Enrollment over Secure Transport
draft-ietf-pkix-est-03

Abstract

This document profiles certificate enrollment for clients using
Certificate Management over CMS (CMC) messages over a secure
transport. This profile, called Enrollment over Secure Transport
(EST), describes a simple yet functional certificate management
protocol targeting Public Key Infrastructure (PKI) clients that need
to acquire client certificate(s) and associated Certification
Authority (CA) certificate(s).

Status of this Memo

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents.
carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................... 4
   1.1. Terminology ......................................... 5
2. Operational Scenario Overviews ............................ 5
   2.1. Obtaining CA Certificates ............................ 6
   2.2. Initial Enrollment .................................... 7
      2.2.1. Previously Installed Client Certificate .......... 7
      2.2.2. Username/Password Distributed Out-of-Band ........ 7
   2.3. Client Certificate Re-issuance ....................... 8
      2.3.1. Re-issuance of Signature Certificates ........... 8
      2.3.2. Re-issuance of Key Exchange Certificates ......... 8
   2.4. Server Key Generation .................................. 8
   2.5. Full PKI Request messages ............................ 8
   2.6. CSR (Certificate Signing Request) Attributes Request .. 8
3. Protocol Design and Layering ............................... 9
   3.1. Application Layer Design ............................. 12
   3.2. HTTP Layer Design .................................... 12
      3.2.1. HTTP headers for control ......................... 12
      3.2.2. HTTP URIs for control ........................... 13
      3.2.3. HTTP-Based Client Authentication ................. 14
      3.2.4. Message types .................................. 15
   3.3. TLS Layer Design ..................................... 16
      3.3.1. TLS for transport security ....................... 17
         3.3.1.1. TLS-Based Server Authentication .............. 17
         3.3.1.2. TLS-Based Client Authentication .............. 17
         3.3.1.3. Certificate-less TLS Mutual Authentication .... 18
   3.4. Proof-of-Possession ................................... 18
   3.5. Linking Identity and POP information ................ 18
4. Protocol Exchange Details .................................. 20
   4.1. Server Authorization .................................. 20
   4.2. Client Authorization .................................. 20
   4.3. Distribution of CA certificates ....................... 21
      4.3.1. Distribution of CA certificates request .......... 21
      4.3.2. Bootstrap Distribution of CA certificates ....... 21
      4.3.3. Distribution of CA certificates response ......... 22
   4.4. Client Certificate Request Functions ................ 23
      4.4.1. Simple Enrollment of Clients ..................... 23
      4.4.2. Simple Re-Enrollment of Clients .................. 24
      4.4.3. Simple Enroll and Re-Enroll Response ............. 24
   4.5. Full CMC ........................................... 25
4.5.1. Full CMC Request ........................................ 25
4.5.2. Full CMC Response ........................................ 26
4.6. Server-side Key Generation ................................. 26
  4.6.1. Server-side Key Generation Request .................. 27
  4.6.2. Server-side Key Generation Response ................. 27
4.7. CSR Attributes ............................................. 28
  4.7.1. CSR Attributes Request ................................ 28
  4.7.2. CSR Attributes Response ............................... 28
5. IANA Considerations ........................................... 29
6. Security Considerations ....................................... 31
7. References ..................................................... 32
  7.1. Normative References .................................... 32
  7.2. Informative References .................................. 35
Appendix A. Operational Scenario Example Messages .............. 36
  A.1. Obtaining CA Certificates ............................... 36
  A.2. Previously Installed Signature Certificate ............ 37
  A.3. Username/Password Distributed Out-of-Band ............. 39
  A.4. Re-Enrollment ............................................ 42
  A.5. Server Key Generation ................................... 43
  A.6. CSR Attributes ........................................... 47
Authors’ Addresses ............................................. 47
1. Introduction

This document profiles certificate enrollment for clients using Certificate Management over CMS (CMC) [RFC5272] messages over a secure transport. Enrollment over Secure Transport (EST) describes the use of TLS 1.1 [RFC4346] (or a later version) and HTTP 1.1 [RFC2616] to provide an authenticated and authorized channel for Simple PKI Requests and Responses [RFC5272].

Architecturally the EST service is located between a CA and the client device. It performs several functions traditionally allocated to the PKI role of the Registration Authority (RA). The nature of the communication of EST server to CA is not described in this document.

EST adopts the CMP [RFC4210] model for CA certificate rollover, but does not use the CMP message syntax or protocol. EST servers are extensible in that new functions may be defined to provide additional capabilities not specified in CMC [RFC5272]. Non-CMC-based extensions such as the requesting of Certificate Signing Request attributes and requests for server generated keys are defined in this document.

EST specifies the transferring of messages securely over HTTPS [RFC2818] where the HTTP headers and content types are used in conjunction with TLS. HTTPS operates over TCP; this document does not specify EST over DTLS/UDP. Figure 1 shows how the layers build upon each other.
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 [RFC2119].

It is assumed that the reader is familiar with the terms and concepts described in PKCS#10 [RFC2314], HTTPS [RFC2818], CMP [RFC4210], CMC [RFC5272][RFC5273][RFC5274], and TLS [RFC5246].

2. Operational Scenario Overviews

This section provides an informative overview of the operational scenarios to better introduce the reader to the protocol discussion. This section does not include [RFC2119] key words.

Both the EST clients and server are configured with information that will be the basis of authentication and authorization. The specific initialization data depends on the methods available in the client device and server, but can include shared secrets, network service
names and locations (e.g. a URI [RFC3986]), trust anchor information
(e.g. current CA certificate or third party TA(s) or a hash of the
CA’s root certificate), and enrollment keys and certificates.
Depending on the enterprise’s acquisition and network management
practices, some initialization may be performed by the vendor prior
to client delivery. In that case, the client device vendor will
provide data, such as trust anchors, to the enterprise via a secure
procedural mechanism. The distribution of this initial information
is out of scope.

Distribution of trust anchors and certificates can be made through
the EST server. However, nothing can be inferred about the
authenticity of these trust anchors and certificates until an out-of-
band mechanism from the above list is used to verify them.

Sections 2.1-2.3 very closely mirror the text of the Scenarios
Appendix of [RFC6403] with such modifications as are appropriate for
this profile. (Our thanks are extended to the authors of that
document). More importantly, Sections 2.1-2.6 mirror the set of EST
functions (see Figure 4) and provide an informative overview of EST’s
capabilities.

The client device begins by initiating a TLS-secured HTTP session
with the EST server. The specific EST service requested is named in
an operational URI portion. The client device and server
authenticate each other, and the client ascertains the authorization
of the server. The server ascertains the authorization of the client
and services the request.

2.1. Obtaining CA Certificates

The EST client can request a copy of the current CA certificates.
The EST client is assumed to perform this operation before performing
other operations.

The EST client authenticates and authorizes the EST server when
requesting the current CA certificates. As detailed in
Section 3.3.1.1 and Section 3.3.1.3) the available options include:

- Verifying the EST server’s HTTPS URI against the EST server’s
certificate using third party TAs (similar to a common HTTPS
exchange). This allows the EST server and client to leverage
existing TAs that might be known to the EST client.

- The client can leverage a previously distributed trust anchor
specific to the EST server. This allows the EST client to use an
existing, potentially older, CA certificate to request more recent
CA certificates.
o  For bootstrapping the EST client can accept manual authentication performed by the end user as detailed in Section 4.3.2.

Client authentication is not required for this exchange, so it is trivially supported by the EST server.

2.2. Initial Enrollment

After authenticating an EST server and verifying that it is authorized to provide services to the client, an EST client can acquire a certificate by submitting an enrollment request to that server.

The EST server authenticates and authorizes the EST client as specified in Section 3.3.1.2 and Section 4.2. The methods described in the normative text that are expanded on in this overview include:

o  Previously installed certificate (e.g., Manufacturer Installed Certificate or 3rd party issued certificate);

o  Username/password distributed out-of-band

2.2.1. Previously Installed Client Certificate

If the EST client has a previously installed certificate that was issued by a 3rd party this certificate can be used to authenticate the client’s request for a certificate from the EST server’s CA. An EST client responds to the EST server’s TLS certificate request message with the existing certificate (i.e., it provides the previously issued certificate to the EST server). The EST server will authenticate the client’s existing certificate and authorize the client’s request as described in Section 3.3.1.2.

2.2.2. Username/Password Distributed Out-of-Band

When the EST client is not authenticated during the TLS handshake (see Section 3.3.1.2), or if the EST server wishes additional authentication information, the EST server can requests that the EST client submit a username/password using the HTTP Basic or Digest Authentication methods. See Section 3.2.3.

Alternately, the server and client can mutually authenticate using certificate-less TLS authentication (Section 3.3.1.3).
2.3. Client Certificate Re-issuance

An EST client can renew/rekey an existing client certificate by submitting a re-enrollment request to an EST server. As with initial enrollment, the EST server authenticates the client using any combination of the existing client certificate (see Section 3.3.1.2) and/or HTTP Basic or Digest Authentication with a username/password (see Section 3.2.3).

Two common renew/rekey scenarios for clients that are already enrolled are described here. One addresses the renew/rekey of signature certificates and the other addresses the renew/rekey of key exchange certificates. The certification request message includes the same Subject and SubjectAltName as the current key exchange certificate with name changes handled as specified in Section 4.4.2.

2.3.1. Re-issuance of Signature Certificates

When a signature certificate is re-issued, the existing certificate can be used by an EST client for authentication.

2.3.2. Re-issuance of Key Exchange Certificates

When a key exchange certificate is re-issued an existing signature certificate is used by an EST client for authentication. If there is no current signature certificate available, the EST server falls back on the HTTP authentication method (Section 3.2.3).

2.4. Server Key Generation

The EST client can request a server-generated certificate and key pair.

2.5. Full PKI Request messages

Full PKI Request messages can be transported via EST with the Full CMC Request function, allowing access to functionality not provided by the Simple Enrollment of Clients functions. Full PKI Request messages are defined in Sections 3.2 and 4.2 of [RFC5272]. See Section 4.5 for a discussion of how EST provides a transport for these functions.

2.6. CSR (Certificate Signing Request) Attributes Request

Prior to sending an enrollment request to an EST server, an EST client can query the EST server for the set of additional attribute(s) that the client is requested to supply in the subsequent enrollment request(s).
3. Protocol Design and Layering

Figure 2 provides an expansion of Figure 1 describing how the layers are used. Each aspect is described in more detail in the sections that follow.

EST Layering:

Protocols and uses:

<table>
<thead>
<tr>
<th>Message types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- &quot;Simple PKI&quot; messages (incorporating proof-of-possession)</td>
</tr>
<tr>
<td>- CA certificate retrieval</td>
</tr>
<tr>
<td>- &quot;Full PKI&quot; messages (OPTIONAL)</td>
</tr>
<tr>
<td>- CSR attribute request (OPTIONAL)</td>
</tr>
<tr>
<td>- Server-generated key request (OPTIONAL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- HTTP headers and URIs for control</td>
</tr>
<tr>
<td>- Content-Type headers specify message type</td>
</tr>
<tr>
<td>- Headers for control/error messages</td>
</tr>
<tr>
<td>- URIs for selecting functions</td>
</tr>
<tr>
<td>- Basic or Digest authentication if no client certificate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLS for transport security</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Authentication is REQUIRED for EST server</td>
</tr>
<tr>
<td>- OPTIONAL for EST clients</td>
</tr>
<tr>
<td>- Indirectly provides proof-of-identity for EST</td>
</tr>
<tr>
<td>- Provides communications integrity</td>
</tr>
<tr>
<td>- Channel Binding [RFC5929] to link proof-of-identity with message-based proof-of-possession. (OPTIONAL)</td>
</tr>
</tbody>
</table>

Figure 2

Specifying HTTPS as the secure transport for enrollment messages introduces two 'layers' to communicate authentication and control messages: TLS and HTTP.
The TLS layer provides message authentication and integrity during transport. The proof-of-identity is supplied by either the certificate exchange during the TLS handshake or within the HTTP layer headers. The message type along with control/error messages are included in the HTTP headers.

The TLS and HTTP layer provided proof-of-identity means the CMC [RFC5272] Section 3.1 note that "the Simple PKI Request MUST NOT be used if a proof-of-identity needs to be included" is not applicable and thus the Simple PKI message types are used.

The TLS layer certificate exchange provides a method for authorizing client enrollment requests using existing certificates. Such existing certificates may have been issued by the CA (from which the client is requesting a certificate) or they may have been issued under a distinct PKI (e.g., an IEEE 802.1AR IDevID [IDevID] credential).

Proof-of-possession is a distinct issue from proof-of-identity and is included in the Simple PKI message type as described in Section 3.4. A method of linking proof-of-identity and proof-of-possession is described in Section 3.5.

This document also defines transport for CMC [RFC5272] specification compliant with CMC Transport Protocols [RFC5273].

During the protocol operations various different certificates can be used. The following table provides an informative overview. End-entities MAY have one or more certificates of each type listed in Figure 3:
Certificates/Trust-anchors and their corresponding uses:

<table>
<thead>
<tr>
<th>Certificate/TA database</th>
<th>Issuer</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EST server certificate</td>
<td>The CA served by the EST server</td>
<td>Presented by the EST server during the TLS handshake</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST server certificate</td>
<td>An unrelated CA e.g., a Web site CA</td>
<td>Presented by the EST server during the TLS handshake</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST client Trust Anchor Database</td>
<td>Trust anchor for the CA issuing certificates.</td>
<td>EST clients use this trust anchor database to authenticate certificates issued by the CA, including EST server certificates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST client third party Trust Anchor Database</td>
<td>Trust anchors for third party CAs e.g., a list of Web site CA root certificates</td>
<td>EST clients can use this trust anchor database to authenticate an EST server that uses an externally issued certificate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST client certificate</td>
<td>An unrelated CA e.g., a device manufacturer</td>
<td>Presented by the EST client to the EST server by clients that have not yet enrolled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST client certificate</td>
<td>The CA served by the EST server</td>
<td>Presented by the EST client to PKI End Entities. Including to the EST server during future EST operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST client certificate</td>
<td>The CA served by the EST server</td>
<td>Clients can obtain certs that can not be used for EST authentication (e.g., Key Encryption certs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.1. Application Layer Design

An EST client SHOULD have its own client certificate suitable for TLS client authentication (e.g., the digitalSignature bit is set). The client certificate, if available, MUST be used when authenticating to the EST server. If a client does not have a certificate, then the client MUST use HTTP Basic or Digest authentication credentials (see Section 3.2.3). HTTP authentication provides a bootstrap for clients that have not yet been issued a certificate. EST clients obtaining a certificates for other protocol purposes are RECOMMENDED to first obtain an appropriate certificate for use when authenticating to the EST server.

The client also SHOULD also have a CA certificate that will be used to authenticate the EST server.

An EST client MUST be capable of generating and parsing Simple PKI messages (see Section 4.4). Generating and parsing Full PKI messages is OPTIONAL (see Section 4.5). The client MUST also be able to request CA certificates from the EST server and parse the returned "bag" of certificates (see Section 4.3). Requesting CSR attributes and parsing the returned list of attributes is OPTIONAL (see Section 4.7).

3.2. HTTP Layer Design

HTTP is used to transfer EST messages. URIs are provisioned for handling each media type (i.e., message type) as described in Section 3.2.2. HTTP is also used for client authentication services when TLS client authentication is not available due to lack of a client certificate suitable for use by TLS, as detailed in Section 3.2.3. Registered media types are used to convey EST messages as specified in Figure 5.

HTTP 1.1 [RFC2616] and above support persistent connections. As given in Section 8.1 of that RFC persistent connections may be used to reduce network and processing load associated with multiple HTTP requests. EST does not require or preclude persistent HTTP connections and their use is out of scope of this specification.

3.2.1. HTTP headers for control

This document profiles the HTTP content-type header (as defined in [RFC2046], but see Figure 5 for specific values) to indicate the media type for EST messages and to specify control messages for EST. The HTTP Status value is used to communicate success or failure of
EST functions Support for the HTTP authentication methods is available for a client that does not have a certificate.

CMC uses the same messages for certificate renewal and certificate rekey. This specification defines the renewal and rekey behavior of both the client and server. It does so by using the HTTP control mechanisms employed by the client and server as opposed to using CMC.

Various media types as indicated in the HTTP content-type header are used to transfer EST messages. Media types used by EST are specified in Section 3.2.4.

3.2.2. HTTP URIs for control

This profile supports six operations indicated by specific URIs:

Operations and their corresponding URIs:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation Path</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution of CA certificates</td>
<td>/CACerts</td>
<td>Section 4.3</td>
</tr>
<tr>
<td>(MUST)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enrolment of new clients (MUST)</td>
<td>/simpleEnroll</td>
<td>Section 4.4</td>
</tr>
<tr>
<td>Re-Enrollment of existing clients (MUST)</td>
<td>/simpleReEnroll</td>
<td>Section 4.4.1</td>
</tr>
<tr>
<td>Full CMC (OPTIONAL)</td>
<td>/fullCMC</td>
<td>Section 4.5</td>
</tr>
<tr>
<td>Server-side Key Generation (OPTIONAL)</td>
<td>/serverKeyGen</td>
<td>Section 4.6</td>
</tr>
<tr>
<td>Request CSR attributes (OPTIONAL)</td>
<td>/CSRAttrs</td>
<td>Section 4.7</td>
</tr>
</tbody>
</table>

Figure 4

An HTTP base path common for all of an EST server’s requests is defined in the form of an path-absolute ([RFC3986], section 3.3). The operation path (Figure 4 is appended to the base path to form the URI used with HTTP GET or POST to perform the desired EST operation.

An example:

With a base path of "/arbitrary/path" and an operation path of
"/CACerts", the EST client would combine them to form an absolute path of "/arbitrary/path/CACerts". Thus, to retrieve the CA’s certificates, the EST client would use the following HTTP request:

GET /arbitrary/path/CACerts HTTP/1.1

Likewise, to request a new certificate in this example scheme, the EST client would use the following request:

POST /arbitrary/path/simpleEnroll HTTP/1.1

The mechanisms by which the EST server interacts with an HTTPS server to handle GET and POST operations at these URIs is outside the scope of this document. The use of distinct operation paths simplifies implementation for servers that do not perform client authentication when distributing /CACerts responses.

EST clients are provided with the base path URI of the EST server. Potential methods of distributing the URI are discussed within the Security Considerations (see Section 6 and Section 4.1).

An EST server MAY provide additional, services using other URIs.

An EST server MAY use multiple base paths in order to provide service for multiple CAs. Each CA would use a distinct base path, but operations are otherwise the same as specified for an EST server operating on behalf of only one CA.

3.2.3. HTTP-Based Client Authentication

An EST server that has authenticated itself to the client MAY request HTTP-based client authentication. This request can be in addition to successful TLS client authentication (Section 3.3.1.2) if EST server policy requires additional authentication (for example the EST server wishes to confirm the EST client "knows" a password in addition to "having" an existing client certificate). Or HTTP-based client authentication can be an EST server policy specified fallback in situations where the EST client did not successfully complete the TLS client authentication (for example if the EST client is enrolling for the first time or the existing EST client certificates can not be used for TLS client authentication).

HTTP Basic and Digest authentication MUST only be performed over TLS 1.1 [RFC4346] (or later). As specified in CMC: Transport Protocols [RFC5273] the server "MUST NOT assume client support for any type of HTTP authentication such as cookies, Basic authentication, or Digest authentication". Clients intended for deployments where password authentication is advantageous SHOULD support the Basic and Digest
authentication mechanism. Servers MAY provide configuration mechanisms for administrators to enable Basic and Digest authentication methods.

Servers that wish to use Basic and Digest authentication reject the HTTP request using the HTTP defined WWW-Authenticate response-header ([RFC2616], Section 14.47). At that point the client SHOULD repeat the request, including the appropriate Authorization Request Header ([RFC2617], Section 3.2.2) if the client is capable of using the Basic or Digest authentication. If the client is not capable then the client MUST terminate the connection.

Clients MAY set the username to the empty string (""") if they wish to present a "one-time password" or "PIN" that is not associated with a username.

Support for HTTP-based client authentication has security ramifications as discussed in Section 6. The client MUST NOT respond to the server’s HTTP authentication request unless the client has authenticated the EST server (as per Section 4.1).

3.2.4. Message types

This document uses existing media types for the messages as specified by [RFC2585], [RFC5967], and CMC [RFC5272]. To support distribution of multiple certificates for the CA certificate chain, the [RFC2046] multipart/mixed media type is used.

The message type is specified in the HTTP Content-Type header with a media type. The use herein is consistent with [RFC5273].

For reference the messages and their corresponding media types are:
### Figure 5

<table>
<thead>
<tr>
<th>Message type</th>
<th>Request media type</th>
<th>Response media type</th>
<th>Source(s) of types</th>
<th>Request section</th>
<th>Response section</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA certificate request</td>
<td>N/A</td>
<td>application/pkcs7-mime</td>
<td>This RFC</td>
<td>Section 4.3</td>
<td>Section 4.3.1</td>
</tr>
<tr>
<td></td>
<td>[RFC5751]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cert enroll/renew/rekey</td>
<td>application/pkcs10</td>
<td>application/pkcs7-mime</td>
<td>[RFC5967] [RFC5751]</td>
<td>Section 4.4/4.4.1</td>
<td>Section 4.4.2</td>
</tr>
<tr>
<td></td>
<td>application/pkcs7-mime</td>
<td></td>
<td>[RFC5751]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full CMC</td>
<td>application/pkcs7-mime</td>
<td>application/pkcs7-mime</td>
<td>[RFC5751]</td>
<td>Section 4.5.1</td>
<td>Section 4.5.2</td>
</tr>
<tr>
<td></td>
<td>application/pkcs7-mime</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server-side Key Generation</td>
<td>application/pkcs10</td>
<td>multipart/mixed</td>
<td>[RFC5967] [RFC5751]</td>
<td>Section 4.6.1</td>
<td>Section 4.6.2</td>
</tr>
<tr>
<td></td>
<td>multipart/mixed</td>
<td>(application/pkcs7-mime &amp;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>application/pkcs8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[RFC5967] [RFC5751]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request CSR attributes</td>
<td>N/A</td>
<td>application/csrattrs</td>
<td>[RFC5751]</td>
<td>Section 4.7.1</td>
<td>Section 4.7.2</td>
</tr>
<tr>
<td></td>
<td>application/csrattrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This RFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 3.3. TLS Layer Design

TLS provides communications security for the layers above it. The integrity and authentication services it provides are leveraged to supply proof-of-identity and to allow authorization decisions to be made. The higher layer EST server and EST client are responsible for ensuring that an acceptable cipher suite is used and that bidirectional authentication has been established. Alternately, certificate-less TLS authentication—where neither the client nor server present a certificate—is also an acceptable method for EST server and client authentication.

When the EST server uses a certificate for authentication, TLS client authentication is the preferred method for identifying EST clients. If the EST client does not yet have a suitable client certificate the EST server can request HTTP Basic or Digest authentication protected by the TLS encryption. Alternately, certificate-less TLS authentication is also an acceptable method for EST server and client authentication.
authentication—where neither the client nor server present a certificate—is also an acceptable method for EST client authentication.

TLS channel binding information may be optionally inserted into a certificate request as detailed in Section 3.5 in order to provide the EST server with assurance that the authenticated TLS client entity has possession of the private key for the certificate being requested.

3.3.1. TLS for transport security

HTTPS [RFC2818] and specifies how HTTP messages are carried over TLS. HTTPS MUST be used. TLS 1.1 [RFC4346] (or later) SHOULD be supported. TLS session resumption [RFC5077] SHOULD be supported.

3.3.1.1. TLS-Based Server Authentication

The EST client authenticates the EST server as appropriate for the cipher suite negotiated. The following provides details assuming the TLS 1.1 [RFC4346] Section 9 Mandatory Cipher Suite TLS_RSA_WITH_3DES_EDE_CBC_SHA with a TLS server certificate presented during the TLS 1.1 [RFC4346] (or later) exchange-defined Server Certificate message. As an alternative to authentication using a certificate, an EST client MAY support certificate-less TLS authentication (Section 3.3.1.3).

Certificate validation MUST be performed as given in [RFC5280] and [RFC6125]. The EST server certificate MUST conform to the [RFC5280] certificate profile.

The client validates the TLS server certificate using local TAs, which may be in the form of certificates. If certificate verification fails the client MAY follow the procedure outlined in Section 4.3.2 for bootstrap distribution of CA certificates.

The EST client MUST perform authorization checks as specified in Section 4.1.

3.3.1.2. TLS-Based Client Authentication

The EST server MUST authenticate the EST client as appropriate for the cipher suite negotiated. The following provides details assuming the TLS 1.1 [RFC4346] Section 9 Mandatory Cipher Suite TLS_RSA_WITH_3DES_EDE_CBC_SHA with a TLS client certificate presented during the TLS 1.1 [RFC4346] (or later) exchange-defined Client certificate message. As an alternative to authentication using a certificate, an EST server MAY support certificate-less TLS authentication.
3.3.1.3. Certificate-less TLS Mutual Authentication

The client and server MAY negotiate a certificate-less cipher suite for mutual authentication. When using certificate-less mutual authentication in TLS for enrollment, the cipher suite MUST be resistant to dictionary attack and MUST provide sufficient information to perform the authorization checks. For example if the cipher suite uses a pre-shared secret, provisioned in an out-of-band fashion, as a credential to perform mutual authentication then knowledge of the pre-shared secret implies authorization as a peer in the exchange.

3.4. Proof-of-Possession

As defined in Section 2.1 of CMC [RFC5272], Proof-of-posses­sion (POP) "refers to a value that can be used to prove that the private key corresponding to the public key is in the possession and can be used by an end-entity."

The signed enrollment request provides a "Signature"-based proof-of-pos­session. The mechanism described in Section 3.5 strengthens this by optionally including "Direct"-based proof-of-pos­session by including TLS session specific information within the data covered by the enrollment request signature (thus linking the enrollment request to the authenticated end-point of the TLS connection).

3.5. Linking Identity and POP information

This specification provides an OPTIONAL method of linking identity and proof-of-pos­session by including information specific to the current authenticated TLS session within the signed certification request. Clients MAY use this method as a result of client configuration. If configuration is not possible the client can determine that this method is required by parsing the error responses.
or by examining the CSR Attributes Response (see Section 4.7.2).

Linking identity and proof-of-possession proves to the server that the authenticated TLS client has possession of the private key associated with the certification request and that the client was able to sign the certification request after the TLS session was established. This is an alternative to the [RFC5272] Section 6.3-defined "Linking Identity and POP information" method available if Full PKI messages are used.

The client generating the request obtains the tls-unique value as defined in Channel Bindings for TLS [RFC5929] from the TLS subsystem. The tls-unique specification includes a synchronization issue as described in Channel Bindings for TLS [RFC5929] section 3.1. To avoid this problem EST implementations MUST use the tls-unique value from the first TLS handshake. EST clients and servers use their tls-unique implementation specific synchronization methods to obtain this first tls-unique value. TLS "secure_renegotiation" [RFC5746] MUST be used. This maintains the binding from the first tls-unique value across renegotiations to the most recently negotiated connection.

The tls-unique value is Base 64 encoded as specified in Section 4 of [RFC4648] and the resulting string is placed in the certification request challenge-password field ([RFC2985], Section 5.4.1). If tls-unique information is not embedded within the certification request the challenge-password field MUST be empty to indicate that the client did not include the optional channel-binding information (any value submitted is verified by the server as tls-unique information).

The EST server MUST verify the tls-unique information embedded within the certification request according to server policy regarding the authenticated client. If the EST server forwards the request to back-end infrastructure for processing it is RECOMMENDED that the results of this verification be communicated. (For example this communication might use the CMC "RA POP Witness Control" in a CMC Full PKI Request message or the back-end infrastructure might authenticate the EST server as being a trusted infrastructure element that does not forward invalid requests. A detailed discussion of back-end processing is out of scope).

When rejecting requests the EST server response is as described for all enroll responses (Section 4.4.3). If a Full PKI Response is included the CMCFailInfo MUST be set to popFailed. If a human readable reject message is included it SHOULD include an informative text message indicating that linking of identity and POP information is required.
4.  Protocol Exchange Details

Before processing a request, an EST server determines if the client is authorized to receive the requested services. Likewise, the client determines if it will accept services from the EST server. These authorization decisions are described in the next two sections. Assuming that both sides of the exchange are authorized, then the actual operations are as described in the sections that follow.

4.1.  Server Authorization

The client MUST check the EST server authorization before accepting any server responses or responding to HTTP authentication requests.

When the server authenticates with a certificate the client MUST check the URI "against the server’s identity as presented in the server’s Certificate message" (HTTP Over TLS Section 3.1 "Server Identity" [RFC2818] and [RFC6125]). The provisioned URI provides the authorization statement and the server’s authenticated identity confirms it is the authorized server. Successful authentication using a certificate-less cipher suite implies authorization of the server.

If the URI does not match the server identity check then the TLS server certificate MUST contain the id-kp-cmcRA [CMC RFC6402] extended key usage extension and the TLS server certificate MUST be issued by the CA the EST server is providing services for.

The client MUST maintain the distinction between the EST specific TA for the CA issuing certificates and the TAs for third party CAs in order to make this determination (see, Section 3).

If these checks fail then authorization of the EST server does not occur except for as specified in Section 4.3.2.

4.2.  Client Authorization

When the EST server receives a Simple PKI Request or rekey/renew message, the decision to issue a certificate is always the CA’s. The EST server configuration reflects the CA policy and can use any data it wishes in determining whether to issue the certificate (e.g. CSR attributes, client identity, linking of client identity and proof-of-possession, etc). The details are out-of-scope. EST provides the EST server access to client’s authenticated identity-- e.g. the TLS client’s certificate in addition to any HTTP user authentication credentials-- to help in implementing configured policy.

If the client’s authenticated certificate was issued by the EST
server CA and includes the id-kp-cmcRA [RFC6402] extended key usage extension then the CA SHOULD apply policy consistent with a client that is acting as an RA (such as policy to support enrollment requests initiated either by the RA itself or by clients that are in communication with the RA).

4.3. Distribution of CA certificates

The EST client can request a copy of the current CA certificates and this function is generally performed before other EST functions.

4.3.1. Distribution of CA certificates request

EST clients MAY request TA information of the CA (in the form of certificates) with an HTTPS GET message with an operation path of "/CACerts". EST clients and servers MUST support the /CACerts function. Clients SHOULD request an up-to-date response before stored information has expired in order to maintain continuity of trust.

The EST server SHOULD NOT require client authentication or authorization to reply to this request.

The client MUST authenticate the EST server as specified in Section 3.3.1 and check the server’s authorization as given in Section 4.1 or follow the procedure outlined in Section 4.3.2.

4.3.2. Bootstrap Distribution of CA certificates

If the TLS authentication or authorization fails then the client MAY provisionally continue the TLS handshake to completion for the purposes of accessing the /CACerts or /fullCMC method. If the EST client continues with an unauthenticated connection the EST client MUST extract the HTTP content data from the response (Section 4.3.3 or Section 4.5.2) and engage the end-user to authorize the CA certificate using out-of-band pre-configuration data such as a CA certificate "fingerprint" (e.g., a SHA-1, SHA-256, SHA-512 [SHS], or MD5 [RFC1321] hash on the whole CA certificate). In a /fullCMC response it is the Publish Trust Anchors control within the Full PKI Response that must be accepted manually. It is incumbent on the end-user to properly verify the fingerprint or to provide valid out-of-band data necessary to verify the fingerprint.

HTTP authentication requests MUST NOT be responded to since the server is unauthenticated. The EST client uses the /CACerts response to build the trust anchor for subsequent TLS server authentication. EST clients MUST NOT make any other protocol exchange until after the /CACerts response has been accepted and a new TLS session
4.3.3. Distribution of CA certificates response

The EST server responds to the client HTTPS GET request with an HTTP GET response that includes CA trust anchor information, in the form of certificates within the Simple PKI Response. If the certificates are successfully returned, the server response MUST have an HTTP 200 response code with a content-type of "application/pkcs7-mime". Any other response code indicates an error and the client should abort the protocol.

The EST server MUST include the current CA certificate in the response. The EST server MUST include any additional certificates the client would need to build a chain to the current root CA certificate. For example if the EST server is configured to use a subordinate CA when signing new client requests then the appropriate subordinate CA certificates to chain to the root CA are included in this response.

If support for the CMP root certificate update mechanism is provided by the CA then the server MUST include the three "Root CA Key Update" certificates OldWithOld, OldWithNew, and NewWithOld. These are defined in Section 4.4 of CMP [RFC4210].

The client can always find the current TA in the form of a self-signed certificate by examining the received certificates. The CA’s most recent self signed certificate (e.g. NewWithNew certificate) is self-signed and has the latest NotAfter date.

The most recent CA certificate is the certificate that is extracted and authorized using out-of-band information as described in Section 4.3.2. After out-of-band validation occurs each of the other certificates MUST be validated using normal [RFC5280] certificate path validation (using the most recent CA certificate as the TA) before they can be used to build certificate paths during certificate validation.

The response format is the CMC Simple PKI Response as defined in [RFC5272]. The HTTP content-type of "application/pkcs7-mime" is used. The Simple PKI response is Base64 encoded, as specified in Section 4 of [RFC4648], and sandwched between headers:
4.4. Client Certificate Request Functions

EST clients MAY request a certificate from the EST server with an HTTPS POST using the operation path value of "/simpleEnroll". The EST server MUST support the /simpleEnroll function. EST clients MAY request a renew/rekey of existing certificates with an HTTP POST using the operation path value of "/simpleReEnroll". The EST server SHOULD support the /simpleReEnroll function.

The client is RECOMMENDED to have obtained the current CA certificates using Section 4.3 before performing certificate request functions to ensure it can validate the EST server certificate. The client MUST authenticate the EST server as specified in Section 3.3.1.1. The client MUST authorize the EST server as specified in Section 4.1.

The server MUST check client authentication as specified in Section 3.3.1.2. The server MUST check client authorization as specified in Section 4.2. The EST server MUST check the tls-unique value as described in Section 3.5.

The server MAY accept the certificate request for manual authorization by the administrator. (Section 4.4.3 describes the use of an HTTP 202 response to the EST client if this occurs).

4.4.1. Simple Enrollment of Clients

When HTTPS POSTing to /simpleEnroll the client MUST include a Simple PKI Request as specified in CMC Section 3.1 (i.e., a PKCS#10 Certification Request).

The Certification Request signature provides proof-of-possession of the private key to the EST server. If the requested KeyUsage extensions support digital signing operations then the certification request signature MUST be generated using the private key corresponding to the public key in the CertificationRequestInfo. If the requested KeyUsage extensions do not allow for digital signing operations the request MAY sign the certificate request, however the private key MUST NOT be used to perform signature operations after certificate issuance. The use of /fullCMC operations provides access to more advanced proof-of-possession methods that SHOULD be used when
the keys are not available for digital signing operations. This is consistent with the recommendations concerning submission of proof-of-possession to an RA or CA as described in [SP-800-57-Part-1].

The HTTP content-type of "application/pkcs10" is used. The format of the message is as specified in Section 6.4 of [RFC4945].

The client MAY request an additional certificate even when using an existing certificate in the TLS client authentication. For example the client can use an existing signature certificate to request a key exchange certificate.

4.4.2. Simple Re-Enrollment of Clients

EST clients renew/rekey certificates with an HTTPS POST using the operation path value of "/simpleReEnroll". EST clients and server MUST support the /simpleReEnroll function.

The certificate request is the same format as for the "simpleEnroll" request with the same HTTP content-type. The request Subject and SubjectAltName field(s) MUST contain the identity of the certificate being renewed/rekeyed. The ChangeSubjectName attribute, as defined in [RFC6402], MAY be included in the certificate request.

If the public key information in the certification request is the same as the currently issued certificate the EST server performs a renew operation. If the public key information is different than the currently issued certificate then the EST server performs a rekey operation. The specifics of these operations are out of scope of this profile.

4.4.3. Simple Enroll and Re-Enroll Response

If the enrollment is successful, the server response MUST have an HTTP 200 response code with a content-type of "application/pkcs7-mime". The response data is a degenerate certs-only Simple PKI Response containing only the certificate issued. The Simple PKI response is Base64 encoded and sandwiched between headers:

-----BEGIN PKCS7-----
MIIBhDCB7gIBADB0BMQsQoQYVQQGEwJBDTEʒTUxSB0IhJzAlB1JzAwIBAgIeGly
Simplified example of Base64 encoding of CMC Simple PKI Response
ED8rf3UDF6HjloV3jBnpeXt4JjZI/B1mD9HMQqfVetb1e4iZgMUu0gEWpD
8J40hHvLho=
-----END PKCS7-----

When rejecting a request the server MUST specify either an HTTP 4xx/401 error, or an HTTP 5xx error. A PKI Response with an HTTP
content-type of "application/pkcs7-mime" (see Section 4.5.2) MAY be included in the response data for any error response. If the content-type is not set, the response data MUST be a plain text human-readable error message containing informative information concerning why the request was rejected (for example indicating that CSR attributes are incomplete). A client MAY elect not to parse a CMC error response in favor of a generic error message.

If the server responds with an HTTP [RFC2616] 202 this indicates that the request has been accepted for processing but that a response is not yet available. The server MUST include a Retry-After header as defined for HTTP 503 responses and MAY include informative human-readable content. The client MUST wait at least the specified 'retry-after' time before repeating the same request. The client repeats the initial enrollment request after the appropriate 'retry-after' interval has expired. The client SHOULD log or inform the end user of this event. The server is responsible for maintaining all state necessary to recognize and handle retry operations as the client is stateless in this regard (it simply sends the same request repeatedly until it receives a different response code).

All other return codes are handled as specified in HTTP [RFC2616].

If the EST client has not obtained the current CA certificates using Section 4.3 then it may not be able to validate the certificate received.

4.5. Full CMC

EST clients can also request a certificate from the EST server with an HTTPS POST using the operation path value of "/fullCMC". Support for the /fullCMC function is OPTIONAL.

The client SHOULD authenticate the server as specified in Server Authentication (Section 3.3.1.1). Bootstrap distribution of CA certificates is specified in Section 4.3.2.

The server SHOULD authenticate the client as specified in Section 3.3.1. The server MAY depend on CMC client authentication methods instead.

4.5.1. Full CMC Request

If the HTTP POST to /fullCMC is not a valid Full PKI Request, the server MUST reject the message. The HTTP content-type used is "application/pkcs7-mime", as specified in [RFC5273].
4.5.2. Full CMC Response

The server responds with the client’s newly issued certificate or provides an error response.

If the enrollment is successful the server response MUST have an HTTP 200 response code with a content-type of "application/pkcs7-mime" as specified in [RFC5273]. The response data includes either the Simple PKI Response or the Full PKI Response.

When rejecting a request the server MAY specify either an HTTP 4xx/401 error or an HTTP 5xx error. A CMC response with content-type of "application/pkcs7-mime" SHOULD be included in the response data for any CMC error response. The client parses the CMC response to determine the current status.

All other return codes are handled as specified in Section 4.4.3 or HTTP [RFC2616]. For example the client interprets a HTTP 404 or 501 response to indicate that this service is not implemented.

The Full PKI Response is Base64 encoded and sandwiched between headers:

-----BEGIN PKCS7-----
MIIBhDCB7gIBADBFMQswCQYDVQQGEwJBVTETMBEGA1UECBMKU29tZS1TdGF0ZTEh
Simplified example of Base64 encoding of CMC Full PKI Response
ED8rf3UDF6HjloiV3jBnpetx4JjZH/BlmD9HMQofVERyble4iZgMvuIgwEjQwpD
8J40hVhLho=
-----END PKCS7-----

4.6. Server-side Key Generation

[[EDNOTE: This section includes references [draft-ietf-pkix-cmc-serverkeygeneration-00] which has not yet been published.]]

EST clients request a "private" key and associated certificate from the EST server with an HTTPS POST using the operation path value of "/serverKeyGen". Support for the /serverKeyGen function is OPTIONAL.

The client MUST authenticate the server as specified in Section 3.3.1.1. The EST client is RECOMMENDED to have obtained the current CA certificates using Section 4.3 to ensure it can validate the EST server certificate.

The EST server MUST authenticate the client as specified in Section 3.3.1. The server SHOULD use TLS-Based Client Authentication for authorization purposes. The EST server applies whatever
authorization or policy logic it chooses to determine if the "private" key and certificate should be generated.

Proper random number and key generation [RFC4086] as well as storage is a server implementation responsibility. The key pair and certificate are transferred over the TLS session; the EST server MUST verify that the current cipher suite is acceptable for securing the key data.

4.6.1. Server-side Key Generation Request

The certificate request is HTTPS POSTed and is the same format as for the "/simpleEnroll" and "/simpleReEnroll" path extensions with the same content-type.

The Subject and SubjectAltName field(s) or ChangeSubjectName attribute in the request MAY, as can all fields in a CSR, be ignored by the server as these are only requests. The server uses these fields, along with the authenticated client identity and server policy, to determine if it wishes to generate a new "private" key when servicing the request or re-use an escrowed "private" key. The client MAY request multiple keys and certificates.

In all respects the server SHOULD treat the request as it would any enroll or re-enroll request; with the only distinction being that the server MUST ignore the public key values of the certificate request and the request signature. These are included in the request only to allow re-use of existing codebases for generating and parsing such requests.

4.6.2. Server-side Key Generation Response

If the request is successful the server response MUST have an HTTP 200 response code with a content-type of "multipart/mixed" consisting of two parts. One part is the "private" key data and the other part is the certificate data.

The "private" key data MAY be an "application/pkcs8" consisting of the Base64 encoded DER-encoded PrivatekeyInfo sandwiched between the headers as described in [RFC5958]. Alternatively the "private" key data SHOULD be an "application/pkcs7-mime" containing a CMS [RFC5652] message (also as described in [RFC5958]). The content of this message is an EncryptedData or EnvelopedData content type containing the binary DER-encoded PrivatekeyInfo. The RecipientInfo MAY use any valid key management technique as determined by server policy and authenticated client identity. For example when the client uses a TLS client certificate for authentication the server can use this as the KeyTransRecipientInfo rid. The use of a CMS provides security to
the AsymmetricKeyPackage:

-----BEGIN PRIVATE KEY-----
MIIBhDCB7gIBADBFMQswCQYDVQQGEwJBVTETMBEGA1UECBMKEkU29tZS1TdGF0ZTEh
Simplified example of Base64 encoding of DER-encoded PrivateKeyInfo
ED8rf3UDF6HjloiV3jBnpetx4JjZH/B1mD9HMqofvEnryb1e4iZgMUv+EjWpD
8J40hHvLhlo=
-----END PRIVATE KEY-----

The certificate data part is an "application/pkcs7-mime" and exactly matches the certificate response to /simpleEnroll. If both parts are "application/pkcs7-mime" the client checks each (one will be a certsoonly Simple PKI response and the other will be the CMS message with the encrypted data).

When rejecting a request the server MUST specify either an HTTP 4xx/401 error, or an HTTP 5xx error. If the content-type is not set the response data MUST be a plain text human-readable error message.

Future work might define additional certification request attributes to communicate key management information in addition to using the client’s authenticated identity. Such attributes are out-of-scope of this document.

4.7. CSR Attributes

The CA MAY want to include client-provided attributes in certificates that it issues and some of these attributes may describe information that is not available to the CA. For this reason, the EST client MAY request a set of attributes from the EST server to include in its certification request.

4.7.1. CSR Attributes Request

The EST Client MAY request a list of CA-desired CSR attributes from the CA by sending an HTTPS GET message to the EST server with an operations path of "/CSRAttrs". Clients SHOULD request such a list if they have no a priori knowledge of what attributes are desired by the CA in an enrollment request or when dictated by policy.

4.7.2. CSR Attributes Response

If policy for the authenticated EST client indicates a CSR Attributes Response will be provided the server response MUST have an HTTP 200 response code. An HTTP response code of 204 or 404 indicates that a CSR Attributes Response is not available. Regardless of the response code the EST server and CA MAY reject any subsequent enrollment requests for any reason, including incomplete CSR attributes in the
Requests to attribute request messages MUST be encoded as content type "application/csrattrs". The syntax for application/csrattrs body is as follows:

\[
\text{Csrattrs ::= SEQUENCE SIZE (0..MAX) OF OBJECT IDENTIFIER { }}
\]

Servers include zero or more object identifiers that they wish the client to include in their certification request. When the server encodes Csrattrs as an empty SEQUENCE it means that the server has no specific additional attributes it wants in the client certification requests (this is functionally equivalent to an HTTP response code of 204 or 404). The sequence is DER (preferred) or BER encoded and then base64 encoded (section 4 of [RFC4648]). The resulting text forms the application/csrattr body, without headers.

For example, if a CA wishes the authenticated client to submit a certification request containing the MAC address [RFC2397] of a device and the challengePassword (indicating that Linking of Identity and POP information is requested, see Section 3.5) it takes the following object identifiers:

- macAddress: 1.3.6.1.1.1.1.22
- challengePassword: 1.2.840.113549.1.9.7

and encodes them into an ASN.1 SEQUENCE to produce:

\[
30 14 06 07 2B 06 01 01 01 01 16 06 09 2A 86 48 86 F7 0D 01 09 07
\]

and then base64 encodes the resulting ASN.1 SEQUENCE to produce:

\[
MBQGBysGAQEBARYGCSqGSIb3DQEJBw==
\]

The EST client parses the response OID’s and handles each OID independently on a best effort basis. When an OID indicates a known CSR attribute type the client SHOULD include that CSR attribute in the subsequent CSR submitted, either in the CSR attributes or in any other appropriate CSR field. When an OID is of an unknown type the OID MAY be ignored by the client.

5. IANA Considerations

(This section is incomplete)

IANA is requested to register the following:
IANA SHALL update the Application Media Types registry with the following filled-in template from [RFC4288].

The media subtype for Attributes in a CertificationRequest is application/csrattrs.

Type name: application
Subtype name: csrattrs
Required parameters: None
Optional parameters: None
Encoding considerations: binary;
Security Considerations:

Clients request a list of attributes that servers wish to be in certification requests. The request/response SHOULD be done in a TLS-protected tunnel.

Interoperability considerations: None
Published specification: This memo.
Applications which use this media type:
Enrollment over Secure Transport (EST)
Additional information:

Magic number(s): None
File extension: None
Macintosh File Type Code(s): None
Person & email address to contact for further information:
Dan Harkins <dharkins@arubanetworks.com>
Restrictions on usage: None
Author: Dan Harkins <dharkins@arubanetworks.com>
Intended usage: COMMON
6. Security Considerations

Support for Basic authentication as specified in HTTP [RFC2617] allows the server access to the client’s cleartext password. This provides integration with legacy username/password databases but requires exposing the plaintext password to the EST server. Use of a PIN or one-time-password can help mitigate concerns but EST clients are RECOMMENDED to use such credentials only once to obtain an appropriate client certificate to be used during future interactions with the EST server.

When the client uses a third party trust anchor database for certificate validation (see Section 3) then authorization proceeds as specified in Section 4.1. In this situation the client has validated the server as being a valid responder for the URI configured but can not directly verify that the responder is authorized as an RA within the to-be-enrolled-in PKI hierarchy. Possible avenues for an attack could be an erroneous URI injected into the client via an initial configuration method, or the server could have compromised a third party trust anchor to obtain an apparently valid server certificate. Clients using a third party trust anchor database are RECOMMENDED to only use TLS-based client authentication (to prevent leaking HTTP-based Client Authentication information). Such clients are RECOMMENDED to include "Linking Identity and POP information" (Section 3.5) in requests (to minimize the chance that such requests could be proxied to the real EST server). Additionally it is RECOMMENDED that the third party trust anchor database available for EST server authentication be carefully constructed (to reduce the risk of improperly managed third party CAs).

When using a certificate-less TLS cipher suite, the shared secret used for authentication and authorization MUST be known only to the two parties to the exchange-- the client and the server. Any sharing of secrets completely voids the security afforded by a certificate-less cipher suite. Exposure of a shared secret used by a certificate-less cipher suite to a third party enables client impersonation that can results in corruption of a client’s trust anchor database.

Any certificate-less TLS cipher suite used with EST MUST be resistant to dictionary attack. This means that the advantage an adversary gains through attack MUST be related to interaction and not computation. Certificate-less TLS cipher suites used with EST MUST also be based on a zero knowledge protocol to enable proof of knowledge of the shared secret without exposure of the shared secret.
(or any derived data which can be used to determine the secret). These requirements mean that the adversary gains advantage solely through active attack and the only thing learned from each active attack is whether a single guess of the secret is successful or not. Implementations of EST that support certificate-less TLS cipher suites SHOULD provide countermeasures-- for example, exponential back off after failed attempts or locking of an account after a certain number of unsuccessful attempts-- to mitigate repeated active attacks.

As described in CMC Section 6.7, "For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair". The inclusion of tls-unique within the certification request links the proof-of-possession to the TLS proof-of-identity by enforcing that the POP operation occurred while the TLS session is active. This strongly implies to the server that it is the authenticated client that has possession of the private key. If client authentication indicates a client with specific known behaviour this implication is strengthened but not proven.

The server-side key generation method allows keys to be transported over the TLS connection to the client. The distribution of "private" key material is inherently risky and servers are NOT RECOMMENDED to support this operation by default. Clients are NOT RECOMMENDED to request this service unless there is a compelling operational benefit. Use of a third party trust anchor database is NOT RECOMMENDED for server-side key generation. The use of an encrypted CMS Server-side Key Generation Response is RECOMMENDED.

Regarding the CSR attributes that the CA may list for inclusion in an enrollment request, there are no real inherent security issues with the content being conveyed but an adversary who is able to interpose herself into the conversation could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want.

7. References

7.1. Normative References


[RFC6402] Schaad, J., "Certificate Management over CMS (CMC)"
7.2. Informative References


Appendix A. Operational Scenario Example Messages

This section expands on the Operational Scenario Overviews by providing detailed examples of the messages at each TLS layer.

A.1. Obtaining CA Certificates

The following is an example of a valid /CACerts exchange.

During the initial TLS handshake the client can ignore the optional server generated "certificate request" and can instead proceed with the HTTP GET request:

```
GET /CACerts HTTP/1.1
User-Agent: curl/7.24.0 (i686-pc-linux-gnu) libcurl/7.24.0 OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5
Host: 127.0.0.1:8085
Accept: */*
```

In response the server provides the current CA certificate:

```
<= Recv header, 38 bytes (0x26)
Content-Type: application/pkcs7-mime
== Info: no chunk, no close, no size. Assume close to signal end
<= Recv header, 2 bytes (0x2)
<= Recv data, 1111 bytes (0x457)
-----BEGIN PKCS7-----
MIIDEQYJKoZIhvcNAQcCoIIDAjCCAv4CAQExADALBgkqhkiG9w0BBwGgkiktMiIE4DCCAcigAwIBAgIjAo...

Y6C6Y0f5M5r51EdkHg271xtSaoRr16D.fnVVEQGCj7QrrsUgfXFYVv6cCWkfrmCCA...

-----END PKCS7-----
```
A.2. Previously Installed Signature Certificate

The following is an example of a valid /simpleEnroll exchange. During this exchange the EST client uses an existing certificate issued by a trusted 3rd party PKI to obtain an initial certificate from the EST server.

During the initial TLS handshake the server generated "certificate request" includes both the distinguished name of the CA the EST server provides services for ("estExampleCA") and it includes the distinguished name of a trusted 3rd party CA ("estEXTERNALCA"):

```
0d 00 00 3d 03 01 02 40 00 37 00 1a 30 18 31 16 ...=...@.7..0.1.
30 14 06 03 55 04 03 13 0d 65 73 74 45 58 45 0...U....estEXTE
52 4e 41 4c 43 41 00 19 30 17 31 15 30 13 06 03 RNALCA..0.1.0...
55 04 03 13 0c 65 73 74 45 78 61 6d 70 6c 65 43 U....estExampleC
41
```

Which decodes as:

Acceptable client certificate CA names
/CN=estEXTERNALCA
/CN=estExampleCA

The EST client provides a certificate issued by "estEXTERNALCA" in the certificate response and the TLS handshake proceeds to completion. The EST server accepts the EST client certificate for authentication and accepts the EST client’s POSTed certificate request:
POST /simpleEnroll HTTP/1.1
User-Agent: curl/7.24.0 (i686-pc-linux-gnu) libcurl/7.24.0 OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5
Host: 127.0.0.1:8085
Accept: */*
Content-Type: application/pkcs10
Content-Length: 952

=> Send data, 952 bytes (0x3b8)

-----BEGIN CERTIFICATE REQUEST-----
MIICChjCCAw4CAQAwQTElMCMGA1UEAxMccmVxIGJ5IGNsaWVudCBpbiBhZ2VtYWlvdGVkIHN0ZXMgNjEYMBYGA1UEBRMPEQYfMCMGCSqGSIb3DQEBBQUAMCggMCAwDAYGhgBhskahsACgEB
-----END CERTIFICATE REQUEST-----.

The EST server uses the trusted 3rd party CA issued certificate to perform additional authorization and issues a certificate to the client:
A.3. Username/Password Distributed Out-of-Band

The following is an example of a valid /simpleEnroll exchange. During this exchange the EST client uses an out-of-band distributed username/password to authenticate itself to the EST server.

During the initial TLS handshake the client can ignore the optional server generated "certificate request" and can instead proceed with the HTTP POST request:
POST /simpleEnroll HTTP/1.1
User-Agent: curl/7.24.0 (i686-pc-linux-gnu) libcurl/7.24.0 OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5
Host: 127.0.0.1:8085
Accept: */*
Content-Type: application/pkcs10
Content-Length: 952

=> Send data, 952 bytes (0x3b8)
-----BEGIN CERTIFICATE REQUEST-----.

The EST server accepts this request but since a client certificate
was not provided for authentication/authorization the EST server
responds with the WWW-authenticate header:

<= Recv header, 27 bytes (0x1b)
HTTP/1.1 401 Unauthorized
<= Recv header, 75 bytes (0x4b)
WWW-Authenticate: Digest qop="auth", realm="estrealm", nonce="1341227174"

The EST client repeats the request, this time including the requested
Authorization header:
== Info: SSL connection using AES256-SHA
== Info: Server certificate:
  subject: CN=127.0.0.1
  start date: 2012-07-04 18:39:27 GMT
  expire date: 2013-07-04 18:39:27 GMT
  common name: 127.0.0.1 (matched)
== Info: SSL certificate verify ok.
== Info: Server auth using Digest with user 'estuser'

=> Send header, 416 bytes (0x1a0)
POST /simpleEnroll HTTP/1.1
Authorization: Digest username="estuser", realm="estrealm", nonc="1341427174", uri="/simpleEnroll", cnonce="ODc0OTk2", nc=00000001, qop="auth", resp=48a2b671c6b596adefef039e134b7d5f
User-Agent: curl/7.24.0 (i686-pc-linux-gnu) libcurl/7.24.0 OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5
Host: 127.0.0.1:8085
Accept: */*
Content-Type: application/pkcs10
Content-Length: 952

=> Send data, 952 bytes (0x3b8)
-----BEGIN CERTIFICATE REQUEST-----MIICcCAW4CAQAwQTElMCMGA1UE
AxMccmVxIGJ5IENsAWVucCBpcBkZ2lvZH0M.ZXAgMjEyMBYGA1UEBmpwYlMwMjExMjEx
pD3GdCFCBTtjyMYlBmdgkgkGjdhG0BATQEF.AAOCAQ8AMIIBCgKCAQEA91Xz9
Mowul0x0W5v17K1sNy7mAgmkz/wZDImBDXez.QZCb81r0iTD3tI0NH2xkpY3b
uqFjdtQzCMANLytWTR1sC56j/J/EMJ1Scro/ZA.Mg835RXJTP878N/jSNW7Eszxb
/zK50zKoRbZ4HqZm4NDapMFMcB4jgBdFx0PAgeR.++TKv1+9v1xsvdKIs5H4Msp
O2Wo1hPw5B5Cdu5z16e6Acih7ZpZd2cpPfz6C3HG0G1.0f+/0BzksFSqWsmUomyJy
WCflCuX9gsciCLNLxw0gcMprdTxLxjc18z03ZmBCQg.0.qs5/mUK/tv9R2k8+WuP3a
kzTU1kEtEcS6FVL13d+Tw1DAQABoAAwDQYJKoZIhvcN.AQEFBQADgEBALc7Eutu
R/qQgEaG1s8mDzU3lfFQ/uYgj/NxzzZpZSszVLGMASv9a.pNzawdfqFDs+ZZ+gAQ
QkVcXjdbqY3pAFdC+EeWk+KnuAUIP6Ku3ZBPVdwBxU/Ie7.FlekQ7ILkFKrH5cHRu
2/bPsummaryLRVfMNx3wPqQxqc3mQjIBGATJM5kuHndYFGj.XdfrlGRPyOowG/Xf
QrKBB3tzbPbC+ycoWUAJFPO0/86RuvjF9H+yvoM182vlg8O.FyEeAA/FMPi13aEcT
B12zBfx4e7FlwGlhbgI7/6KOnF93xGd7JYPHasbcWszdD0Z.oYiM+44g0QOhh1j
OWprXcnnngSSutRILaw.-----END CERTIFICATE REQUEST-----.

The EST server uses the username/password to perform authentication/
authorization and responds with the issued certificate:
A.4.  Re-Enrollment

The following is an example of a valid /simpleReEnroll exchange. During this exchange the EST client authenticates itself using an existing certificate issued by the CA the EST server provides services for.

Initially this exchange is identical to enrollment using an externally issued certificate for client authentication since the server is not yet aware of the client’s intention. As in that example the EST server the generated "certificate request" includes both the distinguished name of the CA the EST server provides services for ("estExampleCA") and it includes the distinguished name of a trusted 3rd party CA ("estEXTERNALCA")

================================================================================================

In text format this is:

Acceptable client certificate CA names
/CN=estEXTERNALCA
/CN=estExampleCA

The EST client provides a certificate issued by "estExampleCA" in the certificate response and the TLS handshake proceeds to completion. The EST server accepts the EST client certificate for authentication and accepts the EST client’s POSTed certificate request.

The rest of the protocol traffic is effectively identical to a normal enrollment.

A.5. Server Key Generation

The following is an example of a valid /serverKeyGen exchange. During this exchange the EST client authenticates itself using an existing certificate issued by the CA the EST server provides services for.

The initial TLS handshake is identical to the enrollment example handshake. The HTTP POSTed message is:
POST /serverKeyGen HTTP/1.1
User-Agent: curl/7.24.0 (i686-pc-linux-gnu) libcurl/7.24.0 OpenSSL/0.9.8b zlib/0.6.5
Host: 127.0.0.1:8085
Accept: */*
Content-Type: application/pkcs10
Content-Length: 968

=> Send data, 968 bytes (0x3c8)
-----BEGIN CERTIFICATE REQUEST-----.
MIICkzCCAXsCAQAwTjEyMDAGA1UEAxMpc2VymVySV5rZ2VuIjJlcSBieSBjG11.bmQgaW4gZGVtbyBzdGV2U3VxGDA
WBgNVBAUDTlBJRDPxawRnZXgU046NTCCASIw.DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMnlUlq0ag/fDAVlgn0jEzIy9i9vU26l2Ow.9cH48KmKQ9Q2ZeWdnlJlpo/ib2PLjJNj9o8NNKm9Xq55BqIVi.vkIB+rtMLeYRVULORuBAqX
FmZkKxRcUcVFEYIv4iQW/hSG2JKpArht959aT81yaP.24b2OvuG19dSxtTX3j37K
x4g31XUVs0sp62avfA37y19L8KbKhR1emybUo294.QeLb/VYPI0+Eathv/igiX
1DqH1UcZ2p5SyAUXwZPatFboNVwErVOR3MjwVECAwEA.aaAMACGcSGbISd3QEB
BUA4IBQaqHezK5/tvbXieho/ABTYV0914114NM+WA.wjCns2uyaJycspq1YKk
rial+qgtXFl0fe+5ukAj56t7HnoiI4EO9r8jgCw/fY6i7LYh.JDCx0hWoAsAvH/Vizi
jkhJNhDBiGM7Gd5f/2VJLEPQpmWN05L5f+204eQc/QweYc.BAmfoG6y48b/ZH09/9T
PeaeQpjsjou1/100uU68kU3F МосXMyt1Va0AQjxZ1+5k.EiEJo+1tXsQwdrpOH
asoTNBN+j3K18omJQ90e91X8oV6kMWHYUt0nXD0Yt6S/B9c.AE6GTAHDA/xySpVA
cqlWA+FXjxwEMxV/intVHMuwy031fDZ/TVuCPX.-----END CERTIFICATE REQUEST-----.

After processing the request the EST server response is:
<= Recv header, 17 bytes (0x11)
HTTP/1.1 200 OK
<= Recv header, 16 bytes (0x10)
Status: 200 OK
<= Recv header, 67 bytes (0x43)
Content-Type: multipart/mixed ; boundary=estServerExampleBoundary
<= Info: no chunk, no close, no size. Assume close to signal end
<= Recv data, 2 bytes (0x02)

This is the preamble. It is to be ignored, though it is a handy place
to include an explanatory note including contact or support information...-estServerExampleBoundary.Content-Type=application/pkcs8.--------BEGIN PRIVATE KEY--------.
Bgqkgk9i9G60BQAEPAASCBCwgcgJjAgEAo1BAOQCO78117tr10yi.1.5b92Zydh8ze
izXrjMPF/Rxoz2C9U2THCrlPZlXgQMe/zivc38m8/BMKkUc+DsSM.tzxn4l+9tI
svDkAE4Fyzn0nhId/zawgj6kU0c13mxz3bn2rWdRYAmM5w41imDV3h1v.1aMKUDSjHv
bq+2/GlW1TRX3iW5CMYbK+1pJXPTJz/GuWr/+/Efqwz2LwGcj8Dx.1gbx9vG0
mfTlXvM4UTX28KBlaLjbaluIOx3cZbEyaSPerdzggyVFHgGAh1GFU8.DQ1QEk
nn66GPMt1SNgitxFxWouFsqsax5MWh/i52FeaF2PNTH0uzKtilweJhk.g0GMIQ
HTTP/1.1 200 OK
Status: 200 OK
Content-Type: multipart/mixed ; boundary=estServerExampleBoundary

In text format this is:

HTTP/1.1 200 OK
Status: 200 OK
Content-Type: multipart/mixed ; boundary=estServerExampleBoundary

This is the preamble. It is to be ignored, though it is a handy place for estServer to include an explanatory note including contact or support information.

---estServerExampleBoundary

Content-Type: application/pkcs8

-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCO78117tri0yi1m
Mb9Z2yCh8zeizXrjMPF/Rxozc91U2THcrPGXQOmne/zivce0m8/BNkUkUc+DsSM
 tzxn4l+9tIsVSDkAe4FyxN0Ld/zawgj6kUc13mxZnb2r2WaRYAm5w41ImIV3blv
 aMUKDSJhVbq+z/G1W1TRx3iWi5CMHYb+pJPXTJzj/Gwlr/b/+Efqzwz2l1wGcj4Dx
 Igbx9vG0mfI1xM4UX2K8BbaLgbalaitsu0x3C2bEyaSPerdzqgvXFHGGAhglFU8
 DqiEKeinn66GPMtm1SNgitxfXwouFpqpsax5Mwn/i52TeFaeF2PNThOuzKtilweJhk
g0gMIQTXAgMBAAEcgEAN1rZ8XNK/lXBELlxKH0H830a4YyqKZKF2kUN8hU33pxu
 Y/0scVbLu46Wzys0ffvryUC+ZFJnMBCCOPjGhI/4NKWkEqc9TAKiz+wDo+hfsbf0
 yFrEmihkkH8R3fKvni619ld0wIynqCvF9pV7tGrSmJCly6Ckwbk7TEBuOUGLOwtp
 x106X1hksA0vwh+75q0wSnhl75/10LIVCvLbuaVhodTH1h5M4daFbgyRW7UtUikt
 Q3hdw1rpyVZg82BbFp0j4f9GDrqgs885KMCDu3t9bnn/v1kEte+PxhuywW+d0o
 hkKSEX0ynLcKzqM5tijpsPq0UvzPBKLiACuNFAi+a4AQKBqGud6VH2eYe0TjPpyA
 v0JnNWP7oMzyJ4/eFgDe9m+2Aj0/mQaMY95ftZ+GpEkqgyC625DFevEmgH4Sg2+G
 gFD93diyrRScvne8SmPxxLP0uYVmiCqZsLDNE18B3buxAm8GJ219NEnzOec
 jpMOV/ICGlAzTqQsS13zz0/OgqKBqGUB41o3P3Ebgjt/+d1kLHwU8c7o3UyLKs
 VxVsyQwiyT8xMeCwuoPFPNFCOjW53KN/YSpCvtpptKGSTbmlMKYkgasEqcql
 Vb50Pf/A/jNAP3mdAgCzbn6IF1N0vQe2dclo5puz0G03HDwq7EtqsI9QOJSMc3YC
 QNC0OorPtVwKBqGCHCafayWYdhA/l+/+G2U9x6Yd56if43rCbnV+2EQvCQa9z4Xc
 w4A+HBs0mdl7TunL6MOemzXkKr/SP/7P7KixHYHnpMQoLhavQ2V4w5TQhe9kD7
 wu8aXRv9bJ4g0CuDNO6/jWpm/KenXNN/Fkaya3Vg4zdVbMzBjJdYqczyqKBqFXs
 zBszGgwz1/F7aAdDK49m1wPhyoeBb00qHwbX/L17rZ1Wmef+n5F9Ju/U77B7/J
 UpdO9vGgGSOOnRk0LIRP2s50U5IqGT2TVLvf8a1UmbVgl/KX511Yi5yMztEwRcjEX
 VM9ejXeN01N7i0zPq/z/RXO3K12yLh4T09/E8WjjAgAAlmqUV4hN4fFyf1PdMy
 fpvoWekiiR3E3iEBZNVyYsh7uxwn760pqVifkz2DSrZeeYm4+lw9JWmWtU0zpPzg
 CjYMoGlB46nHlZrbzjB55tW0LV6GrMkK/cF0xPNNzCtSopaQ86HhHq7rRdhXSau/bY
 EKc9t9nNfJFzZxdBgd+f4k=
-----END PRIVATE KEY-----

---estServerExampleBoundary

---estServerExampleBoundary

Content-Type: application/pkcs7-mime

-----BEGIN PKCS7-----
MIIDAYJKoZIhvcNAQcCoIIDLCTCAykCAQExADALBgkqhkiG9w0BBwGgMQPfMID
CzCCAfOAsgwIAgIBTABgkgkli9G0w0BAQFADAMR6u/wEwYDVQDExwlc3RFegFt
cGx1IQWEhVncHNTnwlNA0MTgzOTM2W5hCnMTMnWzA0MTgtz0TM2WJAsMswKAXYVQQD
EyF3zX2ZXXZjx2R1tL1eSbnZW51cmFOZ2QgcmVzcG9uc2VggcmEmAIAGGccgGSi553
DQEBAQUAA4IBDwAgUwggEKAoIBAQCO78117tri0yi1mMB9Z2yCh8zeizXrjMPFRxoz
2C91U2THcrPGXQOmne/zivce0m8/BNkUkUc+DsSMtzxn4l+9tIsVSDkAe4FyxN0Ld/zawgj6kUc13mxZnb2r2WaRYAm5w41ImIV3blv
aMUKDSJhVbq+z/G1W1TRx3iWi5CMHYb+pJPXTJzj/Gwlr/b/+Efqzwz2l1wGcj4Dx
Igbx9vG0mfI1xM4UX2K8BbaLgbalaitsu0x3C2bEyaSPerdzqgvXFHGGAhglFU8DqiEKeinn66GPMtm1SNgitxfXwouFpqpsax5Mwn/i52TeFaeF2PNThOuzKtilweJhkzg0gMIQTXAgMBAAEgEAN1rZ8XNK/lXBELlxKH0H830a4YyqKZKF2kUN8hU33pxuY/0scVbLu46Wzys0ffvryUC+ZFJnMBCCOPjGhI/4NKWkEqc9TAKiz+wDo+hfsbf0 yFrEmihkkH8R3fKvni619ld0wIynqCvF9pV7tGrSmJCly6Ckwbk7TEBuOUGLOwtpx106X1hksA0vwh+75q0wSnhl75/10LIVCvLbuaVhodTH1h5M4daFbgyRW7UtUiktQ3hdw1rpyVZg82BbFp0j4f9GDrqgs885KMCDu3t9bnn/v1kEte+PxhuywW+d0ohkKSEX0ynLcKzqM5tijpsPq0UvzPBKLiACuNFAi+a4AQKBqGud6VH2eYe0TjPpyAv0JmNWP7oMzyJ4/eFgDe9m+2Aj0/mQaMY95ftZ+GpEkqgyC625DFevEmgH4Sg2+GgFD93diyrRScvne8SmPxxLP0uYVmiCqZsLDNE18B3buxAm8GJ219NEnzOecjpMOV/ICGlAzTqQsS13zz0/OgqKBqGUB41o3P3Ebgjt/+d1kLHwU8c7o3UyLKsVxVsyQwiyT8xMeCwuoPFPNFCOjW53KN/YSpCvtpptKGSTbmlMKYkgasEqcqlVb50Pf/A/jNAP3mdAgCzbn6IF1N0vQe2dclo5puz0G03HDwq7EtqsI9QOJSMc3YCQNC0OorPtVwKBqGCHCafayWYdhA/l+/+G2U9x6Yd56if43rCbnV+2EQvCQa9z4Xcw4A+HBs0mdl7TunL6MOemzXkKr/SP/7P7KixHYHnpMQoLhavQ2V4w5TQhe9kD7wu8aXRv9bJ4g0CuDNO6/jWpm/KenXNN/Fkaya3Vg4zdVbMzBjJdYqczyqKBqFXszBszGgwz1/F7aAdDK49m1wPhyoeBb00qHwbX/L17rZ1Wmef+n5F9Ju/U77B7/JUpdO9vGgGSOOnRk0LIRP2s50U5IqGT2TVLvf8a1UmbVgl/KX511Yi5yMztEwRcjEXVM9ejXeN01N7i0zPq/z/RXO3K12yLh4T09/E8WjjAgAAlmqUV4hN4fFyf1PdMfpovoWekiiR3E3iEBZNVyYsh7uxwn760pqVifkz2DSrZeeYm4+lw9JWmWtU0zpPzgCjYMoGlB46nHlZrbzjB55tW0LV6GrMkK/cF0xPNNzCtSopaQ86HhHq7rRdhXSau/bYEKc9t9nNfJFzZxdBgd+f4k=
-----END PKCS7-----

This is the epilogue. It is also to be ignored.

A.6. CSR Attributes

The following is an example of a valid /CSRAttrs exchange. During this exchange the EST client authenticates itself using an existing certificate issued by the CA the EST server provides services for.

The initial TLS handshake is identical to the enrollment example handshake. The HTTP GET request:

```
GET /CSRAttrs HTTP/1.1
User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/SSL/zlib/1.2.3.4 libidn/1.23 librtmp/2.3
Host: 127.0.0.1:8085
Accept: */*
```

In response the server provides suggested attributes that are appropriate for the authenticated client:

```
<= Recv header, 36 bytes (0x24)
Content-Type: application/csrattrs
== info: no chunk, no close, no size. Assume close to signal end
<= Recv header, 2 bytes (0x2)
<= Recv data, 33 bytes (0x21)
0000: MBQGBysGAQEBARYGCSqGS1b3DQEJBw==
```

Authors’ Addresses

Max Pritikin (editor)
Cisco Systems, Inc.
510 McCarthy Drive
Milpitas, CA  95035
USA

Email: pritikin@cisco.com
Abstract

This document specifies a protocol useful in determining the current status of a digital certificate without requiring CRLs. Additional mechanisms addressing PKIX operational requirements are specified in separate documents. This document obsoletes RFC 2560 and RFC 6277.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ld-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice
Copyright (c) 2012 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1.  Introduction .............................................. 4
   1.1. .................................................. 4
2.  Protocol Overview ....................................... 6
   2.1  Request ........................................... 6
   2.2  Response .......................................... 6
   2.3  Exception Cases .................................... 8
   2.4  Semantics of thisUpdate, nextUpdate and producedAt ... 8
   2.5  Response Pre-production ............................ 10
   2.6  OCSP Signature Authority Delegation .................. 10
   2.7  CA Key Compromise ................................ 10
3.  Functional Requirements ................................ 10
   3.1  Certificate Content ................................. 10
   3.2  Signed Response Acceptance Requirements ........... 11
4.  Detailed Protocol ....................................... 11
   4.1  Requests ........................................... 11
       4.1.1  Request Syntax ................................. 11
       4.1.2  Notes on the Request Syntax ................... 12
   4.2  Response Syntax .................................... 13
       4.2.1  ASN.1 Specification of the OCSP Response .... 13
       4.2.2  Notes on OCSP Responses ....................... 16
           4.2.2.1  Time ..................................... 16
           4.2.2.2  Authorized Responders ..................... 16
         4.2.2.2.1 Revocation Checking of an Authorized
                   Responder ............................... 17
       4.2.2.3  Basic Response .............................. 18
   4.3  Mandatory and Optional Cryptographic Algorithms .... 19
   4.4  Extensions ......................................... 19
       4.4.1  Nonce ......................................... 19
       4.4.2  CRL References ............................... 19
       4.4.3  Acceptable Response Types .................... 20
       4.4.4  Archive Cutoff ................................ 20

Stefan Santesson Expires April 17, 2013 [Page 2]
4.4.5  CRL Entry Extensions ........................................... 21
4.4.6  Service Locator .................................................. 21
4.4.7  Preferred Signature Algorithms ............................... 21
   4.4.7.1 Extension Syntax ........................................... 22
   4.4.7.2 Responder Signature Algorithm Selection .................. 23
      4.4.7.2.1 Dynamic Response .................................. 23
      4.4.7.2.2 Static Response .................................... 24
5.  Security Considerations ........................................... 25
   5.1 Preferred Signature Algorithms ............................... 25
      5.1.1 Use of insecure algorithms .............................. 25
      5.1.2 Man in the Middle Downgrade Attack ...................... 26
      5.1.3 Denial of Service Attack ................................. 26
6  IANA Considerations .............................................. 27
7  References .......................................................... 27
   7.1.  Normative References ........................................ 27
   7.2.  Informative References ...................................... 27
7.  Acknowledgement .................................................. 29
Appendix A. .......................................................... 29
   A.1 OCSP over HTTP ............................................... 29
      A.1.1 Request .................................................. 29
      A.1.2 Response ................................................. 29
Appendix B.  ASN.1 Modules .......................................... 30
   B.1.  OCSP in ASN.1 ................................................ 30
   B.2.  Preferred Signature Algorithms ASN.1 ....................... 33
      B.2.1.  ASN.1 Module ........................................... 33
      B.2.2.  1988 ASN.1 Module .................................... 34
Appendix C. MIME registrations ....................................... 34
   C.2 application/ocsp-response .................................... 35
Authors’ Addresses .................................................. 38
1. Introduction

This document specifies a protocol useful in determining the current status of a digital certificate without requiring CRLs. Additional mechanisms addressing PKIX operational requirements are specified in separate documents.

This specification obsoletes [RFC2560] and [RFC6277]. The primary reason for the publication of this document is to address ambiguities that have been found since the publication of RFC 2560. This document differs from RFC 2560 in only a few areas:

- Section 4.4.1 specifies the ASN.1 syntax for the nonce extension, which was missing in RFC 2560.
- Section 4.4.7 specifies a new extension that may be included in a request message to specify signature algorithms the client would prefer the server use to sign the response as specified in [RFC6277].
- Section 2.3 extends the use of the "unauthorized" error response, as specified in [RFC5019].
- Section 4.2.1 and 4.2.2.3 states that a response may include revocation status information for certificates that were not included in the request, as permitted in [RFC5019].
- Section 4.3 changes set of cryptographic algorithms that clients must support and the set of cryptographic algorithms that clients should support as specified in [RFC6277].
- Section 4.2.2.2 has been updated to clarify when a responder is considered an Authorized Responder.
- Section 4.2.2.3 clarify that the ResponderID field corresponds to the OCSP Responder signer certificate.

An overview of the protocol is provided in section 2. Functional requirements are specified in section 4. Details of the protocol are in section 5. We cover security issues with the protocol in section 6. Appendix A defines OCSP over HTTP, appendix B accumulates ASN.1 syntactic elements and appendix C specifies the mime types for the messages.

1.1. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document (in uppercase, as shown) are to be interpreted as described in [RFC2119].
2. Protocol Overview

In lieu of or as a supplement to checking against a periodic CRL, it may be necessary to obtain timely information regarding the revocation status of a certificate (cf. [RFC5280], Section 3.3). Examples include high-value funds transfer or large stock trades.

The Online Certificate Status Protocol (OCSP) enables applications to determine the (revocation) state of an identified certificate. OCSP may be used to satisfy some of the operational requirements of providing more timely revocation information than is possible with CRLs and may also be used to obtain additional status information. An OCSP client issues a status request to an OCSP responder and suspends acceptance of the certificate in question until the responder provides a response.

This protocol specifies the data that needs to be exchanged between an application checking the status of a certificate and the server providing that status.

2.1 Request

An OCSP request contains the following data:

-- protocol version
-- service request
-- target certificate identifier
-- optional extensions which MAY be processed by the OCSP Responder

Upon receipt of a request, an OCSP Responder determines if:

1. the message is well formed
2. the responder is configured to provide the requested service and
3. the request contains the information needed by the responder

If any one of the prior conditions are not met, the OCSP responder produces an error message; otherwise, it returns a definitive response.

2.2 Response

OCSP responses can be of various types. An OCSP response consists of a response type and the bytes of the actual response. There is one basic type of OCSP response that MUST be supported by all OCSP servers and clients. The rest of this section pertains only to this basic response type.
All definitive response messages SHALL be digitally signed. The key used to sign the response MUST belong to one of the following:

- the CA who issued the certificate in question
- a Trusted Responder whose public key is trusted by the requester
- a CA Designated Responder (Authorized Responder) who holds a specially marked certificate issued directly by the CA, indicating that the responder may issue OCSP responses for that CA

A definitive response message is composed of:

- version of the response syntax
- name of the responder
- responses for each of the certificates in a request
- optional extensions
- signature algorithm OID
- signature computed across hash of the response

The response for each of the certificates in a request consists of

- target certificate identifier
- certificate status value
- response validity interval
- optional extensions

This specification defines the following definitive response indicators for use in the certificate status value:

- good
- revoked
- unknown

The "good" state indicates a positive response to the status inquiry. At a minimum, this positive response indicates that the certificate is not revoked, but does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the responder regarding the status of the certificate such as positive statement about issuance, validity, etc.

The "revoked" state indicates that the certificate has been revoked (either permanently or temporarily (on hold)). This state SHOULD also be returned if the responder knows that the requested certificate has never been issued. Note that the receiver of a response may have to consult out-of-band knowledge, or information in an extension defined outside of this standard, to know whether a particular responder has knowledge about whether a requested certificate has been issued or
not and whether it responds "good" or "revoked" to a status request for a non-issued certificate.

The "unknown" state indicates that the responder doesn’t know about the certificate being requested.

2.3 Exception Cases

In case of errors, the OCSP Responder may return an error message. These messages are not signed. Errors can be of the following types:

- malformedRequest
- internalError
- tryLater
- sigRequired
- unauthorized

A server produces the "malformedRequest" response if the request received does not conform to the OCSP syntax.

The response "internalError" indicates that the OCSP responder reached an inconsistent internal state. The query should be retried, potentially with another responder.

In the event that the OCSP responder is operational, but unable to return a status for the requested certificate, the "tryLater" response can be used to indicate that the service exists, but is temporarily unable to respond.

The response "sigRequired" is returned in cases where the server requires the client sign the request in order to construct a response.

The response "unauthorized" is returned in cases where the client is not authorized to make this query to this server or the server is not capable of responding authoritatively (cf. [RFC5019], Section 2.2.3).

2.4 Semantics of thisUpdate, nextUpdate and producedAt

Responses can contain three times in them - thisUpdate, nextUpdate and producedAt. The semantics of these fields are:

- thisUpdate: The time at which the status being indicated is known to be correct
- nextUpdate: The time at or before which newer information will be available about the status of the certificate
- producedAt: The time at which the OCSP responder signed this response.
If nextUpdate is not set, the responder is indicating that newer revocation information is available all the time.
2.5 Response Pre-production

OCSP responders MAY pre-produce signed responses specifying the status of certificates at a specified time. The time at which the status was known to be correct SHALL be reflected in the thisUpdate field of the response. The time at or before which newer information will be available is reflected in the nextUpdate field, while the time at which the response was produced will appear in the producedAt field of the response.

2.6 OCSP Signature Authority Delegation

The key that signs a certificate’s status information need not be the same key that signed the certificate. A certificate’s issuer explicitly delegates OCSP signing authority by issuing a certificate containing a unique value for extendedKeyUsage in the OCSP signer’s certificate. This certificate MUST be issued directly to the responder by the cognizant CA.

2.7 CA Key Compromise

If an OCSP responder knows that a particular CA’s private key has been compromised, it MAY return the revoked state for all certificates issued by that CA.

3. Functional Requirements

3.1 Certificate Content

In order to convey to OCSP clients a well-known point of information access, CAs SHALL provide the capability to include the AuthorityInfoAccess extension (defined in [RFC5280], section 4.2.2.1) in certificates that can be checked using OCSP. Alternatively, the accessLocation for the OCSP provider may be configured locally at the OCSP client.

CAs that support an OCSP service, either hosted locally or provided by an Authorized Responder, MUST provide for the inclusion of a value for a uniformResourceIndicator (URI) accessLocation and the OID value id-ad-ocsp for the accessMethod in the AccessDescription SEQUENCE.

The value of the accessLocation field in the subject certificate defines the transport (e.g. HTTP) used to access the OCSP responder and may contain other transport dependent information (e.g. a URL).
3.2 Signed Response Acceptance Requirements

Prior to accepting a signed response as valid, OCSP clients SHALL confirm that:

1. The certificate identified in a received response corresponds to that which was identified in the corresponding request;
2. The signature on the response is valid;
3. The identity of the signer matches the intended recipient of the request.
4. The signer is currently authorized to sign the response.
5. The time at which the status being indicated is known to be correct (thisUpdate) is sufficiently recent.
6. When available, the time at or before which newer information will be available about the status of the certificate (nextUpdate) is greater than the current time.

4. Detailed Protocol

The ASN.1 syntax imports terms defined in [RFC5280]. For signature calculation, the data to be signed is encoded using the ASN.1 distinguished encoding rules (DER) [X.690].

ASN.1 EXPLICIT tagging is used as a default unless specified otherwise.

The terms imported from elsewhere are: Extensions, CertificateSerialNumber, SubjectPublicKeyInfo, Name, AlgorithmIdentifier, CRLReason

4.1 Requests

This section specifies the ASN.1 specification for a confirmation request. The actual formatting of the message could vary depending on the transport mechanism used (HTTP, SMTP, LDAP, etc.).

4.1.1 Request Syntax

\[
\text{OCSPRequest} ::= \text{SEQUENCE} \{
\text{tbsRequest} \text{TBSRequest},
\text{optionalSignature} [0] \text{EXPLICIT Signature OPTIONAL }
\}\n\]

\[
\text{TBSRequest} ::= \text{SEQUENCE} \{
\}
\]
version             [0]     EXPLICIT Version DEFAULT v1,
requestorName       [1]     EXPLICIT GeneralName OPTIONAL,
requestList                 SEQUENCE OF Request,
requestExtensions   [2]     EXPLICIT Extensions OPTIONAL }

Signature       ::=     SEQUENCE { 
signatureAlgorithm      AlgorithmIdentifier, 
signature               BIT STRING, 
certs               [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL}

Version         ::=             INTEGER  {  v1(0) }

Request         ::=     SEQUENCE { 
reqCert                     CertID, 
singleRequestExtensions     [0] EXPLICIT Extensions OPTIONAL } 

CertID          ::=     SEQUENCE { 
    hashAlgorithm       AlgorithmIdentifier, 
    issuerNameHash      OCTET STRING, -- Hash of Issuer’s DN 
    issuerKeyHash       OCTET STRING, -- Hash of Issuers public key 
    serialNumber        CertificateSerialNumber } 

issuerNameHash is the hash of the Issuer’s distinguished name. The hash shall be calculated over the DER encoding of the issuer’s name field in the certificate being checked. issuerKeyHash is the hash of the Issuer’s public key. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the issuer’s certificate. The hash algorithm used for both these hashes, is identified in hashAlgorithm. serialNumber is the serial number of the certificate for which status is being requested.

4.1.2 Notes on the Request Syntax

The primary reason to use the hash of the CA’s public key in addition to the hash of the CA’s name, to identify the issuer, is that it is possible that two CAs may choose to use the same Name (uniqueness in the Name is a recommendation that cannot be enforced). Two CAs will never, however, have the same public key unless the CAs either explicitly decided to share their private key, or the key of one of the CAs was compromised.

Support for any specific extension is OPTIONAL. The critical flag SHOULD NOT be set for any of them. Section 4.4 suggests several useful extensions. Additional extensions MAY be defined in additional RFCs. Unrecognized extensions MUST be ignored (unless they have the critical flag set and are not understood).
The requestor MAY choose to sign the OCSP request. In that case, the signature is computed over the tbsRequest structure. If the request is signed, the requestor SHALL specify its name in the requestorName field. Also, for signed requests, the requestor MAY include certificates that help the OCSP responder verify the requestor’s signature in the certs field of Signature.

4.2 Response Syntax

This section specifies the ASN.1 specification for a confirmation response. The actual formatting of the message could vary depending on the transport mechanism used (HTTP, SMTP, LDAP, etc.).

4.2.1 ASN.1 Specification of the OCSP Response

An OCSP response at a minimum consists of a responseStatus field indicating the processing status of the prior request. If the value of responseStatus is one of the error conditions, responseBytes are not set.

OCSPResponse ::= SEQUENCE {
    responseStatus         OCSPResponseStatus,
    responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
    successful            (0),  --Response has valid confirmations
    malformedRequest      (1),  --Illegal confirmation request
    internalError         (2),  --Internal error in issuer
    tryLater              (3),  --Try again later
    --(4) is not used
    sigRequired           (5),  --Must sign the request
    unauthorized          (6)   --Request unauthorized
}

The value for responseBytes consists of an OBJECT IDENTIFIER and a response syntax identified by that OID encoded as an OCTET STRING.

ResponseBytes ::=  SEQUENCE {
    responseType   OBJECT IDENTIFIER,
    response       OCTET STRING }

For a basic OCSP responder, responseType will be id-pkix-ocsp-basic.

id-pkix-ocsp OBJECT IDENTIFIER ::= { id-ad-ocsp }

id-pkix-ocsp-basic OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
OCSP responders SHALL be capable of producing responses of the id-pkix-ocsp-basic response type. Correspondingly, OCSP clients SHALL be capable of receiving and processing responses of the id-pkix-ocsp-basic response type.

The value for response SHALL be the DER encoding of BasicOCSPResponse.

BasicOCSPResponse ::= SEQUENCE {
  tbsResponseData  ResponseData,
  signatureAlgorithm  AlgorithmIdentifier,
  signature  BIT STRING,
  certs  [0]  EXPLICIT SEQUENCE OF Certificate OPTIONAL
}

The value for signature SHALL be computed on the hash of the DER encoding ResponseData. The responder MAY include certificates in the certs field of BasicOCSPResponse that help the OCSP client verify the responder’s signature. If no certificates are included then certs SHOULD be absent.

ResponseData ::= SEQUENCE {
  version  [0]  EXPLICIT Version DEFAULT v1,
  responderID  ResponderID,
  producedAt  GeneralizedTime,
  responses  SEQUENCE OF SingleResponse,
  responseExtensions  [1]  EXPLICIT Extensions OPTIONAL
}

ResponderID ::= CHOICE {
  byName  [1]  Name,
  byKey  [2]  KeyHash
}

KeyHash ::= OCTET STRING -- SHA-1 hash of responder’s public key (excluding the tag and length fields)

SingleResponse ::= SEQUENCE {
  certID  CertID,
  certStatus  CertStatus,
  thisUpdate  GeneralizedTime,
  nextUpdate  [0]  EXPLICIT GeneralizedTime OPTIONAL,
  singleExtensions  [1]  EXPLICIT Extensions OPTIONAL
}

CertStatus ::= CHOICE {
  good  [0]  IMPLICIT NULL,
  revoked  [1]  IMPLICIT RevokedInfo,
  unknown  [2]  IMPLICIT UnknownInfo
}

RevokedInfo ::= SEQUENCE {
  revocationTime  GeneralizedTime,
revocationReason [0] EXPLICIT CRLReason OPTIONAL }

UnknownInfo ::= NULL
4.2.2 Notes on OCSP Responses

4.2.2.1 Time

The thisUpdate and nextUpdate fields define a recommended validity interval. This interval corresponds to the \{thisUpdate, nextUpdate\} interval in CRLs. Responses whose nextUpdate value is earlier than the local system time value SHOULD be considered unreliable. Responses whose thisUpdate time is later than the local system time SHOULD be considered unreliable. Responses where the nextUpdate value is not set are equivalent to a CRL with no time for nextUpdate (see Section 2.4).

The producedAt time is the time at which this response was signed.

4.2.2.2 Authorized Responders

The key that signs a certificate’s status information need not be the same key that signed the certificate. It is necessary however to ensure that the entity signing this information is authorized to do so. Therefore, a certificate’s issuer MAY either sign the OCSP responses itself or it MAY explicitly designate this authority to another entity. OCSP signing delegation SHALL be designated by the inclusion of id-kp-OCSPSigning in an extendedKeyUsage certificate extension included in the OCSP response signer’s certificate. This certificate MUST be issued directly by the CA that issued the certificate in question.

The CA SHOULD use the same issuing key to issue a delegation certificate as was used to sign the certificate being checked for revocation. Systems relying on OCSP responses MUST recognize a delegation certificate as being issued by the CA that issued the certificate in question only if the delegation certificate and the certificate being checked for revocation was signed by the same key.

Note: CA key rollover is not prohibited when issuing a certificate for an authorized responder for backwards compatibility with RFC 2560 [RFC2560]. That is, it is not prohibited to issue a certificate for an authorized responder using a different issuing key than the key used to issued the certificate being checked for revocation. However, such practice is strongly discouraged since clients are not required to recognize a responder with such certificate as an authorized responder.

{id-kp-OCSPSigning OBJECT IDENTIFIER ::= {id-kp 9}}

Systems or applications that rely on OCSP responses MUST be capable of detecting and enforcing use of the id-ad-ocspSigning value as
They MAY provide a means of locally configuring one or more OCSP signing authorities, and specifying the set of CAs for which each signing authority is trusted. They MUST reject the response if the certificate required to validate the signature on the response fails to meet at least one of the following criteria:

1. Matches a local configuration of OCSP signing authority for the certificate in question; or

2. Is the certificate of the CA that issued the certificate in question; or

3. Includes a value of id-ad-ocspSigning in an ExtendedKeyUsage extension and is issued by the CA that issued the certificate in question as stated above."

Additional acceptance or rejection criteria may apply to either the response itself or to the certificate used to validate the signature on the response.

4.2.2.2.1 Revocation Checking of an Authorized Responder

Since an Authorized OCSP responder provides status information for one or more CAs, OCSP clients need to know how to check that an authorized responder’s certificate has not been revoked. CAs may choose to deal with this problem in one of three ways:

- A CA may specify that an OCSP client can trust a responder for the lifetime of the responder’s certificate. The CA does so by including the extension id-pkix-ocsp-nocheck. This SHOULD be a non-critical extension. The value of the extension SHALL be NULL. CAs issuing such a certificate should realize that a compromise of the responder’s key is as serious as the compromise of a CA key used to sign CRLs, at least for the validity period of this certificate. CAs may choose to issue this type of certificate with a very short lifetime and renew it frequently.

  id-pkix-ocsp-nocheck OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }

- A CA may specify how the responder’s certificate be checked for revocation. This can be done using CRL Distribution Points if the check should be done using CRLs or CRL Distribution Points, or Authority Information Access if the check should be done in some other way. Details for specifying either of these two mechanisms are available in [RFC5280].

- A CA may choose not to specify any method of revocation checking for the responder’s certificate, in which case, it would be up to the
OCSP client’s local security policy to decide whether that certificate should be checked for revocation or not.

4.2.2.3 Basic Response

The basic response type contains:

- the version of the response syntax, which MUST be v1 (value is 0) for this version of the basic response syntax;
- either the name of the responder or a hash of the responder’s public key as the ResponderID;
- the time at which the response was generated;
- responses for each of the certificates in a request;
- optional extensions;
- a signature computed across a hash of the response; and
- the signature algorithm OID.

The purpose of the ResponderID information is to allow clients to find the certificate used to sign a signed OCSP response. Therefore, the information MUST correspond to the certificate that was used to sign the response.

The responder MAY include certificates in the certs field of BasicOCSPResponse that help the OCSP client verify the responder’s signature.

The response for each of the certificates in a request consists of:

- an identifier of the certificate for which revocation status information is being provided (i.e., the target certificate);
- the revocation status of the certificate (good, revoked, or unknown);
- the validity interval of the response; and
- optional extensions.

The response MUST include a SingleResponse for each certificate in the request and SHOULD NOT include any additional SingleResponse elements. OCSP responders that pre-generate status responses MAY return responses that include additional SingleResponse elements if
necessary to improve response pre-generation performance or cache efficiency. [Editor’s note: From Section 2.2.1 of RFC 5019.]

4.3 Mandatory and Optional Cryptographic Algorithms

Clients that request OCSP services SHALL be capable of processing responses signed using RSA with SHA-1 (identified by sha1WithRSAEncryption OID specified in [RFC3279]) and RSA with SHA-256 (identified by sha256WithRSAEncryption OID specified in [RFC4055]). Clients SHOULD also be capable of processing responses signed using DSA keys (identified by the id-dsa-with-sha1 OID specified in [RFC3279]). Clients MAY support other algorithms.

4.4 Extensions

This section defines some standard extensions, based on the extension model employed in X.509 version 3 certificates see [RFC5280]. Support for all extensions is optional for both clients and responders. For each extension, the definition indicates its syntax, processing performed by the OCSP Responder, and any extensions which are included in the corresponding response.

4.4.1 Nonce

The nonce cryptographically binds a request and a response to prevent replay attacks. The nonce is included as one of the requestExtensions in requests, while in responses it would be included as one of the responseExtensions. In both the request and the response, the nonce will be identified by the object identifier id-pkix-ocsp-nonce, while the extnValue is the value of the nonce.

    id-pkix-ocsp-nonce OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }

Nonce ::= OCTET STRING

4.4.2 CRL References

It may be desirable for the OCSP responder to indicate the CRL on which a revoked or onHold certificate is found. This can be useful where OCSP is used between repositories, and also as an auditing mechanism. The CRL may be specified by a URL (the URL at which the CRL is available), a number (CRL number) or a time (the time at which the relevant CRL was created). These extensions will be specified as singleExtensions. The identifier for this extension will be id-pkix-ocsp-crl, while the value will be CrlID.
id-pkix-ocsp-crl OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }

CrlID ::= SEQUENCE {
  crlUrl               [0]     EXPLICIT IA5String OPTIONAL,
  crlNum               [1]     EXPLICIT INTEGER OPTIONAL,

For the choice crlUrl, the IA5String will specify the URL at which the CRL is available. For crlNum, the INTEGER will specify the value of the CRL number extension of the relevant CRL. For crlTime, the GeneralizedTime will indicate the time at which the relevant CRL was issued.

4.4.3 Acceptable Response Types

An OCSP client MAY wish to specify the kinds of response types it understands. To do so, it SHOULD use an extension with the OID id-pkix-ocsp-response, and the value AcceptableResponses. This extension is included as one of the requestExtensions in requests. The OIDs included in AcceptableResponses are the OIDs of the various response types this client can accept (e.g., id-pkix-ocsp-basic).

id-pkix-ocsp-response OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

As noted in section 4.2.1, OCSP responders SHALL be capable of responding with responses of the id-pkix-ocsp-basic response type. Correspondingly, OCSP clients SHALL be capable of receiving and processing responses of the id-pkix-ocsp-basic response type.

4.4.4 Archive Cutoff

An OCSP responder MAY choose to retain revocation information beyond a certificate’s expiration. The date obtained by subtracting this retention interval value from the producedAt time in a response is defined as the certificate’s "archive cutoff" date.

OCSP-enabled applications would use an OCSP archive cutoff date to contribute to a proof that a digital signature was (or was not) reliable on the date it was produced even if the certificate needed to validate the signature has long since expired.

OCSP servers that provide support for such historical reference SHOULD include an archive cutoff date extension in responses. If included, this value SHALL be provided as an OCSP singleExtensions extension identified by id-pkix-ocsp-archive-cutoff and of syntax
GeneralizedTime.

id-pkix-ocsp-archive-cutoff OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }

ArchiveCutoff ::= GeneralizedTime

To illustrate, if a server is operated with a 7-year retention interval policy and status was produced at time $t_1$ then the value for ArchiveCutoff in the response would be $(t_1 - 7 \text{ years})$.

4.4.5 CRL Entry Extensions

All the extensions specified as CRL Entry Extensions - in Section 5.3 of [RFC5280] - are also supported as singleExtensions.

4.4.6 Service Locator

An OCSP server may be operated in a mode whereby the server receives a request and routes it to the OCSP server which is known to be authoritative for the identified certificate. The ServiceLocator request extension is defined for this purpose. This extension is included as one of the singleRequestExtensions in requests.

id-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }

ServiceLocator ::= SEQUENCE {
  issuer   Name,
  locator  AuthorityInfoAccessSyntax OPTIONAL }

Values for these fields are obtained from the corresponding fields in the subject certificate.

4.4.7 Preferred Signature Algorithms

Since algorithms other than the mandatory to implement algorithms are allowed, and since a client currently has no mechanism to indicate it’s algorithm preferences, there is always a risk that a server choosing a non-mandatory algorithm, will generate a response that the client may not support.

While an OCSP responder may apply rules for algorithm selection, e.g., using the signature algorithm employed by the CA for signing CRLs and certificates, such rules may fail in common situations:

- The algorithm used to sign the CRLs and certificates may not be consistent with key pair being used by the OCSP responder to sign responses.
o A request for an unknown certificate provides no basis for a responder to select from among multiple algorithm options.

The last criterion cannot be resolved through the information available from in-band signaling using the RFC 2560 [RFC2560] protocol, without modifying the protocol.

In addition, an OCSP responder may wish to employ different signature algorithms than the one used by the CA to sign certificates and CRLs for several reasons:

o The responder may employ an algorithm for certificate status response that is less computationally demanding than for signing the certificate itself.

o An implementation may wish to guard against the possibility of a compromise resulting from a signature algorithm compromise by employing two separate signature algorithms.

This section describes:

o An extension that allows a client to indicate the set of preferred signature algorithms.

o Rules for signature algorithm selection that maximizes the probability of successful operation in the case that no supported preferred algorithm(s) are specified.

4.4.7.1 Extension Syntax

A client MAY declare a preferred set of algorithms in a request by including a preferred signature algorithms extension in requestExtensions of the OCSPRequest.

```
id-pkix-ocsp-pref-sig-algs OBJECT IDENTIFIER ::= { id-pkix-ocsp 8 }

PreferredSignatureAlgorithms ::= SEQUENCE OF
  PreferredSignatureAlgorithm

PreferredSignatureAlgorithm ::= SEQUENCE {
  sigIdentifier        AlgorithmIdentifier,
  publicKeyAlgIdentifier  SMIMECapability OPTIONAL
}
```

The syntax of AlgorithmIdentifier is defined in section 4.1.1.2 of...
RFC 5280 [RFC5280] The syntax of SMIMECapability is defined in RFC 5751 [RFC5751]

sigIdentifier specifies the signature algorithm the client prefers, e.g. algorithm=ecdsa-with-sha256. Parameters are absent for most common signature algorithms.

pubKeyAlgIdentifier specifies the subject public key algorithm identifier the client prefers in the server’s certificate used to validate the OCSP response. e.g. algorithm=id-ecPublicKey and parameters= secp256r1.

pubKeyAlgIdentifier is OPTIONAL and provides means to specify parameters necessary to distinguish among different usages of a particular algorithm, e.g. it may be used by the client to specify what curve it supports for a given elliptic curve algorithm.

The client MUST support each of the specified preferred signature algorithms and the client MUST specify the algorithms in the order of preference, from the most preferred to the least preferred.

Section 4.4.7.1 of this document describes how a server selects an algorithm for signing OCSP responses to the requesting client.

4.4.7.2 Responder Signature Algorithm Selection

RFC 2560 [RFC2560] did not specify a mechanism for deciding the signature algorithm to be used in an OCSP response. This does not provide a sufficient degree of certainty as to the algorithm selected to facilitate interoperability.

4.4.7.2.1 Dynamic Response

A responder MAY maximize the potential for ensuring interoperability by selecting a supported signature algorithm using the following order of precedence, as long as the selected algorithm meets all security requirements of the OCSP responder, where the first method has the highest precedence:

1. Select an algorithm specified as a preferred signing algorithm in the client request

2. Select the signing algorithm used to sign a certificate revocation list (CRL) issued by the certificate issuer providing status information for the certificate specified by CertID

3. Select the signing algorithm used to sign the OCSPRequest
4. Select a signature algorithm that has been advertised as being the default signature algorithm for the signing service using an out of band mechanism.

5. Select a mandatory or recommended signing algorithm specified for the version of the OCSP protocol in use.

A responder SHOULD always apply the lowest numbered selection mechanism that results in the selection of a known and supported algorithm that meets the responder’s criteria for cryptographic algorithm strength.

4.4.7.2.2 Static Response

For purposes of efficiency, an OCSP responder is permitted to generate static responses in advance of a request. The case may not permit the responder to make use of the client request data during the response generation, however the responder SHOULD still use the client request data during the selection of the pre-generated response to be returned. Responders MAY use the historical client requests as part of the input to the decisions of what different algorithms should be used to sign the pre-generated responses.
5. Security Considerations

For this service to be effective, certificate-using systems must connect to the certificate status service provider. In the event such a connection cannot be obtained, certificate-using systems could implement CRL processing logic as a fall-back position.

A denial of service vulnerability is evident with respect to a flood of queries. The production of a cryptographic signature significantly affects response generation cycle time, thereby exacerbating the situation. Unsigned error responses open up the protocol to another denial of service attack, where the attacker sends false error responses.

The use of precomputed responses allows replay attacks in which an old (good) response is replayed prior to its expiration date but after the certificate has been revoked. Deployments of OCSP should carefully evaluate the benefit of precomputed responses against the probability of a replay attack and the costs associated with its successful execution.

Requests do not contain the responder they are directed to. This allows an attacker to replay a request to any number of OCSP responders.

The reliance of HTTP caching in some deployment scenarios may result in unexpected results if intermediate servers are incorrectly configured or are known to possess cache management faults. Implementors are advised to take the reliability of HTTP cache mechanisms into account when deploying OCSP over HTTP.

5.1 Preferred Signature Algorithms

The mechanism used to choose the response signing algorithm MUST be considered to be sufficiently secure against cryptanalytic attack for the intended application.

In most applications it is sufficient for the signing algorithm to be at least as secure as the signing algorithm used to sign the original certificate whose status is being queried. This criteria may not hold in long term archival applications however in which the status of a certificate is being queried for a date in the distant past, long after the signing algorithm has ceased being considered trustworthy.

5.1.1 Use of insecure algorithms

It is not always possible for a responder to generate a response that
the client is expected to understand and that meets contemporary standards for cryptographic security. In such cases an OCSP responder operator MUST balance the risk of employing a compromised security solution and the cost of mandating an upgrade, including the risk that the alternative chosen by end users will offer even less security or no security.

In archival applications it is quite possible that an OCSP responder might be asked to report the validity of a certificate on a date in the distant past. Such a certificate might employ a signing method that is no longer considered acceptably secure. In such circumstances the responder MUST NOT generate a signature using a signing mechanism that is not considered acceptably secure.

A client MUST accept any signing algorithm in a response that it specified as a preferred signing algorithm in the request. It follows therefore that a client MUST NOT specify as a preferred signing algorithm any algorithm that is either not supported or not considered acceptably secure.

5.1.2 Man in the Middle Downgrade Attack

The mechanism to support client indication of preferred signature algorithms is not protected against a man in the middle downgrade attack. This constraint is not considered to be a significant security concern since the OCSP responder MUST NOT sign OCSP Responses using weak algorithms even if requested by the client. In addition, the client can reject OCSP responses that do not meet its own criteria for acceptable cryptographic security no matter what mechanism is used to determine the signing algorithm of the response.

5.1.3 Denial of Service Attack

Algorithm agility mechanisms defined in this document introduces a slightly increased attack surface for Denial-of-Service attacks where the client request is altered to require algorithms that are not supported by the server. Denial-of-Service considerations from RFC 4732 [RFC4732] are relevant for this document.
6  IANA Considerations

<IANA considerations text>

7.  References

7.1.  Normative References


7.2.  Informative References


7. Acknowledgement

TBD

Appendix A.

A.1 OCSP over HTTP

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

A.1.1 Request

HTTP based OCSP requests can use either the GET or the POST method to submit their requests. To enable HTTP caching, small requests (that after encoding are less than 255 bytes), MAY be submitted using GET. If HTTP caching is not important, or the request is greater than 255 bytes, the request SHOULD be submitted using POST. Where privacy is a requirement, OCSP transactions exchanged using HTTP MAY be protected using either TLS/SSL or some other lower layer protocol.

An OCSP request using the GET method is constructed as follows:

GET {url}/{url-encoding of base-64 encoding of the DER encoding of the OCSPRequest}

where {url} may be derived from the value of AuthorityInfoAccess or other local configuration of the OCSP client.

An OCSP request using the POST method is constructed as follows: The Content-Type header has the value "application/ocsp-request" while the body of the message is the binary value of the DER encoding of the OCSPRequest.

A.1.2 Response

An HTTP-based OCSP response is composed of the appropriate HTTP headers, followed by the binary value of the DER encoding of the OCSPResponse. The Content-Type header has the value "application/ocsp-response". The Content-Length header SHOULD specify the length of the response. Other HTTP headers MAY be present and MAY be ignored if not understood by the requestor.
Appendix B. ASN.1 Modules

This appendix includes the ASN.1 modules for OCSP. Appendix C.1 includes an ASN.1 module that conforms to the 1998 version of ASN.1 for all syntax elements of OCSP other than the preferred signature algorithms extension. An alternative to this module that conforms to the 2002 version of ASN.1 may be found in Section 4 of [RFC5912]. Appendix C.2 includes two ASN.1 modules for the preferred signature algorithms extension, one that conforms to the 1998 version of ASN.1 and one that conforms to the 2002 version of ASN.1.

B.1. OCSP in ASN.1

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

DEFINITIONS EXPLICIT TAGS ::= 

BEGIN 

IMPORTS

-- PKIX Certificate Extensions
AuthorityInfoAccessSyntax, CRLReason, GeneralName
FROM PKIX1Implicit88 { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit(19) }

Name, CertificateSerialNumber, Extensions,
id-kp, id-ad-ocsp, Certificate, AlgorithmIdentifier
FROM PKIX1Explicit88 { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit(18) };

OCSPRequest ::= SEQUENCE {
  tbsRequest              TBSRequest,
  optionalSignature   [0] EXPLICIT Signature OPTIONAL } 

TBSRequest ::= SEQUENCE {
  version             [0] EXPLICIT Version DEFAULT v1,
  requestorName       [1] EXPLICIT GeneralName OPTIONAL,
  requestList             SEQUENCE OF Request,
  requestExtensions   [2] EXPLICIT Extensions OPTIONAL } 

Signature ::= SEQUENCE {
  signatureAlgorithm      AlgorithmIdentifier,
  signature               BIT STRING,

Stefan Santesson          Expires April 17, 2013
certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

Version ::= INTEGER { v1(0) }

Request ::= SEQUENCE {
  reqCert CertID,
  singleRequestExtensions [0] EXPLICIT Extensions OPTIONAL }

CertID ::= SEQUENCE {
  hashAlgorithm AlgorithmIdentifier,
  issuerNameHash OCTET STRING, -- Hash of Issuer’s DN
  issuerKeyHash OCTET STRING, -- Hash of Issuers public key
  serialNumber CertificateSerialNumber }

OCSPResponse ::= SEQUENCE {
  responseStatus OCSPResponseStatus,
  responseBytes [0] EXPLICIT ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
  successful (0), -- Response has valid confirmations
  malformedRequest (1), -- Illegal confirmation request
  internalError (2), -- Internal error in issuer
  tryLater (3), -- Try again later
  sigRequired (5), -- Must sign the request
  unauthorized (6) -- Request unauthorized }

ResponseBytes ::= SEQUENCE {
  responseType OBJECT IDENTIFIER,
  response OCTET STRING }

BasicOCSPResponse ::= SEQUENCE {
  tbsResponseData ResponseData,
  signatureAlgorithm AlgorithmIdentifier,
  signature BIT STRING,
  certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
  version [0] EXPLICIT Version DEFAULT v1,
  responderID ResponderID,
  producedAt GeneralizedTime,
  responses SEQUENCE OF SingleResponse,
  responseExtensions [1] EXPLICIT Extensions OPTIONAL }

ResponderID ::= CHOICE {
  byName [1] Name,
  byKey [2] KeyHash }
KeyHash ::= OCTET STRING --SHA-1 hash of responder's public key
-- (i.e., the SHA-1 hash of the value of the
-- BIT STRING subjectPublicKey [excluding
-- the tag, length, and number of unused
-- bits] in the responder's certificate)

SingleResponse ::= SEQUENCE {
certID                  CertID,
certStatus              CertStatus,
thisUpdate              GeneralizedTime,
nextUpdate          [0] EXPLICIT GeneralizedTime OPTIONAL,
singleExtensions    [1] EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
good                [0] IMPLICIT NULL,
revoked             [1] IMPLICIT RevokedInfo,
unknown             [2] IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
revocationTime          GeneralizedTime,
revocationReason    [0] EXPLICIT CRLReason OPTIONAL }

UnknownInfo ::= NULL

ArchiveCutoff ::= GeneralizedTime

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

ServiceLocator ::= SEQUENCE {
issuer                  Name,
locator                 AuthorityInfoAccessSyntax }

-- Object Identifiers

id-kp-OCSPSigning          OBJECT IDENTIFIER ::= { id-kp 9 }
id-pkix-ocsp              OBJECT IDENTIFIER ::= { id-ad-ocsp }
id-pkix-ocsp-basic        OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
id-pkix-ocsp-nonce        OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
id-pkix-ocsp-crl          OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }
id-pkix-ocsp-response     OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }
id-pkix-ocsp-nocheck      OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }
id-pkix-ocsp-archive-cutoff OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }
id-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }

END
B.2. Preferred Signature Algorithms ASN.1

B.2.1. ASN.1 Module

OCSP-AGILITY-2009 { iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-ocsp-agility-2009-93(66) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

EXPORTS ALL; -- export all items from this module

IMPORTS

id-pkix-ocsp
FROM OCSP-2009 -- From [RFC5912]
{ iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-mod-ocsp-02(48) }

AlgorithmIdentifier{}, SIGNATURE-ALGORITHM, PUBLIC-KEY
FROM AlgorithmInformation-2009 -- From [RFC5912]
{ iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-algorithmInformation-02(58) }

EXTENSION
FROM PKIX-CommonTypes-2009 -- From [RFC5912]
{ iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-mod-pkixCommon-02(57) };

-- Add re-preferred-signature-algorithms to the set of extensions
-- for TBSRequest.requestExtensions

re-preferred-signature-algorithms EXTENSION ::= {
SYNTAX PreferredSignatureAlgorithms
IDENTIFIED BY id-pkix-ocsp-pref-sig-algs }

id-pkix-ocsp-pref-sig-algs OBJECT IDENTIFIER ::= { id-pkix-ocsp 8 }

PreferredSignatureAlgorithms ::= SEQUENCE OF PreferredSignatureAlgorithm

PreferredSignatureAlgorithm ::= SEQUENCE {
sigIdentifier AlgorithmIdentifier{SIGNATURE-ALGORITHM, {...}},
certIdentifier AlgorithmIdentifier{PUBLIC-KEY, {...}} OPTIONAL
}

END
B.2.2.  1988 ASN.1 Module

OCSP-AGILITY-88 { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-ocsp-agility-2009-88(67) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS ALL;
IMPORTS

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

AlgorithmIdentifier
FROM PKIX1Explicit88 { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit(18) };

id-pkix-ocsp-pref-sig-algs OBJECT IDENTIFIER ::= { id-pkix-ocsp 8 }

PreferredSignatureAlgorithms ::= SEQUENCE OF PreferredSignatureAlgorithm

PreferredSignatureAlgorithm ::= SEQUENCE {
  sigIdentifier   AlgorithmIdentifier,
  certIdentifier  AlgorithmIdentifier OPTIONAL }

END

Appendix C. MIME registrations

C.1 application/ocsp-request

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

MIME media type name: application
MIME subtype name: ocsp-request
Required parameters: None
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 Online Certificate Status Protocol - OCSP

Applications which use this media type: OCSP clients

Additional information:

Magic number(s): None
File extension(s): .ORQ
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/ocsp-response

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

MIME media type name: application

MIME subtype name: ocsp-response

Required parameters: None

Optional parameters: None
Encoding considerations: binary

Security considerations: Carries a cryptographically signed response

Interoperability considerations: None

Published specification: IETF PKIX Working Group Draft on Online Certificate Status Protocol - OCSP
Applications which use this media type: OCSP servers

Additional information:

Magic number(s): None
File extension(s): .ORS
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>
Authors’ Addresses

Stefan Santesson
3xA Security AB
Scheeleve. 17
223 70 Lund
Sweden
EMail: sts@aaa-sec.com

Michael Myers
TraceRoute Security
EMail: mmyers@fastq.com

Rich Ankney
EMail: no e-mail

Ambarish Malpani
Arcot Systems
EMail: ambarish@gmail.com

Slava Galperin
A9.com inc
130 Lytton Ave Suite 300
Palo Alto, California 94301
United States
EMail: slava.galperin@gmail.com

Carlisle Adams
University of Ottawa
75 Laurier Avenue East
Ottawa ON K1N 6N5
Canada
EMail: cadams@eecs.uottawa.ca