional items. Conforming SCVP clients MUST be capable of processing a CVResponse with the following optional items: respValidationPolicy, requestRef, requestorName, replyObjects, and respNonce.

Conforming SCVP clients that are capable of including requestorRef in a request MUST be capable of processing a CVResponse that includes the requestorRef item. Conforming SCVP clients MUST be capable of processing a CVResponse that includes the serverContextInfo or cvResponseExtensions items. Conforming clients MUST be able to determine if critical extensions are present in the cvResponseExtensions item.

4.1. cvResponseVersion

The syntax and semantics of cvResponseVersion are the same as cvRequestVersion as described in Section 3.1. The cvResponseVersion MUST match the cvRequestVersion in the request. If the server cannot generate a response with a matching version number, then the server MUST return an error response that indicates the highest version number that the server supports as the version number.

4.2. serverConfigurationID

The server configuration ID item represents the version of the SCVP server configuration when it processed the request. See Section 6.4 for details.

4.3. producedAt

The producedAt item tells the date and time at which the SCVP server generated the response. The producedAt item MUST be expressed in UTC, and it MUST be interpreted as defined in Section 3.2.7. This value is independent of the validation time.

4.4. responseStatus

The responseStatus item gives status information to the SCVP client about its request. The responseStatus item has a numeric status code and an optional string that is a sequence of characters from the ISO/IEC 10646-1 character set encoded with the UTF-8 transformation format defined in [UTF8].

The string MAY be used to transmit status information. The client MAY choose to display the string to a human user. However, because there is often no way to know the languages understood by a human user, the string may be of little or no assistance.

The responseStatus item uses the ResponseStatus type, which has the following syntax:

```
ResponseStatus ::= SEQUENCE {
  statusCode            CVStatusCode DEFAULT  okay,
  errorMessage          UTF8String OPTIONAL }

CVStatusCode ::= ENUMERATED {
  okay                                (0),
  skipUnrecognizedItems               (1),
  tooBusy                            (10),
  invalidRequest                     (11),
  internalError                      (12),
  badStructure                       (20),
  unsupportedVersion                 (21),
  abortUnrecognizedItems             (22),
  unrecognizedSigKey                 (23),
  badSignatureOrMAC                  (24),
  unableToDecode                     (25),
  notAuthorized                      (26),
  unsupportedChecks                  (27),
  unsupportedWantBacks               (28),
  unsupportedSignatureOrMAC          (29),
  invalidSignatureOrMAC              (30),
  protectedResponseUnsupported       (31),
  unrecognizedResponderName          (32),
  relayingLoop                       (40),
  unrecognizedValPol                 (50),
```

```
      unrecognizedValAlg                (51),
      fullRequestInResponseUnsupported  (52),
      fullPolResponseUnsupported        (53),
      inhibitPolicyMappingUnsupported   (54),
      requireExplicitPolicyUnsupported  (55),
      inhibitAnyPolicyUnsupported       (56),
      validationTimeUnsupported         (57),
      unrecognizedCritQueryExt          (63),
      unrecognizedCritRequestExt        (64) }
```

The CVStatusCode values have the following meaning:

0 The request was fully processed.
1 The request included some unrecognized non-critical extensions; however, processing was able to continue ignoring them.
10 Too busy; try again later.
11 The server was able to decode the request, but there was some other problem with the request.
12 An internal server error occurred.
20 The structure of the request was wrong.
21 The version of request is not supported by this server.
22 The request included unrecognized items, and the server was not able to continue processing.
23 The server could not validate the key used to protect the request.
24 The signature or message authentication code did not match the body of the request.
25 The encoding was not understood.
26 The request was not authorized.
27 The request included unsupported checks items, and the server was not able to continue processing.
28 The request included unsupported wantBack items, and the server was not able to continue processing.
29 The server does not support the signature or message authentication code algorithm used by the client to protect the request.
30 The server could not validate the client’s signature or message authentication code on the request.
31 The server could not generate a protected response as requested by the client.
32 The server does not have a certificate matching the requested responder name.
40 The request was previously relayed by the same server.
50 The request contained an unrecognized validation policy reference.
51 The request contained an unrecognized validation algorithm OID.
52 The server does not support returning the full request in the response.

53 The server does not support returning the full validation policy by value in the response.
54 The server does not support the requested value for inhibit policy mapping.
55 The server does not support the requested value for require explicit policy.
56 The server does not support the requested value for inhibit anyPolicy.
57 The server only validates requests using current time.
63 The query item in the request contains a critical extension whose OID is not recognized.
64 The request contains a critical request extension whose OID is not recognized.

Status codes 0-9 are reserved for codes that indicate the request was processed by the server and therefore MUST be sent in a success response. Status codes 10 and above indicate an error and MUST therefore be sent in an error response.

4.5. respValidationPolicy

The respValidationPolicy item contains either a reference to the full validation policy or the full policy by value used by the server to validate the request. It MUST be present in success responses and MUST NOT be present in error responses. The choice between returning the policy by reference or by value is controlled by the responseValidationPolByRef item in the request. The resultant validation policy is the union of the following:

1. Values from the request.

2. For values that are not explicitly included in the request, values from the validation policy specified by reference in the request.

The RespValidationPolicy syntax is:

```
RespValidationPolicy ::= ValidationPolicy
```

The validationPolicy item is defined in Section 3.2.4. When responseValidationPolByRef is set to FALSE in the request, all items in the validationPolicy item MUST be populated. When responseValidationPolByRef is set to TRUE, OPTIONAL items in the validationPolicy item only need to be populated for items for which the value in the request differs from the value from the referenced validation policy.

Conforming SCVP clients MUST be capable of processing the validation policy by reference. SCVP clients MAY be capable of processing the optional items in the validation policy.

Conforming SCVP server implementations MUST be capable of asserting the policy by reference, and MUST be capable of including the optional items.

4.6. requestRef

The requestRef item allows the SCVP client to identify the request that corresponds to this response from the server. It associates the response to a particular request using either a hash of the request or a copy of CVRequest from the request.

The requestRef item does not provide authentication, but does allow the client to determine that the request was not maliciously modified.

The requestRef item allows the client to associate a response with a request. The requestNonce provides an alternative mechanism for matching requests and responses. When the fullRequest alternative is used, the response provides a single data structure that is suitable for archive of the transaction.

The requestRef item uses the RequestReference type, which has the following syntax:

```
RequestReference ::= CHOICE {
  requestHash       [0] HashValue, -- hash of CVRequest
  fullRequest       [1] CVRequest }
```

SCVP clients MUST support requestHash, and they MAY support fullRequest. SCVP servers MUST support using requestHash, and they SHOULD support using fullRequest.

4.6.1. requestHash

The requestHash item is the hash of the CVRequest. The one-way hash function used to compute the hash of the CVRequest is as specified in Section 3.9. The requestHash item serves two purposes. First, it allows a client to determine that the request was not maliciously modified. Second, it allows the client to associate a response with a request when using connectionless protocols. The requestNonce provides an alternative mechanism for matching requests and responses.

The requestHash item uses the HashValue type, which has the following syntax:

```
HashValue ::= SEQUENCE {
  algorithm       AlgorithmIdentifier DEFAULT { algorithm sha-1 },
  value           OCTET STRING }

sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    oiw(14) secsig(3) algorithm(2) 26 }
```

The algorithm identifier for SHA-1 is imported from [PKIX-ALG]. It is repeated here for convenience.

4.6.2. fullRequest

Like requestHash, the fullRequest alternative allows a client to determine that the request was not maliciously modified. It also provides a single data structure that is suitable for archive of the transaction.

The fullRequest item uses the CVRequest type. The syntax and semantics of the CVRequest type are described in Section 3.

4.7. requestorRef

The optional requestorRef item is used by the client to identify the original requestor in cases where SCVP relay is used. The value is only of local significance to the client. If the SCVP client includes a requestorRef value in the request, then the SCVP server MUST return the same value if the server is generating a non-cached response.

4.8. requestorName

The optional requestorName item is used by the server to return one or more identities associated with the client in the response.

The SCVP server MAY choose to include any or all of the following:

(1) the identity asserted by the client in the requestorName item of the request,

(2) an authenticated identity for the client from a certificate or other credential used to authenticate the request, or

(3) a client identifier from an out-of-band mechanism.

Alternatively, the SCVP server MAY omit this item.

In the case of non-cached responses to authenticated requests, the SCVP server SHOULD return a requestor name.

SCVP servers that support authenticated requests SHOULD support this item.

SCVP clients MUST be able to process responses that include this item, although the item value might not impact the processing in any manner.

4.9. replyObjects

The replyObjects item returns requested objects to the SCVP client, each of which tells the client about a single certificate from the request. The replyObjects item MUST be present in the response, unless the response is reporting an error. The CertReply item MUST contain cert, replyStatus, replyValTime, replyChecks, and replyWantBacks items, and the CertReply item MAY contain the validationErrors, nextUpdate, and certReplyExtensions items.

A success response MUST contain one CertReply for each certificate specified in the queriedCerts item in the request. The order is important. The first CertReply in the sequence MUST correspond to the first certificate in the request, the second CertReply in the sequence MUST correspond to the second certificate in the request, and so on.

The checks item in the request determines the content of the replyChecks item in the response. The wantBack item in the request determines the content of the replyWantBacks item in the response. The queryExtensions items in the request controls the absence or the presence and content of the certReplyExtensions item in the response.

The replyObjects item uses the ReplyObjects type, which has the following syntax:

```
ReplyObjects ::= SEQUENCE SIZE (1..MAX) OF CertReply

CertReply ::= SEQUENCE {
  cert                       CertReference,
  replyStatus                ReplyStatus DEFAULT success,
  replyValTime               GeneralizedTime,
  replyChecks                ReplyChecks,
  replyWantBacks             ReplyWantBacks,
  validationErrors       [0] SEQUENCE SIZE (1..MAX) OF
                               OBJECT IDENTIFIER OPTIONAL,
  nextUpdate             [1] GeneralizedTime OPTIONAL,
  certReplyExtensions    [2] Extensions OPTIONAL }
```

4.9.1. cert

The cert item contains either the certificate or a reference to the certificate about which the client is requesting information. If the certificate was specified by reference in the request, the request included either the id-swb-pkc-cert or id-swb-aa-cert wantBack, and the server was able to obtain the referenced certificate, then this item MUST include the certificate. Otherwise, this item MUST include the same value as was used in the queriedCerts item in the request.

CertReference has the following syntax:

```
CertReference ::= CHOICE {
  pkc                   PKCReference,
  ac                    ACReference }
```

4.9.2. replyStatus

The replyStatus item gives status information to the client about the request for the specific certificate. Note that the responseStatus item is different from the replyStatus item. The responseStatus item is the status of the whole request, while the replyStatus item is the status for the individual query item.

The replyStatus item uses the ReplyStatus type, which has the following syntax:

```
ReplyStatus ::= ENUMERATED {
    success                   (0),
    malformedPKC              (1),
    malformedAC               (2),
    unavailableValidationTime (3),
    referenceCertHashFail     (4),
    certPathConstructFail     (5),
    certPathNotValid          (6),
    certPathNotValidNow       (7),
    wantBackUnsatisfied       (8) }
```

The meanings of the various ReplyStatus values are:

0 Success: all checks were performed successfully.
1 Failure: the public key certificate was malformed.
2 Failure: the attribute certificate was malformed.
3 Failure: historical data for the requested validation time is not available.
4 Failure: the server could not locate the reference certificate or the referenced certificate did not match the hash value provided.
5 Failure: no certification path could be constructed.

6 Failure: the constructed certification path is not valid with respect to the validation policy.
7 Failure: the constructed certification path is not valid with respect to the validation policy, but a query at a later time may be successful.
8 Failure: all checks were performed successfully; however, one or more of the wantBacks could not be satisfied.

Codes 1 and 2 are used to tell the client that the request was properly formed, but the certificate in question was not. This is especially useful to clients that do not parse certificates.

Code 7 is used to tell the client that a valid certification path was found with the exception that a certificate in the path is on hold, current revocation information is unavailable, or the validation time precedes the notBefore time in one or more certificates in the path.

For codes 1, 2, 3, and 4, the replyChecks and replyWantBacks items are not populated (i.e., they MUST be an empty sequence). For codes 5, 6, 7, and 8, replyChecks MUST include an entry corresponding to each check in the request; the replyWantBacks item is not populated.

4.9.3. replyValTime

The replyValTime item tells the time at which the information in the CertReply was correct. The replyValTime item represents the date and time in UTC, using GeneralizedTime type. The encoding rules for GeneralizedTime in Section 3.2.7 MUST be used.

Within the request, the optional validationTime item tells the date and time relative to which the SCVP client wants the server to perform the checks. If the validationTime is not present, the server MUST respond as if the client provided the date and time at which the server processes the request.

The information in the CertReply item MUST be formatted as if the server created this portion of the response at the time indicated in the validationTime item of the query. However, if the server does not have appropriate historical information, the server MAY either return an error or return information for a later time.

4.9.4. replyChecks

The replyChecks item contains the responses to the checks item in the query. The replyChecks item includes the object identifier (OID) from the query and an integer. The value of the integer indicates whether the requested check was successful. The OIDs in the checks item of the query are used to identify the corresponding replyChecks

values. Each OID specified in the checks item in the request MUST be matched by an OID in the replyChecks item of the response. In the case of an error response, the server MAY include additional checks in the response to further explain the error. Clients MUST ignore any unrecognized ReplyCheck included in the response.

The replyChecks item uses the ReplyChecks type, which has the following syntax:

```
ReplyChecks ::= SEQUENCE OF ReplyCheck

ReplyCheck ::= SEQUENCE {
  check                     OBJECT IDENTIFIER,
  status                    INTEGER DEFAULT 0 }
```

The status value for public key certification path building to a trusted root, { id-stc 1 }, can be one of the following:

```
0: Built a path
1: Could not build a path
```

The status value for public key certification path building to a trusted root along with simple validation processing, { id-stc 2 }, can be one of the following:

```
0: Valid
1: Not valid
```

The status value for public key certification path building to a trusted root along with complete status checking, { id-stc 3 }, can be one of the following:

```
0: Valid
1: Not valid
2: Revocation off-line
3: Revocation unavailable
4: No known source for revocation information
```

Revocation off-line means that the server or distribution point for the revocation information was connected to successfully without a network error but either no data was returned or if data was returned it was stale. Revocation unavailable means that a network error was returned when an attempt was made to reach the server or distribution point. No known source for revocation information means that the server was able to build a valid certification path but was unable to locate a source for revocation information for one or more certificates in the path.

The status value for AC issuer certification path building to a trusted root, { id-stc 4 }, can be one of the following:

0: Built a path
1: Could not build a path

The status value for AC issuer certification path building to a trusted root along with simple validation processing, { id-stc 5 }, can be one of the following:

0: Valid
1: Not valid

The status value for AC issuer certification path building to a trusted root along with complete status checking, { id-stc 6 }, can be one of the following:

0: Valid
1: Not valid
2: Revocation off-line
3: Revocation unavailable
4: No known source for revocation information

The status value for revocation status checking of an AC as well as AC issuer certification path building to a trusted root along with complete status checking, { id-stc 7 }, can be one of the following:

0: Valid
1: Not valid
2: Revocation off-line
3: Revocation unavailable
4: No known source for revocation information

4.9.5. replyWantBacks

The replyWantBacks item contains the responses to the wantBack item in the request. The replyWantBacks item includes the object identifier (OID) from the wantBack item in the request and an OCTET STRING. Within the OCTET STRING is the requested value. The OIDs in the wantBack item in the request are used to identify the corresponding reply value. The OIDs in the replyWantBacks item MUST match the OIDs in the wantBack item in the request. For a non-error response, replyWantBacks MUST include exactly one ReplyWantBack for each wantBack specified in the request (excluding id-swb-pkc-cert and id-swb-ac-cert, where the requested information is included in the cert item).

The replyWantBacks item uses the ReplyWantBacks type, which has the following syntax:

```
ReplyWantBacks ::= SEQUENCE OF ReplyWantBack

ReplyWantBack::= SEQUENCE {
  wb                        OBJECT IDENTIFIER,
  value                     OCTET STRING }
```

The OCTET STRING value for the certification path used to verify the certificate in the request, { id-swb 1 }, contains the CertBundle type. The syntax and semantics of the CertBundle type are described in Section 3.2.8. This CertBundle includes all the certificates in the path, starting with the end certificate and ending with the certificate issued by the trust anchor.

The OCTET STRING value for the proof of revocation status, { id-swb 2 }, contains the RevInfoWantBack type. The RevInfoWantBack type is a SEQUENCE of the RevocationInfos type and an optional CertBundle. The syntax and semantics of the RevocationInfos type are described in Section 3.2.9. The CertBundle MUST be included if any certificates required to validate the revocation information were not returned in the id-swb-pkc-best-cert-path or id-swb-pkc-all-cert-paths wantBack. The CertBundle MUST include all such certificates, but there are no ordering requirements.

```
RevInfoWantBack ::= SEQUENCE {
  revocationInfo            RevocationInfos,
  extraCerts                CertBundle OPTIONAL }
```

The OCTET STRING value for the public key information, { id-swb 4 }, contains the SubjectPublicKeyInfo type. The syntax and semantics of the SubjectPublicKeyInfo type are described in [PKIX-1].

The OCTET STRING value for the AC issuer certification path used to verify the certificate in the request, { id-swb 5 }, contains the CertBundle type. The syntax and semantics of the CertBundle type are described in Section 3.2.8. This CertBundle includes all the certificates in the path, beginning with the AC issuer certificate and ending with the certificate issued by the trust anchor.

The OCTET STRING value for the proof of revocation status of the AC issuer certification path, { id-swb 6 }, contains the RevInfoWantBack type. The RevInfoWantBack type is a SEQUENCE of the RevocationInfos type and an optional CertBundle. The syntax and semantics of the RevocationInfos type are described in Section 3.2.9. The CertBundle

MUST be included if any certificates required to validate the revocation information were not returned in the id-aa-cert-path wantBack. The CertBundle MUST include all such certificates, but there are no ordering requirements.

The OCTET STRING value for the proof of revocation status of the attribute certificate, { id-swb 7 }, contains the RevInfoWantBack type. The RevInfoWantBack type is a SEQUENCE of the RevocationInfos type and an optional CertBundle. The syntax and semantics of the RevocationInfos type are described in Section 3.2.9. The CertBundle MUST be included if any certificates required to validate the revocation information were not returned in the id-swb-aa-cert-path wantBack. The CertBundle MUST include all such certificates, but there are no ordering requirements.

The OCTET STRING value for returning all paths, { id-swb 12 }, contains an ASN.1 type CertBundles, as defined below. The syntax and semantics of the CertBundle type are described in Section 3.2.8. Each CertBundle includes all the certificates in one path, starting with the end certificate and ending with the certificate issued by the trust anchor.

```
CertBundles ::= SEQUENCE SIZE (1..MAX) OF CertBundle
```

The OCTET STRING value for relayed responses, { id-swb 9 }, contains an ASN.1 type SCVPResponses, as defined below. If the SCVP server used information obtained from other SCVP servers when generating this response, then SCVPResponses MUST include each of the SCVP responses received from those servers. If the SCVP server did not use information obtained from other SCVP servers when generating the response, then SCVPResponses MUST be an empty sequence.

```
SCVPResponses ::= SEQUENCE OF ContentInfo
```

The OCTET STRING value for the proof of revocation status of the path's target certificate, { id-swb-13 }, contains the RevInfoWantBack type. The RevInfoWantBack type is a SEQUENCE of the RevocationInfos type and an optional CertBundle. The syntax and semantics of the RevocationInfos type are described in Section 3.2.9. The CertBundle MUST be included if any certificates required to validate the revocation information were not returned in the id-swb-pkc-best-cert-path or id-swb-pkc-all-cert-paths wantBack. The CertBundle MUST include all such certificates, but there are no ordering requirements.

The OCTET STRING value for the proof of revocation status of the intermediate certificates in the path, { id-swb 14 }, contains the RevInfoWantBack type. The RevInfoWantBack type is a SEQUENCE of the

RevocationInfos type and an optional CertBundle. The syntax and semantics of the RevocationInfos type are described in Section 3.2.9. The CertBundle MUST be included if any certificates required to validate the revocation information were not returned in the id-swb-pkc-best-cert-path or id-swb-pkc-all-cert-paths wantBack. The CertBundle MUST include all such certificates, but there are no ordering requirements.

4.9.6. validationErrors

The validationErrors item MUST only be present in failure responses. If present, it MUST contain one or more OIDs representing the reason the validation failed (validation errors for the basic validation algorithm and name validation algorithm are defined in Sections 3.2.4.2.2 and 3.2.4.2.4). The validationErrors item SHOULD only be included when the replyStatus is 3, 5, 6, 7, or 8. SCVP servers are not required to specify all of the reasons that validation failed. SCVP clients MUST NOT assume that the OIDs included in validationErrors represent all of the validation errors for the certification path.

4.9.7. nextUpdate

The nextUpdate item tells the time at which the server expects a refresh of information regarding the validity of the certificate to become available. The nextUpdate item is especially interesting if the certificate revocation status information is not available or the certificate is suspended. The nextUpdate item represents the date and time in UTC, using the GeneralizedTime type. The encoding rules for GeneralizedTime in Section 3.2.7 MUST be used.

4.9.8. certReplyExtensions

The certReplyExtensions item contains the responses to the queryExtensions item in the request. The certReplyExtensions item uses the Extensions type defined in [PKIX-1]. The object identifiers (OIDs) in the queryExtensions item in the request are used to identify the corresponding reply values. The certReplyExtensions item, when present, contains a sequence of Extension items, each of which contains an extnID item, a critical item, and an extnValue item.

The extnID item is an identifier for the extension. It contains the OID that names the extension, and it MUST match one of the OIDs in the queryExtensions item in the request.

The critical item is a BOOLEAN, and it MUST be set to FALSE.

The extnValue item contains an OCTET STRING. Within the OCTET STRING is the extension value. An ASN.1 type is specified for each extension, identified by the associated extnID object identifier.

4.10. respNonce

The respNonce item contains an identifier to bind the request to the response.

If the client includes a requestNonce value in the request and the server is generating a specific non-cached response to the request then the server MUST return the same value in the response.

If the server is using a cached response to the request then it MUST omit the respNonce item.

If the server is returning a specific non-cached response to a request without a nonce, then the server MAY include a message-specific nonce. For digitally signed messages, the server MAY use the value of the message-digest attribute in the signedAttrs within SignerInfo of the request as the value in the respNonce item.

The requestNonce item uses the OCTET STRING type.

Conforming client implementations MUST be able to process a response that includes this item. Conforming servers MUST support respNonce.

4.11. serverContextInfo

The serverContextInfo item in a response is a mechanism for the server to pass some opaque context information to the client. If the client does not like the certification path returned, it can make a new query and pass along this context information.

Section 3.2.6 contains information about the client's usage of this item.

The context information is opaque to the client, but it provides information to the server that ensures that a different certification path will be returned (if another one can be found). The context information could indicate the state of the server, or it could contain a sequence of hashes of certification paths that have already been returned to the client. The protocol does not dictate any structure or requirements for this item. However, implementers should review the Security Considerations section of this document before selecting a structure.

Servers that are incapable of returning additional paths MUST NOT include the serverContextInfo item in the response.

4.12. cvResponseExtensions

If present, the cvResponseExtensions item contains a sequence of extensions that extend the response. This specification does not define any extensions. The facility is provided to allow future specifications to extend SCVP. The syntax for Extensions is imported from [PKIX-1]. The cvResponseExtensions item, when present, contains a sequence of Extension items, each of which contains an extnID item, a critical item, and an extnValue item.

The extnID item is an identifier for the extension. It contains the object identifier (OID) that names the extension.

The critical item is a BOOLEAN. Each extension is designated as either critical (with a value of TRUE) or non-critical (with a value of FALSE). An SCVP client MUST reject the response if it encounters a critical extension it does not recognize; however, a non-critical extension MAY be ignored if it is not recognized.

The extnValue item contains an OCTET STRING. Within the OCTET STRING is the extension value. An ASN.1 type is specified for each extension, identified by the associated extnID object identifier.

4.13. requestorText

The requestorText item contains a text field supplied by the client.

If the client includes a requestorText value in the request and the server is generating a specific non-cached response to the request, then the server MUST return the same value in the response.

If the server is using a cached response to the request, then it MUST omit the requestorText item.

The requestNonce item uses the UTF8 string type.

Conforming client implementations that support the requestorText item in requests (see Section 3.10) MUST be able to process a response that includes this item. Conforming servers MUST support requestorText in responses.

4.14. SCVP Response Validation

There are two mechanisms for validation of SCVP responses, one based on the client's knowledge of a specific SCVP server key and the other based on validation of the certificate corresponding to the private key used to protect the SCVP response.

4.14.1. Simple Key Validation

The simple key validation method is where the SCVP client has a local policy of one or more SCVP server keys that directly identify the set of valid SCVP servers. Mechanisms for storage of server keys or identifiers are a local matter. For example, a client could store cryptographic hashes of public keys used to verify SignedData responses. Alternatively, a client could store shared symmetric keys used to verify MACs in AuthenticatedData responses.

Simple key validation MUST be used by SCVP clients that cannot validate PKIX-1 certificates and are therefore making delegated path validation requests to the SCVP server [RQMTS]. It is a matter of local policy with these clients whether to use SignedData or AuthenticatedData. Simple key validation MAY be used by other SCVP clients for other reasons.

4.14.2. SCVP Server Certificate Validation

It is a matter of local policy what validation policy the client uses when validating responses. When validating protected SCVP responses, SCVP clients SHOULD use the validation algorithm defined in Section 6 of [PKIX-1]. SCVP clients may impose additional limitations on the algorithm, such as limiting the number of certificates in the path or establishing initial name constraints, as specified in Section 6.2 of [PKIX-1].

If the certificate used to sign the validation policy responses and SignedData validation responses contains the key usage extension ([PKIX-1], Section 4.2.1.3), it MUST have either the digital signature bit set, the non-repudiation bit set, or both bits set.

If the certificate for AuthenticatedData validation responses contains the key usage extension, it MUST have the key agreement bit set.

If the certificate used on a validation policy response or a validation response contains the extended key usage extension ([PKIX-1], Section 4.2.1.13), it MUST contain either the anyExtendedKeyUsage OID or the following OID:

```
id-kp-scvpServer             OBJECT IDENTIFIER ::= { id-kp 15 }
```

5. Server Policy Request

An SCVP client uses the ValPolRequest item to request information about an SCVP server's policies and configuration information, including the list of validation policies supported by the SCVP server. When a ValPolRequest is encapsulated in a MIME body part, it MUST be carried in an application/scvp-vp-request MIME body part.

The request consists of a ValPolRequest encapsulated in a ContentInfo. The client does not sign the request.

```
ContentInfo {
  contentType        id-ct-scvp-valPolRequest,
                              -- (1.2.840.113549.1.9.16.1.12)
  content            ValPolRequest }
```

The ValPolRequest type has the following syntax:

```
ValPolRequest ::= SEQUENCE {
  vpRequestVersion           INTEGER DEFAULT 1,
  requestNonce               OCTET STRING }
```

Conforming SCVP server implementations MUST recognize and process the server policy request. Conforming clients SHOULD support the server policy request.

5.1. vpRequestVersion

The syntax and semantics of vpRequestVersion are the same as cvRequestVersion as described in Section 3.1.

5.2. requestNonce

The requestNonce item contains a request identifier generated by the SCVP client. If the server returns a specific response, it MUST include the requestNonce from the request in the response, but the server MAY return a cached response, which MUST NOT include a requestNonce.

6. Validation Policy Response

In response to a ValPolRequest, the SCVP server provides a ValPolResponse. The ValPolResponse may not be unique to any ValPolRequest, so may be reused by the server in response to multiple ValPolRequests. The ValPolResponse also has an indication of how frequently the ValPolResponse may be reissued. The server MUST sign the response using its digital signature certificate. When a ValPolResponse is encapsulated in a MIME body part, it MUST be carried in an application/scvp-vp-response MIME body part.

The response consists of a ValPolResponse encapsulated in a SignedData, which is in turn encapsulated in a ContentInfo. That is, the EncapsulatedContentInfo field of SignedData consists of an eContentType field with a value of id-ct-scvp-valPolResponse (1.2.840.113549.1.9.16.1.13) and an eContent field that contains a DER-encoded ValPolResponse. The SCVP server MUST include its own certificate in the certificates field within SignedData, and the signerInfos field of SignedData MUST include exactly one SignerInfo. The SignedData MUST NOT include the unsignedAttrs field.

The ValPolResponse type has the following syntax:

```
ValPolResponse ::= SEQUENCE {
  vpResponseVersion               INTEGER,
  maxCVRequestVersion             INTEGER,
  maxVPRequestVersion             INTEGER,
  serverConfigurationID           INTEGER,
  thisUpdate                      GeneralizedTime,
  nextUpdate                      GeneralizedTime OPTIONAL,
  supportedChecks                 CertChecks,
  supportedWantBacks              WantBack,
  validationPolicies              SEQUENCE OF OBJECT IDENTIFIER,
  validationAlgs                  SEQUENCE OF OBJECT IDENTIFIER,
  authPolicies                    SEQUENCE OF AuthPolicy,
  responseTypes                   ResponseTypes,
  defaultPolicyValues             RespValidationPolicy,
  revocationInfoTypes             RevocationInfoTypes,
  signatureGeneration             SEQUENCE OF AlgorithmIdentifier,
  signatureVerification           SEQUENCE OF AlgorithmIdentifier,
  hashAlgorithms                  SEQUENCE SIZE (1..MAX) OF
                                     OBJECT IDENTIFIER,
  serverPublicKeys                SEQUENCE OF KeyAgreePublicKey
                                     OPTIONAL,
  clockSkew                       INTEGER DEFAULT 10,
  requestNonce                    OCTET STRING OPTIONAL }
```

```
ResponseTypes  ::= ENUMERATED {
  cached-only               (0),
  non-cached-only           (1),
  cached-and-non-cached     (2) }

RevocationInfoTypes ::= BIT STRING {
  fullCRLs                  (0),
  deltaCRLs                 (1),
  indirectCRLs              (2),
  oCSPResponses             (3) }
```

SCVP clients that support validation policy requests MUST support validation policy responses. SCVP servers MUST support validation policy responses.

SCVP servers MUST support cached policy responses and MAY support specific responses to policy requests.

6.1. vpResponseVersion

The syntax and semantics of the vpResponseVersion item are the same as cvRequestVersion as described in Section 3.1. The vpResponseVersion used MUST be the same as the vpRequestVersion unless the client has used a value greater than the values the server supports. If the client submits a vpRequestVersion greater than the version supported by the server, the server MUST return a vpResponseVersion using the highest version number the server supports as the version number.

6.2. maxCVRequestVersion

The maxCVRequestVersion item defines the maximum version number for CV requests that the server supports.

6.3. maxVPRequestVersion

The maxVPRequestVersion item defines the maximum version number for VP requests that the server supports.

6.4. serverConfigurationID

The serverConfigurationID item is an integer that uniquely represents the version of the server configuration as represented by the validationPolicies, validationAlgs, authPolicies, defaultPolicyValues, and clockSkew. If any of these values change, the server MUST create a new ValPolResponse with a new serverConfigurationID. If the configuration has not changed, then the server may reuse serverConfigurationID across multiple

ValPolResponse messages. However, if the server reverts to an earlier configuration, the server MUST NOT revert the configuration ID as well, but MUST select another unique value.

6.5. thisUpdate

This item indicates the signing date and time of this policy response.

GeneralizedTime values MUST be expressed in Greenwich Mean Time (Zulu) and interpreted as defined in Section 3.2.7.

6.6. nextUpdate and requestNonce

These items are used to indicate whether policy responses are specific to policy requests. Where policy responses are cached, these items indicate when the information will be updated. The optional nextUpdate item indicates the time by which the next policy response will be published. The optional requestNonce item links the response to a specific request by returning the nonce provided in the request.

If the nextUpdate item is omitted, it indicates a non-cached response generated in response to a specific request (i.e., the ValPolResponse is bound to a specific request). If this item is omitted, the requestNonce item MUST be present and MUST include the requestNonce value from the request.

If the nextUpdate item is present, it indicates a cached response that is not bound to a specific request. An SCVP server MUST periodically generate a new response as defined by the next update time, but MAY use the same ValPolResponse to respond to multiple requests. The requestNonce is omitted if the nextUpdate item is present.

It is a matter of local server policy to return a cached or non-cached specific response.

GeneralizedTime values in nextUpdate MUST be expressed in Greenwich Mean Time (Zulu) as specified in Section 3.2.7.

6.7. supportedChecks

The supportedChecks item contains a sequence of object identifiers representing the checks supported by the server.

6.8. supportedWantBacks

The supportedWantBacks item contains a sequence of object identifiers representing the wantBacks supported by the server.

6.9. validationPolicies

The validationPolicies item contains a sequence of object identifiers representing the validation policies supported by the server. It is a matter of local policy if the server wishes to process requests using the default validation policy, and if it does not, then it MUST NOT include the id-svp-defaultValPolicy in this list.

6.10. validationAlgs

The validationAlgs item contains a sequence of OIDs. Each OID identifies a validation algorithm supported by the server.

6.11. authPolicies

The authPolicies item contains a sequence of policy references for authenticating to the SCVP server.

The reference to the authentication policy is an OID that the client and server have agreed represents an authentication policy. The list of policies is intended to document to the client if authentication is required for some requests and if so how.

```
AuthPolicy ::=  OBJECT IDENTIFIER
```

6.12. responseTypes

The responseTypes item allows the server to publish the range of response types it supports. Cached only means the server will only return cached responses to requests. Non-cached only means the server will return a specific response to the request, i.e., containing the requestor’s nonce. Both means that the server supports both cached and non-cached response types and will return either a cached or non- cached response, depending on the request.

6.13. revocationInfoTypes

The revocationInfoTypes item allows the server to indicate the sources of revocation information that it is capable of processing. For each bit in the RevocationInfoTypes BIT STRING, the server MUST set the bit to one if it is capable of processing the corresponding revocation information type and to zero if it cannot.

6.14. defaultPolicyValues

This is the default validation policy used by the server. It contains a RespValidationPolicy, which is defined in Section 4.5. All OPTIONAL items in the validationPolicy item MUST be populated. A server will use these default values when the request references the default validation policy and the client does not override the default values by supplying other values in the request.

This allows the client to optimize the request by omitting parameters that match the server default values.

6.15. signatureGeneration

This sequence specifies the set of digital signature algorithms supported by an SCVP server for signing CVResponse messages. Each digital signature algorithm is specified as an AlgorithmIdentifier, using the encoding rules associated with the signatureAlgorithm field in a public key certificate [PKIX-1]. Supported algorithms are defined in [PKIX-ALG] and [PKIX-ALG2], but other signature algorithms may also be supported.

By including an algorithm (e.g., RSA with SHA-1) in this list, the server states that it has a private key and corresponding certified public key for that asymmetric algorithm, and supports the specified hash algorithm. The list is ordered; the first digital signature algorithm is the server's default algorithm. The default algorithm will be used by the server to protect signed messages unless the client specifies another algorithm.

For servers that do not have an on-line private key, and cannot sign CVResponse messages, the signatureGeneration item is encoded as an empty sequence.

6.16. signatureVerification

This sequence specifies the set of digital signature algorithms that can be verified by this SCVP server. Each digital signature algorithm is specified as an AlgorithmIdentifier, using the encoding rules associated with the signatureAlgorithm field in a public key certificate [PKIX-1]. Supported algorithms are defined in [PKIX-ALG] and [PKIX-ALG2], but other signature algorithms may also be supported.

For servers that do not verify signatures on CVRequest messages, the signatureVerification item is encoded as an empty sequence.

6.17. hashAlgorithms

This sequence specifies the set of hash algorithms that the server can use to hash certificates and requests. The list is ordered; the first hash algorithm is the server's default algorithm. The default algorithm will be used by the server to compute hashes included in responses unless the client specifies another algorithm. Each hash algorithm is specified as an object identifier. [PKIX-ALG2] specifies object identifiers for SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. Other hash algorithms may also be supported.

6.18. serverPublicKeys

The serverPublicKeys item is a sequence of one or more key agreement public keys and associated parameters. It is used by clients making AuthenticatedData requests to the server. Each item in the serverPublicKeys sequence is of the KeyAgreePublicKey type:

```
KeyAgreePublicKey ::= SEQUENCE {
  algorithm            AlgorithmIdentifier,
  publicKey            BIT STRING,
  macAlgorithm         AlgorithmIdentifier,
  kDF                  AlgorithmIdentifier OPTIONAL }
```

The KeyAgreePublicKey includes the algorithm identifier and the server's public key. SCVP servers that support the key agreement mode of AuthenticatedData for SCVP requests MUST support serverPublicKeys and the Diffie-Hellman key agreement algorithm as specified in [PKIX-ALG]. SCVP servers that support serverPublicKeys MUST support the 1024-bit Modular Exponential (MODP) group key (group 2) as defined in [IKE]. SCVP servers that support serverPublicKeys MAY support other Diffie-Hellman groups [IKE-GROUPS], as well as other key agreement algorithms.

The macAlgorithm item specifies the symmetric algorithm the server expects the client to use with the result of the key agreement algorithm. A key derivation function (KDF), which derives symmetric key material from the key agreement result, may be implied by the macAlgorithm. Alternatively, the KDF may be explicitly specified using the optional kDF item.

6.19. clockSkew

The clockSkew item is the number of minutes the server will allow for clock skew. The default value is 10 minutes.

7. SCVP Server Relay

In some network environments, especially ones that include firewalls, an SCVP server might not be able to obtain all of the information that it needs to process a request. However, the server might be configured to use the services of one or more other SCVP servers to fulfill all requests. In such cases, the SCVP client is unaware that the initial SCVP server is using the services of other SCVP servers. The initial SCVP server acts as a client to another SCVP server. Unlike the original client, the SCVP server is expected to have moderate computing and memory resources. This section describes SCVP server-to-SCVP server exchanges. This section does not impose any requirements on SCVP clients that are not also SCVP servers. Further, this section does not impose any requirements on SCVP servers that do not relay requests to other SCVP servers.

When one SCVP server relays a request to another server, in an incorrectly configured system of servers, it is possible that the same request will be relayed back again. Any SCVP server that relays requests MUST implement the conventions described in this section to detect and break loops.

When an SCVP server relays a request, the request MUST include the requestorRef item. If the request to be relayed already contains a requestorRef item, then the server-generated request MUST contain a requestorRef item constructed from this value and an additional GeneralName that contains an identifier of the SCVP server. If the request to be relayed does not contain a requestorRef item, then the server-generated request MUST contain a requestorRef item that includes a GeneralName that contains an identifier of the SCVP server.

To prevent false loop detection, servers should use identifiers that are unique within their network of cooperating SCVP servers. SCVP servers that support relay SHOULD populate this item with the DNS name of the server or the distinguished name in the server’s certificate. SCVP servers MAY choose other procedures for generating identifiers that are unique within their community.

When an SCVP server receives a request that contains a requestorRef item, the server MUST check the sequence of names in the requestorRef item for its own identifier. If the server discovers its own identifier in the requestorRef item, it MUST respond with an error, setting the statusCode in the responseStatus item to 40.

When an SCVP server generates a non-cached response to a relayed request, the server MUST include the requestorRef item from the request in the response.

8. SCVP ASN.1 Module

This section defines the syntax for SCVP request-response pairs. The semantics for the messages are defined in Sections 3, 4, 5, and 6. The SCVP ASN.1 module follows.

```
SCVP

  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) 21 }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

IMPORTS

AlgorithmIdentifier, Attribute, Certificate, Extensions,
-- Import UTF8String if required by compiler
-- UTF8String, -- CertificateList, CertificateSerialNumber
  FROM PKIX1Explicit88 -- RFC 3280
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) 18 }

GeneralNames, GeneralName, KeyUsage, KeyPurposeId
  FROM PKIX1Implicit88 -- RFC 3280
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) 19 }

AttributeCertificate
  FROM PKIXAttributeCertificate -- RFC 3281
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) 12 }

OCSPResponse
  FROM OCSP -- RFC 2560
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0) 14 }

ContentInfo
  FROM CryptographicMessageSyntax2004 -- RFC 3852
  { iso(1) member-body(2) us(840) rsadsi(113549)
    pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) } ;


-- SCVP Certificate Validation Request

id-ct OBJECT IDENTIFIER ::= { iso(1) member-body(2)
          us(840) rsadsi(113549) pkcs(1) pkcs9(9)
          id-smime(16) 1 }
```

```
id-ct-scvp-certValRequest OBJECT IDENTIFIER ::= { id-ct 10 }

CVRequest ::= SEQUENCE {
  cvRequestVersion           INTEGER DEFAULT 1,
  query                      Query,
  requestorRef           [0] GeneralNames OPTIONAL,
  requestNonce           [1] OCTET STRING OPTIONAL,
  requestorName          [2] GeneralName OPTIONAL,
  responderName          [3] GeneralName OPTIONAL,
  requestExtensions      [4] Extensions OPTIONAL,
  signatureAlg           [5] AlgorithmIdentifier OPTIONAL,
  hashAlg                [6] OBJECT IDENTIFIER OPTIONAL,
  requestorText          [7] UTF8String (SIZE (1..256)) OPTIONAL }

Query ::= SEQUENCE {
  queriedCerts             CertReferences,
  checks                   CertChecks,
  wantBack             [1] WantBack OPTIONAL,
  validationPolicy         ValidationPolicy,
  responseFlags            ResponseFlags OPTIONAL,
  serverContextInfo    [2] OCTET STRING OPTIONAL,
  validationTime       [3] GeneralizedTime OPTIONAL,
  intermediateCerts    [4] CertBundle OPTIONAL,
  revInfos             [5] RevocationInfos OPTIONAL,
  producedAt           [6] GeneralizedTime OPTIONAL,
  queryExtensions      [7] Extensions OPTIONAL }

CertReferences ::= CHOICE {
  pkcRefs       [0] SEQUENCE SIZE (1..MAX) OF PKCReference,
  acRefs        [1] SEQUENCE SIZE (1..MAX) OF ACReference }

CertReference::= CHOICE {
  pkc               PKCReference,
  ac                ACReference }

PKCReference ::= CHOICE {
  cert          [0] Certificate,
  pkcRef        [1] SCVPCertID }

ACReference ::= CHOICE {
  attrCert      [2] AttributeCertificate,
  acRef         [3] SCVPCertID }

SCVPCertID ::= SEQUENCE {
    certHash        OCTET STRING,
    issuerSerial    SCVPIssuerSerial,
    hashAlgorithm   AlgorithmIdentifier DEFAULT { algorithm sha-1 } }
```

```
   SCVPIssuerSerial ::= SEQUENCE {
        issuer         GeneralNames,
        serialNumber   CertificateSerialNumber
   }

   ValidationPolicy ::= SEQUENCE {
     validationPolRef           ValidationPolRef,
     validationAlg          [0] ValidationAlg OPTIONAL,
     userPolicySet          [1] SEQUENCE SIZE (1..MAX) OF OBJECT
                                  IDENTIFIER OPTIONAL,
     inhibitPolicyMapping   [2] BOOLEAN OPTIONAL,
     requireExplicitPolicy  [3] BOOLEAN OPTIONAL,
     inhibitAnyPolicy       [4] BOOLEAN OPTIONAL,
     trustAnchors           [5] TrustAnchors OPTIONAL,
     keyUsages              [6] SEQUENCE OF KeyUsage OPTIONAL,
     extendedKeyUsages      [7] SEQUENCE OF KeyPurposeId OPTIONAL,
     specifiedKeyUsages     [8] SEQUENCE OF KeyPurposeId OPTIONAL }


   CertChecks ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER

   WantBack ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER

   ValidationPolRef ::= SEQUENCE {
       valPolId             OBJECT IDENTIFIER,
       valPolParams         ANY DEFINED BY valPolId OPTIONAL }

   ValidationAlg ::= SEQUENCE {
     valAlgId               OBJECT IDENTIFIER,
     parameters             ANY DEFINED BY valAlgId OPTIONAL }

   NameValidationAlgParms ::= SEQUENCE {
     nameCompAlgId          OBJECT IDENTIFIER,
     validationNames        GeneralNames }

   TrustAnchors ::= SEQUENCE SIZE (1..MAX) OF PKCReference

   KeyAgreePublicKey ::= SEQUENCE {
     algorithm           AlgorithmIdentifier,
     publicKey           BIT STRING,
     macAlgorithm        AlgorithmIdentifier,
     kDF                 AlgorithmIdentifier OPTIONAL }

   ResponseFlags ::= SEQUENCE {
     fullRequestInResponse      [0] BOOLEAN DEFAULT FALSE,
     responseValidationPolByRef [1] BOOLEAN DEFAULT TRUE,
     protectResponse            [2] BOOLEAN DEFAULT TRUE,
     cachedResponse             [3] BOOLEAN DEFAULT TRUE }
```

```
   CertBundle ::= SEQUENCE SIZE (1..MAX) OF Certificate

   RevocationInfos ::= SEQUENCE SIZE (1..MAX) OF RevocationInfo

   RevocationInfo ::= CHOICE {
     crl                   [0] CertificateList,
     delta-crl             [1] CertificateList,
     ocsp                  [2] OCSPResponse,
     other                 [3] OtherRevInfo }

   OtherRevInfo ::= SEQUENCE {
     riType                    OBJECT IDENTIFIER,
     riValue                   ANY DEFINED BY riType }

   -- SCVP Certificate Validation Response

   id-ct-scvp-certValResponse OBJECT IDENTIFIER ::= { id-ct 11 }

   CVResponse ::= SEQUENCE {
     cvResponseVersion         INTEGER,
     serverConfigurationID     INTEGER,
     producedAt                GeneralizedTime,
     responseStatus            ResponseStatus,
     respValidationPolicy  [0] RespValidationPolicy OPTIONAL,
     requestRef            [1] RequestReference OPTIONAL,
     requestorRef          [2] GeneralNames OPTIONAL,
     requestorName         [3] GeneralNames OPTIONAL,
     replyObjects          [4] ReplyObjects OPTIONAL,
     respNonce             [5] OCTET STRING OPTIONAL,
     serverContextInfo     [6] OCTET STRING OPTIONAL,
     cvResponseExtensions  [7] Extensions OPTIONAL,
     requestorText         [8] UTF8String (SIZE (1..256)) OPTIONAL }

   ResponseStatus ::= SEQUENCE {
       statusCode              CVStatusCode DEFAULT  okay,
       errorMessage            UTF8String OPTIONAL }

   CVStatusCode ::= ENUMERATED {
       okay                                (0),
       skipUnrecognizedItems               (1),
       tooBusy                            (10),
       invalidRequest                     (11),
       internalError                      (12),
       badStructure                       (20),
       unsupportedVersion                 (21),
       abortUnrecognizedItems             (22),
       unrecognizedSigKey                 (23),
       badSignatureOrMAC                  (24),
```

```
      unableToDecode                    (25),
      notAuthorized                     (26),
      unsupportedChecks                 (27),
      unsupportedWantBacks              (28),
      unsupportedSignatureOrMAC         (29),
      invalidSignatureOrMAC             (30),
      protectedResponseUnsupported      (31),
      unrecognizedResponderName         (32),
      relayingLoop                      (40),
      unrecognizedValPol                (50),
      unrecognizedValAlg                (51),
      fullRequestInResponseUnsupported  (52),
      fullPolResponseUnsupported        (53),
      inhibitPolicyMappingUnsupported   (54),
      requireExplicitPolicyUnsupported  (55),
      inhibitAnyPolicyUnsupported       (56),
      validationTimeUnsupported         (57),
      unrecognizedCritQueryExt          (63),
      unrecognizedCritRequestExt        (64) }

   RespValidationPolicy ::= ValidationPolicy

   RequestReference ::= CHOICE {
     requestHash   [0] HashValue, -- hash of CVRequest
     fullRequest   [1] CVRequest }

   HashValue ::= SEQUENCE {
     algorithm         AlgorithmIdentifier DEFAULT { algorithm sha-1 },
     value             OCTET STRING }

   sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
             oiw(14) secsig(3) algorithm(2) 26 }

   ReplyObjects ::= SEQUENCE SIZE (1..MAX) OF CertReply

   CertReply ::= SEQUENCE {
     cert                      CertReference,
     replyStatus               ReplyStatus DEFAULT success,
     replyValTime              GeneralizedTime,
     replyChecks               ReplyChecks,
     replyWantBacks            ReplyWantBacks,
     validationErrors       [0] SEQUENCE SIZE (1..MAX) OF
                                  OBJECT IDENTIFIER OPTIONAL,
     nextUpdate             [1] GeneralizedTime OPTIONAL,
     certReplyExtensions    [2] Extensions OPTIONAL }
```

```
   ReplyStatus ::= ENUMERATED {
     success                    (0),
     malformedPKC               (1),
     malformedAC                (2),
     unavailableValidationTime  (3),
     referenceCertHashFail      (4),
     certPathConstructFail      (5),
     certPathNotValid           (6),
     certPathNotValidNow        (7),
     wantBackUnsatisfied        (8) }

   ReplyChecks ::= SEQUENCE OF ReplyCheck

   ReplyCheck ::= SEQUENCE {
     check                      OBJECT IDENTIFIER,
     status                     INTEGER DEFAULT 0 }

   ReplyWantBacks ::= SEQUENCE OF ReplyWantBack

   ReplyWantBack::= SEQUENCE {
     wb                         OBJECT IDENTIFIER,
     value                      OCTET STRING }

   CertBundles ::= SEQUENCE SIZE (1..MAX) OF CertBundle

   RevInfoWantBack ::= SEQUENCE {
     revocationInfo             RevocationInfos,
     extraCerts                 CertBundle OPTIONAL }

   SCVPResponses ::= SEQUENCE OF ContentInfo

   -- SCVP Validation Policies Request

   id-ct-scvp-valPolRequest     OBJECT IDENTIFIER ::= { id-ct 12 }

   ValPolRequest ::= SEQUENCE {
     vpRequestVersion           INTEGER DEFAULT 1,
     requestNonce               OCTET STRING }

   -- SCVP Validation Policies Response

   id-ct-scvp-valPolResponse OBJECT IDENTIFIER ::= { id-ct 13 }

   ValPolResponse ::= SEQUENCE {
     vpResponseVersion               INTEGER,
     maxCVRequestVersion             INTEGER,
     maxVPRequestVersion             INTEGER,
     serverConfigurationID           INTEGER,
```

```
    thisUpdate                      GeneralizedTime,
    nextUpdate                      GeneralizedTime OPTIONAL,
    supportedChecks                 CertChecks,
    supportedWantBacks              WantBack,
    validationPolicies              SEQUENCE OF OBJECT IDENTIFIER,
    validationAlgs                  SEQUENCE OF OBJECT IDENTIFIER,
    authPolicies                    SEQUENCE OF AuthPolicy,
    responseTypes                   ResponseTypes,
    defaultPolicyValues             RespValidationPolicy,
    revocationInfoTypes             RevocationInfoTypes,
    signatureGeneration             SEQUENCE OF AlgorithmIdentifier,
    signatureVerification           SEQUENCE OF AlgorithmIdentifier,
    hashAlgorithms                  SEQUENCE SIZE (1..MAX) OF
                                      OBJECT IDENTIFIER,
    serverPublicKeys                SEQUENCE OF KeyAgreePublicKey
                                      OPTIONAL,
    clockSkew                       INTEGER DEFAULT 10,
    requestNonce                    OCTET STRING OPTIONAL }

ResponseTypes  ::= ENUMERATED {
    cached-only               (0),
    non-cached-only           (1),
    cached-and-non-cached     (2) }

RevocationInfoTypes ::= BIT STRING {
    fullCRLs                  (0),
    deltaCRLs                 (1),
    indirectCRLs              (2),
    oCSPResponses             (3) }

AuthPolicy ::= OBJECT IDENTIFIER

-- SCVP Check Identifiers

id-stc OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
          dod(6) internet(1) security(5) mechanisms(5) pkix(7) 17 }

id-stc-build-pkc-path        OBJECT IDENTIFIER ::= { id-stc 1 }
id-stc-build-valid-pkc-path  OBJECT IDENTIFIER ::= { id-stc 2 }
id-stc-build-status-checked-pkc-path
                             OBJECT IDENTIFIER ::= { id-stc 3 }
id-stc-build-aa-path         OBJECT IDENTIFIER ::= { id-stc 4 }
id-stc-build-valid-aa-path   OBJECT IDENTIFIER ::= { id-stc 5 }
id-stc-build-status-checked-aa-path
                             OBJECT IDENTIFIER ::= { id-stc 6 }
id-stc-status-check-ac-and-build-status-checked-aa-path
                             OBJECT IDENTIFIER ::= { id-stc 7 }
```

```
   -- SCVP WantBack Identifiers

   id-swb OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
             dod(6) internet(1) security(5) mechanisms(5) pkix(7) 18 }

   id-swb-pkc-best-cert-path       OBJECT IDENTIFIER ::= { id-swb 1 }
   id-swb-pkc-revocation-info      OBJECT IDENTIFIER ::= { id-swb 2 }
   id-swb-pkc-public-key-info      OBJECT IDENTIFIER ::= { id-swb 4 }
   id-swb-aa-cert-path             OBJECT IDENTIFIER ::= { id-swb 5 }
   id-swb-aa-revocation-info       OBJECT IDENTIFIER ::= { id-swb 6 }
   id-swb-ac-revocation-info       OBJECT IDENTIFIER ::= { id-swb 7 }
   id-swb-relayed-responses        OBJECT IDENTIFIER ::= { id-swb 9 }
   id-swb-pkc-cert                 OBJECT IDENTIFIER ::= { id-swb 10}
   id-swb-ac-cert                  OBJECT IDENTIFIER ::= { id-swb 11}
   id-swb-pkc-all-cert-paths       OBJECT IDENTIFIER ::= { id-swb 12}
   id-swb-pkc-ee-revocation-info   OBJECT IDENTIFIER ::= { id-swb 13}
   id-swb-pkc-CAs-revocation-info  OBJECT IDENTIFIER ::= { id-swb 14}

   -- SCVP Validation Policy and Algorithm Identifiers

   id-svp OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
             dod(6) internet(1) security(5) mechanisms(5) pkix(7) 19 }

   id-svp-defaultValPolicy OBJECT IDENTIFIER ::= { id-svp 1 }

   -- SCVP Basic Validation Algorithm Identifier

   id-svp-basicValAlg OBJECT IDENTIFIER ::= { id-svp 3 }

   -- SCVP Basic Validation Algorithm Errors

   id-bvae OBJECT IDENTIFIER ::= id-svp-basicValAlg

   id-bvae-expired              OBJECT IDENTIFIER ::= { id-bvae 1 }
   id-bvae-not-yet-valid        OBJECT IDENTIFIER ::= { id-bvae 2 }
   id-bvae-wrongTrustAnchor     OBJECT IDENTIFIER ::= { id-bvae 3 }
   id-bvae-noValidCertPath      OBJECT IDENTIFIER ::= { id-bvae 4 }
   id-bvae-revoked              OBJECT IDENTIFIER ::= { id-bvae 5 }
   id-bvae-invalidKeyPurpose    OBJECT IDENTIFIER ::= { id-bvae 9 }
   id-bvae-invalidKeyUsage      OBJECT IDENTIFIER ::= { id-bvae 10 }
   id-bvae-invalidCertPolicy    OBJECT IDENTIFIER ::= { id-bvae 11 }

   -- SCVP Name Validation Algorithm Identifier

   id-svp-nameValAlg OBJECT IDENTIFIER ::= { id-svp 2 }
```

```
   -- SCVP Name Validation Algorithm DN comparison algorithm

   id-nva-dnCompAlg   OBJECT IDENTIFIER ::= { id-svp 4 }

   -- SCVP Name Validation Algorithm Errors

   id-nvae OBJECT IDENTIFIER ::= id-svp-nameValAlg

   id-nvae-name-mismatch          OBJECT IDENTIFIER ::= { id-nvae 1 }
   id-nvae-no-name                OBJECT IDENTIFIER ::= { id-nvae 2 }
   id-nvae-unknown-alg            OBJECT IDENTIFIER ::= { id-nvae 3 }
   id-nvae-bad-name               OBJECT IDENTIFIER ::= { id-nvae 4 }
   id-nvae-bad-name-type          OBJECT IDENTIFIER ::= { id-nvae 5 }
   id-nvae-mixed-names            OBJECT IDENTIFIER ::= { id-nvae 6 }

   -- SCVP Extended Key Usage Key Purpose Identifiers

   id-kp OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
             dod(6) internet(1) security(5) mechanisms(5) pkix(7) 3 }

   id-kp-scvpServer               OBJECT IDENTIFIER ::= { id-kp 15 }

   id-kp-scvpClient               OBJECT IDENTIFIER ::= { id-kp 16 }

   END
```

9. Security Considerations

For security considerations specific to the Cryptographic Message Syntax message formats, see [CMS]. For security considerations specific to the process of PKI certification path validation, see [PKIX-1].

A client that trusts a server's response for validation of a certificate inherently trusts that server as much as it would trust its own validation software. This means that if an attacker compromises a trusted SCVP server, the attacker can change the validation processing for every client that relies on that server. Thus, an SCVP server must be protected at least as well as the trust anchors that the SCVP server trusts.

Clients MUST verify that the response matches their original request. Clients need to ensure that the server has performed the appropriate checks for the correct certificates under the requested validation policy for the specified validation time, and that the response includes the requested wantBacks and meets the client's freshness requirements.

When the SCVP response is used to determine the validity of a certificate, the client MUST validate the digital signature or MAC on the response to ensure that the expected SCVP server generated it. If the client does not check the digital signature or MAC on the response, a man-in-the-middle attack could fool the client into believing modified responses from the server or responses to questions the client did not ask.

If the client does not include a requestNonce item, or if the client does not check that the requestNonce in the response matches the value in the request, an attacker can replay previous responses from the SCVP server.

If the server does not require some sort of authorization (such as signed requests), an attacker can get the server to respond to arbitrary requests. Such responses may give the attacker information about weaknesses in the server or about the timeliness of the server's checking. This information may be valuable for a future attack.

If the server uses the serverContextInfo item to indicate some server state associated with a requestor, implementers must take appropriate measures against denial-of-service attacks where an attacker sends in a lot of requests at one time to force the server to keep a lot of state information.

SCVP does not include any confidentiality mechanisms. If confidentiality is needed, it can be achieved with a lower-layer security protocol such as TLS [TLS].

If an SCVP client is not operating on a network with good physical protection, it must ensure that there is integrity over the SCVP request-response pair. The client can ensure integrity by using a protected transport such as TLS. It can ensure integrity by using MACs or digital signatures to individually protect the request and response messages.

If an SCVP client populates the userPolicySet in a request with a value other than anyPolicy, but does not set the requireExplicitPolicy flag, the server may return an affirmative answer for paths that do not satisfy any of the specified policies. In general, when a client populates the userPolicySet in a request with a value other than anyPolicy, the requireExplicitPolicy flag should also be set. This guarantees that all valid paths satisfy at least one of the requested policies.

In SCVP, historical validation of a certificate returns the known status of the certificate at the time specified in validationTime. This may be used to demonstrate due diligence, but does not necessarily provide the most complete information. A certificate may have been revoked after the time specified in validationTime, but the revocation notice may specify an invalidity date that precedes the validationTime. The SCVP server would provide an affirmative response even though the most current information available indicates the certificate should not be trusted at that time. SCVP clients may wish to specify a validationTime later than the actual time of interest to mitigate this risk.

10. IANA Considerations

The details of SCVP requests and responses are communicated using object identifiers (OIDs). The objects are defined in an arc delegated by IANA to the PKIX Working Group. This document also includes four MIME type registrations in Appendix A. No further action by IANA is necessary for this document or any anticipated updates.

11. References

11.1. Normative References

[STDWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[OCSP] 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.

[PKIX-1] 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.

[PKIX-AC] Farrell, S. and R. Housley, "An Internet Attribute Certificate Profile for Authorization", RFC 3281, April 2002.

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

[PKIX-ALG2] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[ESS] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[SMIME-CERT] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, July 2004.

[IKE] Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[HTTP] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

11.2. Informative References

[IKE-GROUPS] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RQMTS] Pinkas, D. and R. Housley, "Delegated Path Validation and Delegated Path Discovery Protocol Requirements", RFC 3379, September 2002.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

12. Acknowledgments

The lively debate in the PKIX Working Group has made a significant impact on this protocol. Special thanks to the following for their contributions to this document and diligence in greatly improving it.

Paul Hoffman
Phillip Hallam-Baker
Mike Myers
Frank Balluffi
Ameya Talwalkar
John Thielens
Peter Sylvester
Yuriy Dzambasow
Sean P. Turner
Wen-Cheng Wang
Francis Dupont
Dave Engberg
Faisal Maqsood

Thanks also to working group chair Steve Kent for his support and help.

Appendix A. MIME Media Type Registrations

Four MIME media type registrations are provided in this appendix.

A.1. application/scvp-cv-request

To: ietf-types@iana.org
Subject: Registration of MIME media type application/scvp-cv-request

MIME media type name: application

MIME subtype name: scvp-cv-request

Required parameters: None

Optional parameters: None

Encoding considerations: Binary

Security considerations: Carries a request for information. This request may optionally be cryptographically protected.

Interoperability considerations: None

Published specification: RFC 5055

Applications that use this media type: SCVP clients sending certificate validation requests

Additional information:

- Magic number(s): None
- File extension(s): .SCQ
- Macintosh File Type Code(s): None

Person & email address to contact for further information:
Ambarish Malpani <ambarish@yahoo.com>

Intended usage: COMMON

Restrictions on usage: This media type can be used with any protocol that can transport digitally signed objects.

Author: Ambarish Malpani <ambarish@yahoo.com>

Change controller: IESG

A.2. application/scvp-cv-response

To: ietf-types@iana.org
Subject: Registration of MIME media type application/scvp-cv-response

MIME media type name: application

MIME subtype name: scvp-cv-response

Required parameters: None

Optional parameters: None

Encoding considerations: Binary

Security considerations: The client may require that this response be cryptographically protected, or may choose to use a secure transport mechanism. DPD responses may be unprotected, but the client validates the information provided in the request.

Interoperability considerations: None

Published specification: RFC 5055

Applications that use this media type: SCVP servers responding to certificate validation requests

Additional information:

- Magic number(s): None
- File extension(s): .SCS
- Macintosh File Type Code(s): none

Person & email address to contact for further information:
Ambarish Malpani <ambarish@yahoo.com>

Intended usage: COMMON
Restrictions on usage: This media type can be used with any protocol that can transport digitally signed objects.

Author: Ambarish Malpani <ambarish@yahoo.com>

Change controller: IESG

A.3. application/scvp-vp-request

To: ietf-types@iana.org
Subject: Registration of MIME media type application/scvp-vp-request

MIME media type name: application

MIME subtype name: scvp-vp-request

Required parameters: None

Optional parameters: None

Encoding considerations: Binary

Security considerations: Carries a request for information.

Interoperability considerations: None

Published specification: RFC 5055

Applications that use this media type: SCVP clients sending validation policy requests

Additional information:

Magic number(s): None
File extension(s): .SPQ
Macintosh File Type Code(s): none

Person & email address to contact for further information:
Ambarish Malpani <ambarish@yahoo.com>

Intended usage: COMMON

Restrictions on usage: None

Author: Ambarish Malpani <ambarish@yahoo.com>

Change controller: IESG

A.4. application/scvp-vp-response

To: ietf-types@iana.org
Subject: Registration of MIME media type application/scvp-vp-response

MIME media type name: application

MIME subtype name: scvp-vp-response

Required parameters: None

Optional parameters: None

Encoding considerations: Binary

Security considerations: None

Interoperability considerations: None

Published specification: RFC 5055

Applications that use this media type: SCVP servers responding to validation policy requests

Additional information:

Magic number(s): None
File extension(s): .SPP
Macintosh File Type Code(s): none

Person & email address to contact for further information:
Ambarish Malpani <ambarish@yahoo.com>

Intended usage: COMMON

Restrictions on usage: This media type can be used with any protocol that can transport digitally signed objects.

Author: Ambarish Malpani <ambarish@yahoo.com>

Change controller: IESG

Appendix B. SCVP over HTTP

This appendix describes the formatting and transportation conventions for the SCVP request and response when carried by HTTP.

In order for SCVP clients and servers using HTTP to interoperate, the following rules apply.

- Clients MUST use the POST method to submit their requests.

- Servers MUST use the 200 response code for successful responses.

- Clients MAY attempt to send HTTPS requests using TLS 1.0 or later, although servers are not required to support TLS.

- Servers MUST NOT assume client support for any type of HTTP authentication such as cookies, Basic authentication, or Digest authentication.

- Clients and servers are expected to follow the other rules and restrictions in [HTTP]. Note that some of those rules are for HTTP methods other than POST; clearly, only the rules that apply to POST are relevant for this specification.

B.1. SCVP Request

An SCVP request using the POST method is constructed as follows:

The Content-Type header MUST have the value "application/scvp-cv-request".

The body of the message is the binary value of the DER encoding of the CVRequest, wrapped in a CMS body as described in Section 3.

B.2. SCVP Response

An HTTP-based SCVP response is composed of the appropriate HTTP headers, followed by the binary value of the BER encoding of the CVResponse, wrapped in a CMS body as described in Section 4.

The Content-Type header MUST have the value "application/scvp-cv-response".

B.3. SCVP Policy Request

An SCVP request using the POST method is constructed as follows:

The Content-Type header MUST have the value "application/scvp-vp-request".

The body of the message is the binary value of the BER encoding of the ValPolRequest, wrapped in a CMS body as described in Section 5.

B.4. SCVP Policy Response

An HTTP-based SCVP policy response is composed of the appropriate HTTP headers, followed by the binary value of the DER encoding of the ValPolResponse, wrapped in a CMS body as described in Section 6. The Content-Type header MUST have the value "application/scvp-vp-response".

Authors' Addresses

Trevor Freeman
Microsoft Corporation,
One Microsoft Way
Redmond, WA 98052
USA.
EMail: trevorf@microsoft.com

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
EMail: housley@vigilsec.com

Ambarish Malpani
Malpani Consulting Services
EMail: ambarish@yahoo.com

David Cooper
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
EMail: david.cooper@nist.gov

Tim Polk
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
EMail: wpolk@nist.gov

Full Copyright Statement

Intellectual Property