Internet Draft
draft-ietf-pkix-scvp-08.txt
March 2002
Expires in six months

Ambarish Malpani
ValiCert, Inc
Russ Housley
RSA Laboratories
Trevor Freeman
Microsoft Corp

Simple Certificate Validation Protocol (SCVP)

Status of this memo

Abstract

The SCVP protocol allows a client to offload certificate handling to a server. The server can give a variety of valuable information about the certificate, such as whether or not the certificate is valid, a certification path to a trust anchor, and so on. SCVP has many purposes, including simplifying client implementations and allowing companies to centralize their trust and policy management.

1. Introduction

Certificate validation is a difficult problem. 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 must be reduced. There are a variety of applications that can use public key certificates but are burdened by the overhead of validating the certificates when all the application really wants is the public key and identity from the

certificate, and a determination of whether or not the certificate may be used for a particular purpose. There are other applications that can perform certification path validation but have no reliable method of constructing a certification path to a trust anchor.

1.1 SCVP overview and requirements

[Once the DPV & DPD Requirements document passes WG Last Call, this section will be updated to reference that document.]

The primary goals of SCVP are to make it easier for applications to deploy systems using a PKI and to allow central administration of PKI policies. Parts of SCVP can be used by clients that do much of the PKI processing themselves and simply want a useful but untrusted server that will collect information for them. Other parts can be used by clients that have complete trust in the server to both offload the work of certificate validation and to ensure that policies are enforced in a consistent fashion across an enterprise.

Untrusted SCVP servers can provide clients the certification paths needed for certificate path validation. They can also provide clients revocation information such as CRLs and OCSP responses that the client can use in certification path validation. These services can be valuable to client systems that do not include the protocols needed to find and download all of the intermediate certificates, CRLs, and OCSP responses needed for the client to perform path validation.

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

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

- The client is in an enterprise that wants to centralize its PKI validation policies, such as which trust anchors and which types of policy checking are performed during certification path validation.

1.2 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 [MUSTSHOULD].

1.3 Open Issues

The following is a list of issues that were raised on earlier versions of this document that have not been fully dealt with here. Comments on

these issues are particularly welcome.

- Extensions can be marked as critical. The usefulness and problems of criticality have been long debated and there has not been a great deal of consensus. In SCVP, marking a request extension as critical says to the server "don't give me an answer unless you understand this extension", and marking a response extension as critical says "don't use this response unless you understand this extension". Without the critical bit in the extensions, either the semantics of extensions would have to be changed to essentially say "all extensions are critical" (which is overkill for some extensions that might really be optional), or the semantics would have to be changed to say "you can never rely on the other party understanding an extension", which would limit the usefulness of some extensions.

- Should we allow another way of specifying trust anchors? If so, it needs to include (1) the trusted issuer name, (2) the trusted public key algorithm, (3) the trusted public key, and (4) optionally, the trusted public key parameters associated with the public key.

- Is there any value to an "unvalidated path"?

- The structure allows for the validation of other objects, such as attribute certificates, to be easily added as Query CHOICE items. Should we change the name of the protocol to reflect this flexibility?

- Can CertBundle contain objects of a different type than the object being queried? For example, if the client wants the server to validate a public key certificate, can the CertBundle contain an attribute certificate?

- Can TrustedCerts contain objects of a different type than the object being queried? For example, if, in the future, the client wants the server to validate a attribute certificate, can the CertBundle contain a public key certificate?

- Should we require the certReplies SEQUENCE items to be listed in a particular order?

- ReplyTypesOfCheck and ReplyWantBack use the Extensions structure. What does the critical bit mean? Should a different structure be used?

- TODO: Need to add OPTIONAL variables in the request to be able to control inputs to the path validation algorithm.

- TODO: Show how to delegate SCVP signing authority.

- TODO: Add extensions to allow client to require server to validate a certificate for a particular context, such as SSL/TLS, S/MIME, or IPsec.

- TODO: Allow the response to be unsigned if it is simply reporting an error. Generating malformed requests should not force the server to perform a private key operation.

- TODO: Change the structure of RequestHash to be algorithm identifier followed by hash value. This will allow any one-way hash algorithm to be used.

- TODO: Explain semantics of thisUpdate and nextUpdate in responses

- TODO: Move error codes that apply to the whole request up a level

2. Protocol

The SCVP protocol uses a simple request-response model. That is, a SCVP client creates a single request and sends it to the server; the server creates a single response and sends it to the client. Typical use of SCVP is expected to be over HTTP, and possibly email. This document registers MIME types for SCVP requests and responses.

3. Request

A SCVP client request to the server MUST be a single SCVPRequest item. A SCVPRequest item MUST be carried in an application/scvp-request MIME body part.

There are two forms of SCVP request: unsigned and signed. A signed request can be used to authenticate the client to the server. A server MAY require all requests to be signed, and a server MAY discard all unsigned requests. Alternatively, a server MAY choose to process unsigned requests.

The unsigned request consists of a PSRequest encapsulated in a ContentInfo.

```
ContentInfo {
  contentType        id-ct-scvp-psRequest,
                             -- (1.2.840.113549.1.9.16.1.10)
  content            PSRequest
}
```

The signed request consists of a PSRequest encapsulated in a SignedData which is in turn encapsulated in a ContentInfo.

```
ContentInfo {
  contentType        id-signedData, -- (1.2.840.113549.1.7.2)
  content            SignedData
}

SignedData {
  version            CMSVersion,
```

```
    digestAlgorithms   DigestAlgorithmIdentifiers,
    encapContentInfo   EncapsulatedContentInfo,
    certificates       CertificateSet, -- (Optional)
    crls               CertificateRevocationLists, -- (Optional)
    signerInfos        SET OF SignerInfos -- (only one in SCVP)
  }

  SignerInfo {
    version            CMSVersion,
    sid                SignerIdentifier,
    digestAlgorithm    DigestAlgorithmIdentifier,
    signedAttrs        SignedAttributes, -- (Required)
    signatureAlgorithm SignatureAlgorithmIdentifier,
    signature          SignatureValue,
    unsignedAttrs      UnsignedAttributes -- (not used in SCVP)
  }

  EncapsulatedContentInfo {
    eContentType       id-ct-scvp-psRequest,
                              -- (1.2.840.113549.1.9.16.1.10)
    eContent           OCTET STRING -- Contains PSRequest
  }
```

3.1 PSRequest

The PSRequest item contains the client request. The PSRequest item contains scvpVersion, query, typesOfCheck, and wantBack items. It MAY also contain an requestNonce and reqExtensions items. (The "PS" in PSRequest means "possibly signed".)

The scvpVersion item MUST contain the integer one (1).

The query item MUST contain a Query. This specification includes only the CertsQuery; however, the use of a CHOICE permits future versions of this protocol to support validation of other objects.

The typesOfCheck item MUST contain a sequence of object identifiers. Each object identifier tells the server what types of checking the client expects the server to perform on the on the query item(s).

The wantBack item MUST contain a sequence of object identifiers. Each object identifier tells the server what the client wants to know about the query item(s).

The RequestNonce item MAY contain an octet string. If present, the octet string MUST contain an identifier generated by the client for the request.

The reqExtensions item MAY contain Extensions. If present, each Extension extends the request. For example, an Extension MAY be used to request a different type of item.

The PSRequest MUST have the following syntax:

```
PSRequest ::= SEQUENCE {
    scvpVersion             INTEGER,
    query                   Query,
    typesOfCheck            TypesOfCheck,
    wantBack                WantBack,
    requestNonce        [1] OCTET STRING OPTIONAL,
    reqExtensions       [2] Extensions OPTIONAL
}
```

3.2 scvpVersion

The scvpVersion item tells the version of SCVP used in a request or a response. The value of the scvpVersion item MUST be one (1). Updates to this specification ought to specify other integer values.

3.3 Query

The Query item specifies the object of the request. One type of object is defined in this specification: CertsQuery. (Other types of queries might be specified in the future.) The CertsQuery is a request for information on one or more certificates. A CertsQuery MUST contain a sequence of one or more certificate. A CertsQuery MAY also contain validityTime, IntermediateCerts, TrustedCerts, RevocationInfo, PolicyID, ConfigurationIdentifier, and QueryExtensions items. Query MUST have the following syntax:

```
Query ::= CHOICE {
    certsQuery       [0] CertsQuery
}

CertsQuery ::= SEQUENCE {
    queriedCerts          SEQUENCE SIZE (1..MAX) OF Cert,
    validityTime      [0] GeneralizedTime OPTIONAL,
    intermediateCerts [1] CertBundle OPTIONAL,
    trustedCerts      [2] CertBundle OPTIONAL,
    revInfos          [3] SEQUENCE SIZE (1..MAX) OF RevocationInfo OPTIONAL,
    policyID          [4] OBJECT IDENTIFIER OPTIONAL,
    configID          [5] OBJECT IDENTIFIER OPTIONAL,
    queryExtensions   [6] Extensions OPTIONAL
}
```

The list of certificates in the Query item tells the server the certificate(s) for which the client wants a reply. The optional validityTime item tells the date and time relative to which the client wants the server to perform the checks and generate responses. The optional intermediateCerts, trustedCerts, revocationInfos, policyID,

and configID items provide context for the client request.

3.4 QueriedCerts

The queriedCerts item MUST contain at least one certificate.

3.5 Cert

The Cert item MUST contain an identifier for the type of certificate and the certificate itself. One type of certificate, for the Internet X.509 PKI [PKIX], is defined, but other types of certificates (such as attribute certificates [AC] or OpenPGP certificates [OpenPGP]) might be defined in the future. Cert MUST have the following syntax:

```
   Cert ::= CHOICE {
     pkixCert        [0] Certificate }
```

The ASN.1 definition of Certificate is imported from [PKIX].

3.6 ValidityTime

The optional validityTime item tells the date and time relative to which the client wants the server to perform the checks and generate responses. If the validityTime is present, it MUST be encoded as GeneralizedTime. If the validityTime is not present, the server MUST respond as if the client provided the at which the server processes the request.

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 validityTime. However, if the server does not have historical information about that time, it MAY either return an error or return information for a later time. A client MUST be prepared to handle responses that contain thisUpdate items that do not match the requested validityTime.

3.7 IntermediateCerts

The intermediateCerts item helps the server create valid certification paths. The intermediateCerts item, when present, provides certificates that the server MAY use when forming a certification path. The certificates in the intermediateCerts item MAY be used by the server in addition to any other certificates that the server has access to when building certification paths. The intermediateCerts item, when present, MUST contain at least one certificate. The intermediateCerts item MUST be structured as a CertBundle. The certificates in the intermediateCerts MUST NOT be trusted by the server just because they are present in this item.

3.8 CertBundle

The CertBundle item contains one or more Cert. The order of the

Cert entries in the bundle is not important. CertBundle MUST have the following syntax:

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

3.9 TrustedCerts

The trustedCerts item optionally specifies the trust anchors to be used by the server. If a trustedCerts item is present, the server MUST NOT use any certification path anchors other than those provided. The trustedCerts item MUST be structured as a CertBundle.

3.10 RevocationInfo

The revInfo item optionally specifies revocation information such as CRLs [PKIX] and OCSP responses [OCSP] that the server MAY use when validating certification paths. The purpose of the revInfo item is to provide revocation information to which the server might not otherwise have access (for example, an OCSP response that the client received along with the certificate). Note that the information in the revInfo item might not be used by the server, such as if the information is for certificates that the server does not use in certification path building.

The types of revocation information that can be provided are: a CRL or an OCSP response.

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

The object identifiers for riType for CRLs and OCSP are id-ri-crl and id-ri-ocsp-response, respectively.

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

    id-ri OBJECT IDENTIFIER ::= { id-pkix 16 }

    id-ri-crl OBJECT IDENTIFIER ::= { id-ri 1 }

    id-ri-ocsp-response OBJECT IDENTIFIER ::= { id-ri 2 }
```

3.11 PolicyID

The policyID optional item specifies the policy identifier that the server MUST use when forming a certification path. The policyID item

MUST contain the OID that defines the policy.

3.12 ConfigID

The configID item, when present, tells the server the SCVP options that the client wants the server to use. The client can use this option instead of specifying other SCVP configuration items such as policyID and trustedCerts. The value of this item is determined by private agreement between the client and the server, but it MUST be represented as an OBJECT IDENTIFIER. The server might want to assign identifiers that indicate that some settings are used in addition to others given in the request; in this way, the configuration identifier might be a shorthand for some SCVP options, but not others.

3.13 QueryExtensions

The QueryExtensions item specifies a list of extensions to the SCVP protocol. For example, a client might request additional information about the certificate(s) in the CertsQuery. The QueryExtensions item, when present, contains a sequence of Extension items, each of which contains an ExtnID item, a Critical item, and an ExtnValue item.

The syntax for Extensions is imported from [PKIX].

3.14 ExtnID

The ExtnID item is an identifier for the extension. It contains the object identifier (OID) of the extension.

3.15 Critical

The Critical item is a BOOLEAN that tells whether the extension is critical. The values for the item are:

```
FALSE - Not critical
TRUE  - Critical
```

In a request, if the Critical item is TRUE, the server MUST NOT process the request unless it understands the extension. In a reply, if the Critical item is TRUE, the client MUST NOT process the response unless it understands the extension.

3.16 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 ExtnID).

3.17 TypesOfCheck

The typesOfCheck item describes the checking that the client wants

the server to perform on the certificate(s) in the Query item. If the typesOfCheck item is present in a request, it can contain one or more types of check. For each type of check specified in the request, the server MUST perform all the checks requested, or return an error.

The types of checks are:
- Build a certification path to a trusted root.
- Build a validated certification path to a trusted root.
- Do revocation status checks on the certification path.

Note that revocation status check inherently includes path construction. Also, building a validated certification path does not imply revocation status checks (although a server may still choose to perform them).

The TypesOfCheck MUST have the following syntax:

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

   id-stc OBJECT IDENTIFIER ::= { id-pkix 17 }

   id-stc-build-path OBJECT IDENTIFIER ::= { id-stc 1 }

   id-stc-build-valid-path OBJECT IDENTIFIER ::= { id-stc 2 }

   id-stc-build-valid-status-checked-path
                           OBJECT IDENTIFIER ::= { id-stc 3 }
```

3.18 WantBack

The wantBack item describes the kind of information the client wants from the server for the certificate(s) in the Query item. If the wantBack item is present in a request, it MUST contain one or more types of information. For each type of information specified in the request, the server MUST return information regarding its finding during the check (in a successful response).

The types of information that can be requested are:
- Certification path built for the certificate
- Proof of revocation status

For example, a request might include a typesOfCheck 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 certification path that the server considers to be valid. This might be used by a client that wants to perform its own certification path validation.

The wantBack MUST have the following syntax:

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

id-swb  OBJECT IDENTIFIER ::= { id-pkix 18 }   -- SCVP want back types
id-swb-cert-path             OBJECT IDENTIFIER ::= { id-swb 1 }
want-arc-status ::= {want-arc 1}
id-swb-revocation-info       OBJECT IDENTIFIER ::= { id-swb 2 }
```

3.19 RequestNonce

The requestNonce optional item is an identifier generated by the client for the request. If the client includes a requestNonce value, then the server MUST return the same RequestNonce in the response. The requestN OCTET STRING. The client SHOULD include a requestNonce item in every request to prevent an attacker acting as a man-in-the-middle by replaying old responses from the server. The value of the nonce SHOULD change with every request sent from the client.

3.20 ReqExtensions

The ReqExtensions optional item specifies a list of extensions to the SCVP request. The ReqExtensions item contains a sequence of Extension items, each of which contains an ExtnID item, a Critical item, and an ExtnValue item.

4. Response

A SCVP server response to the client MUST be a single SCVPResponse item. A SCVPRequest item is carried in an application/scvp-response MIME body part.

There are two forms of an SCVP response: unsigned and signed. An unsigned may only be generated for an error status.

An unsigned response is as follows:

```
ContentInfo {
  contentType        id-ct-scvp-psResponse,
                             -- (1.2.840.113549.1.9.16.1.11)
  content            PSResponse
}
```

The signed response consists of a PSResponse encapsulated in a SignedData which is in turn encapsulated in a ContentInfo.

```
ContentInfo {
  contentType        id-signedData, -- (1.2.840.113549.1.7.2)
  content            SignedData
}
```

```
SignedData {
  version             CMSVersion,
  digestAlgorithms    DigestAlgorithmIdentifiers,
  encapContentInfo    EncapsulatedContentInfo,
  certificates        CertificateSet, -- (Optional)
  crls                CertificateRevocationLists, -- (Optional)
  signerInfos         SET OF SignerInfos -- Only 1 in SCVP
}

SignerInfo {
  version             CMSVersion,
  sid                 SignerIdentifier,
  digestAlgorithm     DigestAlgorithmIdentifier,
  signedAttrs         SignedAttributes, -- (Required)
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature           SignatureValue,
  unsignedAttrs       UnsignedAttributes -- (not used in SCVP)
}

EncapsulatedContentInfo {
  eContentType        id-ct-scvp-psResponse,
                              -- (1.2.840.113549.1.9.16.1.11)
  eContent            OCTET STRING -- Contains PSResponse
}
```

4.1 PSResponse

The PSResponse item contains the server response. The PSResponse MUST contain scvpVersion, producedAt, responseStatus, and requestHash items. The PSResponse MAY also contain replyObjects, requestNonce, and respExtensions optional items. The PSResponse MUST contain exactly one CertReply item for each certificate requested in the request. The requestNonce item MUST be included if the request included a requestNonce item.

The PSResponse MUST have the following syntax:

```
PSResponse ::= SEQUENCE {
    scvpVersion           INTEGER,
    producedAt            GeneralizedTime,
    responseStatus        ResponseStatus,
    requestHash           OCTET STRING,
    replyObjects      [0] ReplyObjects OPTIONAL,
    requestNonce      [1] OCTET STRING OPTIONAL,
    respExtensions    [2] Extensions OPTIONAL
}
```

4.2 scvpVersion

See section 3.2.

4.3 ProducedAt

The ProducedAt item tells the date and time at which the response was produced by the server. The producedAt item represents the date and time in UTC. The producedAt item MUST be structured as GeneralizedTime.

4.3.1 GeneralizedTime

The generalized time type, GeneralizedTime, is a standard ASN.1 type for variable precision representation of time. Optionally, the GeneralizedTime field can include a representation of the time differential between local and Greenwich Mean Time.

GeneralizedTime values MUST be expressed Greenwich Mean Time (Zulu) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds.

This rule for GeneralizedTime applies to all occurrences of GeneralizedTime in this specification and is the same as the rule used in certificate profiles [PKIX].

4.4 ResponseStatus

The responseStatus item gives status information to the 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 optional string MAY optionally be used to transmit status information. The client MAY choose to display the string to the human user. However, because there is no way to know the languages understood by the human user, the string may be of little or no use to them.

The ResponseStatus MUST have the following syntax:

```
ResponseStatus ::= SEQUENCE {
    statusCode SCVPStatusCode,
    errorMessage [0] UTF8String OPTIONAL
}

SCVPStatusCode ::= ENUMERATED {
    okay                    (0),
    skipUnrecognizedItems   (1),
    tooBusy                (10),
    badStructure           (20),
    unsupportedVersion     (21),
    abortUnrecognizedItems (22),
```

```
    unrecognizedSigKey     (23),
    badSignature           (24),
    unableToDecode         (25),
    notAuthorized          (26)
}
```

The meaning of the various status codes are:

```
     0  The request was fully processed
     1  The request included unrecognized items; continuing

    10  Too busy; try again later

    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; aborting
    23  The key given in the RequestSignature is not recognized
    24  The signature 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 items; continuing
    28  The request included unsupported items; aborting
```

4.5 RequestHash

The requestHash item is the SHA-1 hash of the PSRequest. The requestHash item serves the following purposes:

- It allows a client to determine that the request was not maliciously modified.

- It allows the client to associate a response with a request when using connectionless protocols. (Although, the RequestNonce provides a better mechanism for matching requests and responses.)

The requestHash item does not provide authentication.

4.6 ReplyObjects

The replyObjects item returns objects to the client. In this specification, the replyObjects item is always a certReplies, which is a SEQUENCE of CertReply, each of which tells the client about a single certificate from the request. The CertReply item MUST contain cert, replyStatus, and thisUpdate items, and it MAY contain a nextUpdate item. The CertReply MAY also contain the following optional objects: ValidationStatus, RevocationStatus, PublicKey, CertSubject, ValidationChain, RevocationProof, and SingleReplyExtensions.

The presence or absence of the ValidationStatus, RevocationStatus, PublicKey, CertSubject, ValidationChain, and RevocationProof objects in the CertReply item is controlled by the TypesOfCheck, and WantBack

items in the request. A server MUST include one of the above items for each related item requested in the TypesOfCheck, and WantBack items.

```
ReplyObjects ::= CHOICE {
    certReplies     [0] SEQUENCE SIZE (1..MAX) OF CertReply
}

CertReply ::= SEQUENCE {
    cert                        Cert,
    replyStatus                 ReplyStatus,
    thisUpdate                  GeneralizedTime,
    nextUpdate              [0] GeneralizedTime OPTIONAL,
    replyTypesOfCheck       [1] Extensions OPTIONAL,
    replyWantBack           [2] Extensions OPTIONAL,
    singleReplyExtensions   [3] Extensions OPTIONAL
}
```

4.7 ReplyStatus

The ReplyStatus item gives status information to the client about the request for the specific certificate. Note that the ResponseStatus item is different than 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 complete list of status codes for the ReplyStatus item is:

```
ReplyStatus ::= ENUMERATED {
    success                 (0),
    certTypeUnrecognized    (1),
    typeOfCheckUnrecognized (2),
    wantBackUnrecognized    (3),
    certMalformed           (4),
    policyIDUnrecognized    (5),
    configInfoUnrecognized  (6),
    unauthorizedRequest     (7)
}
```

The ReplyStatus codes have the following meaning:

```
    0  Success: a definitive answer follows
    1  Failure: the certificate type is not recognized
    2  Failure: an item wanted in TypesOfCheck is not recognized
    3  Failure: an item wanted in WantBack is not recognized
    4  Failure: the certificate was malformed
    5  Failure: the mandatory PolicyID is not recognized
    6  Failure: the ConfigurationIdentifier is not recognized
    7  Failure: unauthorized request
```

Status code 4 is used to tell the client that the request was properly formed but the certificate in question was not. This is useful to

clients that cannot parse a certificate.

4.8 ThisUpdate

The ThisUpdate item tells the time at which the information in the CertReply was correct. The ThisUpdate item represents the date as UTC.

4.9 NextUpdate

The NextUpdate item tells the time at which the server expects additional information regarding the validity of the certificate to become available. Such information could change the status of the certificate, but it might not change the status of the certificate. The NextUpdate item represents the date at UTC.

4.10 ReplyTypesOfCheck

The ReplyTypesOfCheck contains the responses to the TypesOfCheck item in the request. It has the same form as the Extensions item, and the OIDs in the ReplyTypesOfCheck item MUST match the OIDS in the TypesOfCheck item. The criticality bit MUST NOT be set.

The value for path building to a trusted root, {type-arc 0}, can be one of the following:

Value	Meaning
0	Built a path
1	Could not build a path

The value for path validation to a trusted root, {type-arc 1}, can be one of the following:

Value	Meaning
0	Valid
1	Not valid

The value for the revocation status, {type-arc 2}, can be one of the following:

Value	Meaning
0	Good
1	Revoked
2	Unknown

4.11 ReplyWantBack

The ReplyWantBack contains the responses to the WantBack item

in the request. It has the same form as the Extensions item, and the OIDs in the ReplyWantBack item MUST match the OIDS in the WantBack item. The criticality bit MUST NOT be set.

The value for the certification chain used to verify the certificate in the request, {want-arc 0}, is a CertBundle item.

The value for the proof of revocation status, {want-arc 1}, is a RevocationProof item.

4.12 RevocationProof

The RevocationProof item gives the client the proof that the server used to check revocation. The structure of the RevocationProof item is the same as an Extensions item. The OIDs in the RevocationProof item are the same as those in the RevocationInfo item.

4.13 ResponseSignature

The ResponseSignature item is the signature of the PSResponse item.

The client SHOULD check the signature on every signed message it receives from the server. In order to check the signature, the client MUST know and rely on the public signing key of the server. The client could have obtained the server's public key through an out-of-band mechanism of direct trust or through a certificate that chains to a root that the client trusts to delegate this type of authority.

5. ASN.1 Syntax for SCVP

This section defines the syntax for SCVP messages. The semantics for the messages are defined in sections 2, 3, and 4.

5.1 ASN.1 Module definition

```
SCVP DEFINITIONS EXPLICIT TAGS ::=

BEGIN

IMPORTS

      -- Directory Authentication Framework (X.509)
           Certificate, AlgorithmIdentifier
           FROM AuthenticationFramework { joint-iso-itu-t ds(5)
                    module(1) authenticationFramework(7) 3 }

      -- CMS Imports
           ContentInfo, SignedData, CMSVersion,
           FROM CryptographicMessageSyntax { iso(1) member-body(2)
                   us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
                   modules(0) cms(1) }
```

```
      -- PKIX Imports
          Name, Extensions,
             FROM PKIX1Explicit88 {iso(1) identified-organization(3)
                  dod(6) internet(1) security(5) mechanisms(5) pkix(7)
                  id-mod(0) id-pkix1-explicit-88(1)};


PSRequest ::= SEQUENCE {
    scvpVersion                 INTEGER,
    query                    Query,
    typesOfCheck             TypesOfCheck,
    wantBack                 WantBack,
    requestNonce         [1] OCTET STRING OPTIONAL,
    reqExtensions        [2] Extensions OPTIONAL
}

Query ::= CHOICE {
    certsQuery       [0] CertsQuery
}

CertsQuery ::= SEQUENCE {
    queriedCerts           SEQUENCE SIZE (1..MAX) OF Cert,
    validityTime       [0] GeneralizedTime OPTIONAL,
    intermediateCerts  [1] CertBundle OPTIONAL,
    trustedCerts       [2] CertBundle OPTIONAL,
    revocationInfos    [3] SEQUENCE SIZE (1..MAX) OF RevocationInfo OPTIONAL,
    policyID           [4] OBJECT IDENTIFIER OPTIONAL,
    configurationIdentifier [5] OBJECT IDENTIFIER OPTIONAL,
    queryExtensions         [6] Extensions OPTIONAL
}

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

Cert ::= CHOICE {
    pkixCert         [0] Certificate
}


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

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

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


Signature ::= SEQUENCE {
    signerName               Name,
```

```
    signatureAlgorithm       AlgorithmIdentifier,
    signatureBits            BIT STRING,
    certs                [0] CertBundle OPTIONAL
}

PSResponse ::= SEQUENCE {
    scvpVersion               INTEGER,
    producedAt            GeneralizedTime,
    responseStatus        ResponseStatus,
    requestHash           OCTET STRING,
    replyObjects      [0] ReplyObjects OPTIONAL,
    requestNonce      [1] OCTET STRING OPTIONAL,
    respExtensions    [2] Extensions OPTIONAL
}

ResponseStatus ::= SEQUENCE {
    statusCode SCVPStatusCode,
    errorMessage [0] UTF8String OPTIONAL
}

SCVPStatusCode ::= ENUMERATED {
    okay                    (0),
    skipUnrecognizedItems   (1),
    tooBusy                (10),
    badStructure           (20),
    unsupportedVersion     (21),
    abortUnrecognizedItems (22),
    unrecognizedSigKey     (23),
    badSignature           (24),
    unableToDecode         (25),
    notAuthorized          (26)
}


ReplyObjects ::= CHOICE {
    certReplies     [0] SEQUENCE SIZE (1..MAX) OF CertReply
}

CertReply ::= SEQUENCE {
    cert                  Cert,
    replyStatus           ReplyStatus,
    thisUpdate            GeneralizedTime,
    nextUpdate        [0] GeneralizedTime OPTIONAL,
    replyTypesOfCheck [1] Extensions OPTIONAL,
    replyWantBack     [2] Extensions OPTIONAL,
    singleReplyExtensions        [3] Extensions OPTIONAL
}

-- The encoding of the value for path validation and revocation status
-- will be as an INTEGER
```

```
ReplyStatus ::= ENUMERATED {
    success                  (0),
    certTypeUnrecognized     (1),
    typeOfCheckUnrecognized  (2),
    wantBackUnrecognized     (3),
    certMalformed            (4),
    policyIDUnrecognized     (5),
    configInfoUnrecognized   (6),
    unauthorizedRequest      (7)
}

-- OIDs

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

    id-psRequest OBJECT IDENTIFIER ::= { id-ct 10 }
    id-psResponse OBJECT IDENTIFIER ::= { id-ct 11 }

    id-pkix OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
             dod(6) internet(1) security(5) mechanisms(5) pkix(7) }
    id-ri   OBJECT IDENTIFIER ::= { id-pkix 16 }   -- revocation
                                                   -- information types

    id-ri-crl OBJECT IDENTIFIER ::= { id-ri 1 }
    id-ri-ocsp-response OBJECT IDENTIFIER ::= { id-ri 2 }

    id-stc  OBJECT IDENTIFIER ::= { id-pkix 17 }   -- SCVP check type arc}
    id-stc-build-path            OBJECT IDENTIFIER ::= { id-stc 1 }

    id-stc-build-valid-path      OBJECT IDENTIFIER ::= { id-stc 2 }
    id-stc-build-valid-status-checked-path
                               OBJECT IDENTIFIER ::= { id-stc 3 }

    id-swb  OBJECT IDENTIFIER ::= { id-pkix 18 }   -- SCVP want back types
    id-swb-cert-path             OBJECT IDENTIFIER ::= { id-swb 1 }
    want-arc-status ::= {want-arc 1}
    id-swb-revocation-info       OBJECT IDENTIFIER ::= { id-swb 2 }

END
```

6. Security Considerations

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 check the RequestHash in the response and ensure that it matches their original request. Requests contain a lot of information that affects the response and clients need to ensure that the server response corresponds to the expected request.

When the SCVP response is used to determine the validity of a certificate, the client MUST validate the signature on the response to ensure that it was generated by the expected SCVP server.If the client does not check the signature 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 reply matches that in the request, an attacker can replay previous responses from the server. [This is not true if the request hash is used as a nonce by the client.]

If the server does not require some sort of authorization (such as signed requests), an attacker can get the server to reply 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.

A. References

[MUSTSHOULD] "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119.

[CMS] "Cryptographic Message Syntax", RFC 2630.

[OCSP] "PKIX Online Certificate Status Protocol (OCSP)", RFC 2560.

[OpenPGP] "OpenPGP Message Format", RFC 2440.

[PKIX] "PKIX Certificate and CRL Profile", RFC 2459.

[SHA-1] "Secure Hash Standard", NIST FIPS publication 180-1, April 1995.

[UTF8] "UTF-8, a transformation format of ISO 10646", RFC 2279.

[AC] NEED THE REFERENCE

B. Acknowledgments

The lively debate in the PKIX Working Group also had a significant impact on the types of items described in this protocol. Denis Pinkas suggested some additional requirements for the protocol, and Mike Myers helped point out sections that needed clarification. Frank Balluffi and Ameya Talwalkar were responsible for the first implementation and suggestions on a few deficiencies in the document. John Thielens and Peter Sylvester provided a lot of good input to help improve this document.

C. MIME Registrations

C.1 application/scvp-request

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

MIME media type name: application

MIME subtype name: scvp-request

Required parameters: format

Optional parameters: None

Encoding considerations: binary

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

Interoperability considerations: None

Published specification: IETF PKIX Working Group Draft on Simple Certificate Validation Protocol - SCVP

Applications which use this media type: SCVP clients

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@valicert.com>

Intended usage: COMMON

Author/Change controller:

Ambarish Malpani <ambarish@valicert.com>

C.2 application/scvp-response

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

MIME media type name: application

MIME subtype name: scvp-response

Required parameters: format

Optional parameters: None

Encoding considerations: binary

Security considerations: Carries a cryptographically signed response

Interoperability considerations: None

Published specification: IETF PKIX Working Group Draft on Simple Certificate Validation Protocol - SCVP

Applications which use this media type: SCVP servers

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@valicert.com>

Intended usage: COMMON

Author/Change controller:
Ambarish Malpani <ambarish@valicert.com>

D. SCVP over HTTP

This section describes the formatting that will be done to the request and response to support HTTP.

D.1 Request

HTTP based SCVP requests can use the POST method to submit their requests. Where privacy is a requirement, SCVP transactions exchanged using HTTP MAY be

protected using either TLS/SSL or some other lower layer protocol.

An SCVP request using the POST method is constructed as follows: The Content-Type header MUST have the value "application/scvp-request". The Content-Length header MUST be present and have the exact length of the request. The body of the message is the binary value of the DER encoding of the FullRequest. Other HTTP headers MAY be present and MAY be ignored if not understood by the requestor.

Sample Content-Type headers are:
Content-Type: application/scvp-request

D.2 Response

An HTTP-based SCVP response is composed of the appropriate HTTP headers, followed by the binary value of the DER encoding of the FullResponse. The Content-Type header MUST have the value "application/scvp-response". The Content-Length header MUST be present and specify the length of the response. Other HTTP headers MAY be present and MAY be ignored if not understood by the requestor.

E. Author Contact Information

Ambarish Malpani
ValiCert, Inc.
339 N. Bernardo Ave.
Mountain View, CA 94043
ambarish@valicert.com

Russell Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
USA
rhousley@rsasecurity.com

Trevor Freeman
Microsoft Corporation,
One Microsoft way
Redmond, WA98052
trevorf@microsoft.com

F. Changes Between Versions of This Document

F.1 Difference between -04 and -05

1. Removed the XML format of the syntax

2. Used CMS as the base formatting mechanism

3. Changed the format of RevocationInfo

4. Specified rules for GeneralizedTime usage

5. Added a question about the benefit of other types of authentication of responses (not just signatures).

F.2 Differences between -05 and -06

1. Added authors

2. Fixed language and spelling mistakes

3. Changed version to scvpVersion

F.3 Differences between -06 and -07

1. Updated authors list

2. Closed open issue of whether we should deal with cases where the client doesn't have the certificate itself

3. Added text for different types of request and wantbacks

4. Allow for unsigned error responses

5. Moved changes to the end and renumbered sections

6. Added some OIDs that were TBD