Internet Engineering Task Force (IETF)
Request for Comments: 8295
Category: Standards Track
ISSN: 2070-1721

S. Turner
sn3rd

January 2018

EST (Enrollment over Secure Transport) Extensions

Abstract

The EST (Enrollment over Secure Transport) protocol defines the Well-Known URI (Uniform Resource Identifier) -- /.well-known/est -- along with a number of other path components that clients use for PKI (Public Key Infrastructure) services, namely certificate enrollment (e.g., /simpleenroll). This document defines a number of other PKI services as additional path components -- specifically, firmware and trust anchors as well as symmetric, asymmetric, and encrypted keys. This document also specifies the PAL (Package Availability List), which is an XML (Extensible Markup Language) file or JSON (JavaScript Object Notation) object that clients use to retrieve packages available and authorized for them. This document extends the EST server path components to provide these additional services.

Status of This Memo

Table of Contents

1. Introduction

The EST (Enrollment over Secure Transport) protocol [RFC7030] defines the Well-Known URI (Uniform Resource Identifier) -- /.well-known/est -- to support selected services related to the PKI (Public Key Infrastructure), with such PCs (path components) as simple enrollment with /simpleenroll, rekey or renew with /simplereenroll, etc. A server that wishes to support additional PKI-related services and other security-related packages could use the same .well-known URI by defining additional PCs. This document defines six such PCs:

- /pal - The PAL (Package Availability List) provides a list of all known packages available and authorized for a client. By accessing the service provided by this PC first, the client can walk through the PAL and download all the packages necessary to begin operating securely. The PAL essentially points to other PCs, including the PCs defined in this document as well as those defined in [RFC7030] (e.g., /cacerts, /simpleenroll, /simplereenroll, /fullcmc, /serverkeygen, and /csrattrs). The /pal PC is described in Section 2.

- /eecerts - EE (End-Entity) certificates [RFC5280] are needed by the client when they invoke a security protocol for communicating with a peer (i.e., they become operational and do something meaningful as opposed to just communicating with the infrastructure). If the infrastructure knows the certificate(s) needed by the client, then providing the peer’s certificate avoids the client having to discover the peer’s certificate. This service is not meant to be a general-purpose repository to which clients query a "repository" and then get a response; this is purely a push mechanism. The /eecerts PC is described in Section 3.

- /crls - CRLs (Certificate Revocation Lists) and ARLs (Authority Revocation Lists) [RFC5280] are also needed by the client when they validate certificate paths. CRLs (and ARLs) from TAs (Trust Anchors) and intermediate CAs (Certification Authorities) are needed to validate the certificates used to generate the client’s certificate or the peer’s certificate, which is provided by the /eecerts PC, and providing them saves the client from having to "discover" them and then retrieve them. CRL "discovery" is greatly aided by the inclusion of the CRL Distribution Point certificate extension [RFC5280], but this extension is not always present in certificates and requires another connection to retrieve them. Like the /eecerts PC, this service is not meant to be a general-purpose repository to which clients query a repository and then get a response; this is purely a push mechanism. The /crls PC is described in Section 4.

- /symmetrickeys - In some cases, clients use symmetric keys [RFC6031] when communicating with their peers. If the client’s peers are known by the server a priori, then providing them saves the client or an administrator from later having to find, retrieve, and install them. Like the /eecerts and /crls PCs, this service is not meant to be a general-purpose repository to which clients query a repository and then get a response; this is purely a push mechanism for the keys themselves. However, things do not always go as planned, and clients need to inform the server about any errors. If things did go well, then the client, if requested, needs to provide a receipt [RFC7191]. The /symmetrickeys and /symmetrickeys/return PCs are described in Section 5.

- /firmware - Some client firmware and software support automatic update mechanisms, and some do not. For those that do not, the /firmware PC provides a mechanism for the infrastructure to inform the client that firmware and software updates [RFC4108] are available. Because updates do not always go as planned and because sometimes the server needs to know whether the firmware was received and processed, this PC also provides a mechanism to return errors and receipts. The /firmware and /firmware/return PCs are defined in Section 6.

- /tamp - To control the TAs in client TA databases, servers use the /tamp PC to request that clients retrieve TAMP (Trust Anchor Management Protocol) query, update, and adjust packages [RFC5934], and clients use the /tamp/return PC to return TAMP responses, confirms, and errors [RFC5934]. The /tamp and /tamp/return PCs are defined in Section 7.

This document also extends the /est/serverkeygen PC [RFC7030] to support the following (see Section 8):

- Returning asymmetric key package receipts and errors [RFC7191].

- Encapsulating returned asymmetric keys in additional CMS (Cryptographic Message Syntax) content types [RFC7193].

- Returning server-generated public key pairs encapsulated in PKCS #12 (Public Key Cryptography Standard #12) [RFC7292].

While the motivation is to provide packages to clients during enrollment so that they can perform securely after enrollment, the services defined in this specification can be used after enrollment.

1.1. Definitions

Familiarity with the following specifications is assumed:

- "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages" [RFC4108]
- "Certificate Management over CMS (CMC)" [RFC5272]
- "Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type" [RFC6032]
- "Cryptographic Message Syntax (CMS)" [RFC5652]
- "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)" [RFC6268]
- "Trust Anchor Management Protocol (TAMP)" [RFC5934]
- "Cryptographic Message Syntax (CMS) Content Constraints Extension" [RFC6010]
- "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type" [RFC6031]
- "Enrollment over Secure Transport" [RFC7030]
- "Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types" [RFC7191]

Also, familiarity with the CMS protecting content types signed-data and encrypted-data [RFC5652] is assumed. The CMS encrypted key package is defined in [RFC6032].

In addition to the definitions found in [RFC7030], the following definitions are used in this document:

Agent: An entity that performs functions on behalf of a client. Agents can service a) one or more clients on the same network as the server, b) clients on non-IP-based networks, or c) clients that have a non-electronic air gap [RFC4949] between themselves and the server. Interactions between the agent and client in the last two cases are beyond the scope of this document. Before an agent can service clients, the agent must have a trust relationship with the server (i.e., be authorized to act on behalf of clients).

Client: A device that ultimately consumes and uses the packages to enable communications. In other words, the client is the endpoint for the packages, and an agent may have one or more clients. To avoid confusion, this document henceforth uses the term "client" to refer to both agents and clients.

Package: An object that contains one or more content types. There are numerous types of packages, e.g., packages for asymmetric keys, symmetric keys, encrypted keys, CRLs, firmware, and TAMP. See Section 2.1.1. All of these packages are digitally signed by their creator and encapsulated in a CMS signed-data [RFC5652] [RFC6268] (except the public key certificates and CRLs that are already digitally signed by a CA): firmware receipts and errors; TAMP responses, confirms, and errors; and "key package" receipts and errors that can be optionally signed. Certificates and CRLs are included in a package that uses signed-data, which is often referred to as a "degenerate CMS", or as a "certs-only" [RFC5751] [RFC6268] or "crls-only" message (see Section 4.2), but no signature or content is present -- hence the names "certs-only" and "crls-only".

Note: As per [RFC7030], the creator may or may not be the EST server or the EST CA.

1.2. Authentication and Authorization

Client and server authentication as well as client and server authorization are as defined in [RFC7030]. The requirements for each are discussed in the "request" and "response" sections (e.g., Sections 3.1 and 3.2 of this document) of each of the PCs defined herein.

The requirements for the TA databases are as specified in [RFC7030] as well.

1.3. TLS Cipher Suites

TLS (Transport Layer Security) cipher suites and issues associated with them are as defined in [RFC7030].

1.4. URI Configuration

As specified in Section 3.1 of [RFC7030], the client is configured with sufficient information to form the server URI [RFC3986]. Like EST, this configuration mechanism is beyond the scope of this document.

1.5. Message Types

This document uses existing media types for the messages as specified by "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP" [RFC2585], "The application/pkcs10 Media Type" [RFC5967], and "Certificate Management over CMS (CMC)" [RFC5272].

For consistency with [RFC5273], each distinct EST message type uses an HTTP Content-Type header with a specific media type.

The EST messages, their corresponding media types for each operation, and the sections that provide request and response information are as follows:

Message type (per operation)	Request media type Response media type(s) Source(s) of types	Request Response
Locate Available Packages /pal	N/A application/xml or application/json [RFC7303] [RFC8259]	Section 2.2 Section 2.3
Distribute EE Certificates /eecerts	N/A application/pkcs7-mime [RFC5751]	Section 3.1 Section 3.2
Distribute CRLs /crls	N/A application/pkcs7-mime [RFC5751]	Section 4.1 Section 4.2
Symmetric Key Distribution /symmetrickeys	N/A application/cms [RFC7193]	Section 5.1.1 Section 5.1.2
Return Symmetric Key Receipts/Errors /symmetrickeys/ return	application/cms N/A [RFC7193]	Section 5.2.1 Section 5.2.2

Firmware Distribution /firmware	N/A application/cms [RFC7193]	Section 6.1.1 Section 6.1.2
Return Firmware Receipts/Errors /firmware/return	application/cms N/A [RFC7193]	Section 6.2.1 Section 6.2.2
Trust Anchor Management /tamp	N/A application/ tamp-status-query tamp-update tamp-apex-update tamp-community-update tamp-sequence-adjust [RFC5934]	Section 7.1.1 Section 7.1.2
Return TAMP Responses/ Confirms/ Errors /tamp/return	application/ tamp-status-response tamp-update-confirm tamp-apex-update-confirm tamp-community-update-confirm tamp-sequence-adjust-confirm tamp-error N/A [RFC5934]	Section 7.2.1 Section 7.2.2
Server-Side Key Generation /serverkeygen	application/pkcs10 with content type attribute CSR* application/cms [RFC5967] [RFC7193] [RFC7030]	Section 8.1 Section 8.1

Return Asymmetric Key Receipts/Errors /serverkeygen/return	application/cms N/A [RFC7193]	Section 8.2 Section 8.2
Server-Side Key Generation: PKCS #12 /serverkeygen	application/pkcs10 application/pkcs12 [RFC5967] [RFC7193] [RFC7030]	Section 8.3.1 Section 8.3.2

* Certificate Signing Request

1.6. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Locate Available Packages

The PAL (Package Availability List) is either an XML (Extensible Markup Language) [XML] or JSON (JavaScript Object Notation) [RFC8259] object available through the /pal PC, which furnishes the following information to clients:

o Advertisements for available packages that can be retrieved from the server;

o Notifications to begin public key certificate management or to return package receipts and errors; and

o Advertisement for another PAL.

After being configured (see Section 1.4), the client can use this service to retrieve its PAL (see Section 2.1), which, if properly constructed (see Section 2.3), allows the client to determine some or all of the security-related packages needed for bootstrapping. Each PAL entry refers to other PCs (as defined in this document and in [RFC7030]) that clients use to a) retrieve packages that are available to them (e.g., CA certificates, firmware, trust anchors, symmetric keys, and asymmetric keys) or b) receive notifications to initiate public key certificate enrollment. PAL entries can also be used to notify clients that they are to return receipts or errors for certain packages (see Section 2.1.1). Placing these entries after entries that clients used to retrieve the packages is the same as requesting receipts in the originally distributed package. Figure 1 provides a ladder diagram for the /pal PC protocol flow. Appendix A provides a detailed example.

```
                           |                      |
                 Client    |     Establish TLS    |    Server
                           |        Session       |
                           |<-------------------->|
                           |                      |
                           |           Request PAL |
                           |   (HTTP GET Request) |
                           |--------------------->|
                           |<---------------------|
                           | Deliver PAL          |
                           | (HTTP GET Response)  |
                           |                      |
                           | Request package by   |
                           | specified URI        |
                           | (HTTP GET or POST    |
                           |  Request)            |
                           |--------------------->|
                           |<---------------------|
                           | Deliver requested    |
                           | CMS package product  |
                           | (HTTP GET or POST    |
                           |  Response)           |
                           |                      |

                             Repeat as necessary.
```

Figure 1: /pal Message Sequence

PALs are designed to support an arbitrary number of entries, but for PALs that need to be divided for any reason, there is a special PAL entry type that constitutes a collection of "PAL package types". Package type 0001 ("Additional PAL value present") refers to another PAL. See Sections 2.1 and 2.1.1. If present, the 0001 package type is always last because other entries after it are ignored. Also, in order to avoid needlessly dereferencing URIs, the 0001 package type cannot be the only PAL entry. In addition to using the PAL during bootstrapping, clients can be configured to periodically poll the server to determine if updated packages are available for them. Note that the mechanism to configure how often clients poll the server is beyond the scope of this document. However, there are some services that support indicating when a client should retry its request (e.g., simple enrollment and re-enroll responses include the Retry-After header [RFC7030]).

As noted earlier, the PAL supports two variants: XML and JSON. Clients include the HTTP Accept header [RFC7231] when they connect to the server to indicate whether they support XML or JSON.

The client MUST authenticate the server as specified in [RFC7030], and the client MUST check the server's authorization as specified in [RFC7030].

The server MUST authenticate the client as specified in [RFC7030], and the server MUST check the client's authorization as specified in [RFC7030].

PAL support is OPTIONAL. It is shown in figures throughout this document, but clients need not support the PAL to access services offered by the server.

2.1. PAL Format

Each PAL is composed of zero or more entries. Each entry is composed of four fields -- type, date, size, and info -- whose semantics follow:

Note: Both XML elements and JSON values are described below. XML elements are enclosed in angle brackets (<>), and JSON values are enclosed in single quotes (''). When described together, they are enclosed in square brackets ([]) separated by a vertical bar (|).

- [<type> | 'type'] uniquely identifies each package that a client may retrieve from the server with a 4-digit string. [<type> | 'type'] MUST be present. The PAL package types are defined in Section 2.1.1.

- [<date> | 'date'] indicates one of the following:

  - The date and time that the client last successfully downloaded the identified package from the server. [<date> | 'date'] MUST be represented as Generalized Time with 20 characters: YYYY-MM-DDTHH:MM:SSZ; <date> matches the dateTime production in "canonical representation" [XMLSCHEMA]; 'date' is a string. Implementations SHOULD NOT rely on time resolution finer than seconds and MUST NOT generate time instants that specify leap seconds.

  * The omission of [<date> | ’date’] indicates the following:

    - There is no indication that the client has successfully downloaded the identified package, or

    - The PAL entry corresponds to a pointer to the next PAL, or the server is requesting a package from the client (e.g., certification request, receipt, error).

o [<size> | ’size’] indicates the size in bytes of the package; <size> is a nonNegativeInteger, and ’size’ is a number. A package size of zero (i.e., "0" without the quotes) indicates that the client needs to begin a transaction, return an error, or return a receipt. [<size> | ’size’] MUST be present.

o [<info> | ’info’] provides an SKI (Subject Key Identifier), a DN (Distinguished Name), an Issuer and Serial Number tuple, or a URI, i.e., it is a choice between these four items, all of which are defined in [RFC5280]. When a URI [RFC3986] is included, [<uri> | ’uri’] indicates the location where the identified package can be retrieved. When a DN, an SKI, or an Issuer Name and Serial Number tuple is included, it points to a certificate that is the subject of the notification (i.e., the certificate to be rekeyed or renewed); [<dn> | ’dn’] is encoded as a string with the format defined in [RFC4514]; <ski> is a hexBinary, and ’ski’ is a string of hex digits (i.e., 0-9, a-f, and A-F); [<iasn> | ’iasn’] includes both [<issuer> | ’issuer’] and [<serial> | ’serial’] as a complexType in XML and an object in JSON. [<issuer> | ’issuer’] is a DN encoded as a string with the format defined in [RFC4514]; <serial> is a positiveInteger, and ’serial’ is a number. [<info> | ’info’] MUST be present, and [<info> | ’info’] MUST include exactly one [<dn> | ’dn’], [<ski> | ’ski’], [<iasn> | ’iasn’], or [<uri> | ’uri’].

Clients are often limited by the size of objects they can consume; the PAL is not immune to these limitations. As opposed to picking a limit for all clients, a special package type (0001) is defined (see Section 2.1.1) to indicate that another PAL is available. Servers can use this value to limit the size of the PALs provided to clients. The mechanism for servers to know client PAL size limits is beyond the scope of this document; one possible solution is through provisioned information.

2.1.1. PAL Package Types

Table 1 lists the PAL package types that are defined by this document:

Package Number	Package Description
0000	Reserved
0001	Additional PAL value present
0002	X.509 CA certificate
0003	X.509 EE certificate
0004	X.509 ARL
0005	X.509 CRL
0006	Start DS certificate enrollment with CSR attribute
0007	Start DS certificate enrollment
0008	DS certificate enrollment (success)
0009	DS certificate enrollment (failure)
0010	Start DS certificate re-enrollment
0011	DS certificate re-enrollment (success)
0012	DS certificate re-enrollment (failure)
0013	Start KE certificate enrollment with CSR attribute
0014	Start KE certificate enrollment
0015	KE certificate enrollment (success)
0016	KE certificate enrollment (failure)
0017	Start KE certificate re-enrollment
0018	KE certificate re-enrollment (success)
0019	KE certificate re-enrollment (failure)
0020	Asymmetric Key Package (PKCS #8)
0021	Asymmetric Key Package (CMS)
0022	Asymmetric Key Package (PKCS #12)
0023	Asymmetric Key Package Receipt or Error
0024	Symmetric Key Package
0025	Symmetric Key Package Receipt or Error
0026	Firmware Package
0027	Firmware Package Receipt or Error
0028	TAMP Status Query
0029	TAMP Status Query Response or Error
0030	Trust Anchor Update
0031	Trust Anchor Update Confirm or Error
0032	Apex Trust Anchor Update
0033	Apex Trust Anchor Update Confirm or Error
0034	Community Update
0035	Community Update Confirm or Error
0036	Sequence Number Adjust
0037	Sequence Number Adjust Confirm or Error

Table 1: PAL Package Types

Note: "CSR" is Certificate Signing Request, "DS" is Digital Signature, and "KE" is Key Establishment.

PAL package types are essentially hints about the type of package the client is about to retrieve or is asked to return. Savvy clients can parse the packages to determine what has been provided, but in some instances it is better to know before retrieving the package. The hint provided here does not obviate the need for clients to check the type of package provided before they store it, possibly in specially allocated locations (i.e., some clients might store Root ARLs separately from intermediate CRLs). For packages provided by the client, the server is asking the client to provide an enrollment package, receipt, response, confirm, or error.

The PAL package types have the following meanings:

Note: The semantics behind Codes 0002 and 0006-0021 are defined in [RFC7030].

0000 Reserved: Reserved for future use.

0001 Additional PAL value present: Indicates that this PAL entry refers to another PAL by referring to another /pal URI, which is defined in this section. This PAL package type limits the size of PALs to a more manageable size for clients. If this PAL package type appears, it MUST be the last entry in the PAL.

Additionally, in order to avoid needlessly dereferencing URIs, this PAL package type MUST NOT be the only entry.

0002 X.509 CA certificate: Indicates that one or more CA certificates [RFC5280] are available for the client by pointing to a /cacerts URI, which is defined in [RFC7030].

0003 X.509 EE certificate: Indicates that one or more EE certificates [RFC5280] are available for the client by pointing to an /eecerts URI, which is defined in Section 3.

0004 X.509 ARL: Indicates that one or more ARLs (Authority Revocation Lists) [RFC5280] are available for the client by pointing to a /crls URI, which is defined in Section 4.

0005 X.509 CRL: Indicates that one or more CRLs (Certificate Revocation Lists) [RFC5280] are available for the client by pointing to a /crls URI, which is defined in Section 4.

Note: See Section 9 for additional information about PAL and certificate enrollment interaction. See Appendix B for additional informative information.

0006 Start DS certificate enrollment with CSR: Indicates that the client needs to begin enrolling its DS certificate (i.e., any certificate for which the key usage extension will have a digital signature set), using a template provided by the server with a CSR (Certificate Signing Request) attribute (see Appendix B). The PAL entry points to a /csrattrs URI, which is defined in [RFC7030].

0007 Start DS certificate enrollment: Indicates that the client needs to begin enrolling its DS certificate. The PAL entry points to a /simpleenroll URI, which is defined in [RFC7030].

0008 DS certificate enrollment (success): Indicates that the client needs to retrieve a successful certification response. The PAL entry points to a /simpleenroll or a /fullcmc URI, both of which are defined in [RFC7030].

0009 DS certificate enrollment (failure): Indicates that the client needs to retrieve a failed certification response for a DS certificate. This PAL entry points to a /simpleenroll or a /fullcmc URI.

0010 Start DS certificate re-enrollment: Indicates that the client needs to rekey or renew a DS certificate. The PAL entry points to a /simplereenroll or a /fullcmc URI.

0011 DS certificate re-enrollment (success): See PAL package type 0008.

0012 DS certificate re-enrollment (failure): See PAL package type 0009.

Note: The KE (Key Establishment) responses that follow use the same URIs as DS certificates, except that the certificates' key usage extension is set to only key agreement or key transport.

0013 Start KE certificate enrollment with CSR: See PAL package type 0006.

0014 Start KE certificate enrollment: See PAL package type 0007.

0015 KE certificate enrollment (success): See PAL package type 0008.

0016 KE certificate enrollment (failure): See PAL package type 0009.

0017 Start KE certificate re-enrollment: See PAL package type 0010.

0018 KE certificate re-enrollment (success): See PAL package type 0008.

0019 KE certificate re-enrollment (failure): See PAL package type 0009.

Note: The variations in the asymmetric key packages are due to the number of CMS content types that can be used to protect the asymmetric key; the syntax for the asymmetric key is the same, but additional ASN.1 is needed to include it in a signed-data (i.e., the ASN.1 needs to be a CMS content type and not the private key info type). See Section 8 of this document for additional information.

0020 Asymmetric Key Package (PKCS #8): Indicates that an asymmetric key generated by the server is available for the client; the package is an asymmetric key without additional encryption as specified in Section 4.4.2 of [RFC7030]. The PAL entry points to a /serverkeygen or a /fullcmc URI, which are defined in [RFC7030].

0021 Asymmetric Key Package (CMS): See PAL package type 0020 (the difference being that the package available is an asymmetric key package [RFC5958] that is signed and encapsulated in a signed-data content type, as specified in Section 4.4.2 of [RFC7030]). Also, see Section 8.1 of this document.

0022 Asymmetric Key Package (PKCS #12): See PAL package type 0020 (the difference being that the package available is the PKCS #12 [RFC7292] content type). See Section 8.3 of this document.

0023 Asymmetric Key Package Receipt or Error: Indicates that the server wants the client to return a key package receipt or error [RFC7191] to the /serverkeygen/return URI, which is defined in Section 8.

0024 Symmetric Key Package: Indicates that a symmetric key package [RFC6031] is available for the client by pointing to a /symmetrickeys URI, which is defined in Section 5.

0025 Symmetric Key Package Receipt or Error: Indicates that the server wants the client to return a key package receipt or error [RFC7191] to the /symmetrickeys/return URI, which is defined in Section 5.

0026 Firmware Package: Indicates that a firmware package [RFC4108] is available for the client, using the /firmware URI, which is defined in Section 6.

0027 Firmware Package Receipt or Error: Indicates that the server wants the client to return a firmware package load receipt or error [RFC4108] to the /firmware/return URI, which is defined in Section 6.

Note: The /tamp and tamp/return URIs are defined in Section 7.

0028 TAMP Status Query: Indicates that a TAMP Status Query package [RFC5934] is available for the client, using the /tamp URI.

0029 TAMP Status Query Response or Error: Indicates that the server wants the client to return a TAMP Status Query Response or Error [RFC5934] to the /tamp/return URI.

0030 Trust Anchor Update: Indicates that a Trust Anchor Update package [RFC5934] is available for the client, using the /tamp URI.

0031 Trust Anchor Update Confirm or Error: Indicates that the server wants the client to return a Trust Anchor Update Confirm or Error [RFC5934] to the /tamp/return URI.

0032 Apex Trust Anchor Update: Indicates that an Apex Trust Anchor Update package [RFC5934] is available for the client, using the /tamp URI.

0033 Apex Trust Anchor Update Confirm or Error: Indicates that the server wants the client to return an Apex Trust Anchor Update Confirm or Error [RFC5934] to the /tamp/return URI.

0034 Community Update: Indicates that a Community Update package [RFC5934] is available for the client, using the /tamp URI.

0035 Community Update Confirm or Error: Indicates that the server wants the client to return a Community Update Confirm or Error [RFC5934] to the /tamp/return URI.

0036 Sequence Number Adjust: Indicates that a Sequence Number Adjust package [RFC5934] is available for the client, using the /tamp URI.

0037 Sequence Number Adjust Confirm or Error: Indicates that the server wants the client to return a Sequence Number Adjust Confirm or Error [RFC5934] to the /tamp/return URI.

2.1.2. PAL XML Schema

The namespace is specified in Section 11.1. The fields in the schema were discussed earlier, in Sections 2.1 and 2.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="https://www.w3.org/2001/XMLSchema"
  xmlns:pal="urn:ietf:params:xml:ns:pal"
  targetNamespace="urn:ietf:params:xml:ns:pal"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      This schema defines the types and elements needed
      to retrieve client packages from the server or for the
      client to post packages to the server.
    </xsd:documentation>
  </xsd:annotation>

  <!-- ===== Element Declarations ===== -->

  <xsd:element name="pal" type="pal:PAL" />

  <!-- ===== Complex Data Element Type Definitions ===== -->

  <xsd:complexType name="PAL">
    <xsd:annotation>
      <xsd:documentation>
        This type defines the Package Availability List (PAL).
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="message" type="pal:PALEntry"
        minOccurs="0" maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            This item contains information about the package
            and a link that the client uses to download or post
            the package.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

```
     <xsd:complexType name="PALEntry">
       <xsd:annotation>
         <xsd:documentation>
           This type defines a product in the PAL.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
         <xsd:element name="type" type="pal:PackageType" />
         <xsd:element name="date" type="pal:GeneralizedTimeType"
           minOccurs="0" />
         <xsd:element name="size" type="xsd:nonNegativeInteger">
           <xsd:annotation>
             <xsd:documentation>
               This item indicates the package’s size.
             </xsd:documentation>
           </xsd:annotation>
         </xsd:element>
         <xsd:element name="info" type="pal:PackageInfoType" />
       </xsd:sequence>
     </xsd:complexType>

     <xsd:complexType name="PackageInfoType">
       <xsd:annotation>
         <xsd:documentation>
           This type allows a choice of X.500 Distinguished Name,
           Subject Key Identifier, Issuer and Serial Number tuple,
           or URI.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:choice>
         <xsd:element name="dn"   type="pal:DistinguishedName" />
         <xsd:element name="ski"  type="pal:SubjectKeyIdentifier" />
         <xsd:element name="iasn" type="pal:IssuerAndSerialNumber" />
         <xsd:element name="uri"  type="pal:ThisURI" />
       </xsd:choice>
     </xsd:complexType>
     <xsd:complexType name="IssuerAndSerialNumber">
       <xsd:annotation>
         <xsd:documentation>
           This type holds the issuer Distinguished Name and
           serial number of a referenced certificate.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
         <xsd:element name="issuer" type="pal:DistinguishedName" />
         <xsd:element name="serial" type="xsd:positiveInteger" />
       </xsd:sequence>
     </xsd:complexType>

     <!-- ===== Simple Data Element Type Definitions ===== -->

     <xsd:simpleType name="PackageType">
       <xsd:annotation>
         <xsd:documentation>
           This type identifies each package that a client may retrieve
           from the server with a 4-digit string.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:string">
         <xsd:pattern value="d{4}" />
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="GeneralizedTimeType">
       <xsd:annotation>
         <xsd:documentation>
           This type indicates the date and time (YYYY-MM-DDTHH:MM:SSZ)
           that the client last acknowledged successful receipt of the
           package; it is absent if a) there is no indication that the
           package has been downloaded or b) the PAL entry corresponds
           to a pointer to the next PAL.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:dateTime">
         <xsd:pattern value=".*:d{2}Z" />
         <xsd:minInclusive value="2013-05-23T00:00:00Z" />
       </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="DistinguishedName">
       <xsd:annotation>
         <xsd:documentation>
           This type holds an X.500 Distinguished Name.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:string">
         <xsd:maxLength value="1024" />
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="SubjectKeyIdentifier">
       <xsd:annotation>
         <xsd:documentation>
           This type holds a hex string representing the value of a
           certificate’s SubjectKeyIdentifier.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:hexBinary">
         <xsd:maxLength value="1024" />
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="ThisURI">
       <xsd:annotation>
         <xsd:documentation>
           This type holds a URI but is length limited.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:anyURI">
       <xsd:maxLength value="1024" />
       </xsd:restriction>
     </xsd:simpleType>

   </xsd:schema>
```

2.1.3. PAL JSON Object

The following is an example PAL JSON object. The fields in the object were discussed earlier, in Sections 2.1 and 2.1.1.

```
[
  {
    "type": "0003",
    "date": "2016-12-29T09:28:00Z",
    "size": 1234,
    "info":
     {
      "uri": "https://www.example.com/.well-known/est/eecerts/1234"
     }
  },

  {
    "type": "0006",
    "date": "2016-12-29T09:28:00Z",
    "size": 1234,
    "info":
     {
      "iasn":
        {
         "issuer": "CN=Sean Turner,O=sn3rd,C=US",
         "serial": 0
        }
     }
  }
]
```

2.2. Request PAL

Clients request their PAL with an HTTP GET [RFC7231], using an operation path of "/pal". Clients indicate whether they would prefer XML or JSON by including the HTTP Accept header [RFC7231] with either "application/xml" or "application/json", respectively.

2.3. Provide PAL

If the server has a PAL for the client, the server response MUST contain an HTTP 200 response code with a Content-Type of "application/xml" [RFC7303] or "application/json" [RFC8259].

When the server constructs a PAL, an order of precedence for PAL offerings is based on the following rationale:

- o /cacerts and /crls packages are the most important because they support validation decisions on certificates used to sign and encrypt other listed PAL items.

- o /csrattrs are the next in importance, since they provide information that the server would like the client to include in its certificate enrollment request.

- o /simpleenroll, /simplereenroll, and /fullcmc packages are next in importance, since they can impact a certificate used by the client to sign CMS content or a certificate to establish keys for encrypting content exchanged with the client.

  * A client engaged in certificate management SHOULD accept and process CA-provided transactions as soon as possible to avoid undue delays that might lead to protocol failure.

- o /symmetrickeys, /firmware, /tamp, and /eecerts packages containing keys and other types of products are last. Precedence SHOULD be given to packages that the client has not previously downloaded. The items listed in a PAL may not identify all of the packages available for a device. This can be for any of the following reasons:

  * The server may temporarily withhold some outstanding PAL items to simplify client processing.

  * If a CA has more than one certificate ready for the client, the server will provide a notice for one at a time. Pending notices will be serviced in order, according to the date when the certificate will be used (earliest date first).

When rejecting a request, the server specifies either an HTTP 4xx error or an HTTP 5xx error.

All other return codes are handled as specified in Section 4.2.3 of [RFC7030] (i.e., 202 handling and all other HTTP response codes).

3. Distribute EE Certificates

Numerous mechanisms exist for clients to query repositories for certificates. The service provided by the /eecerts PC is different in that it is not a general-purpose query for client certificates; instead, it allows the server to provide peer certificates to a client that the server knows through an out-of-band mechanism that the client will be communicating with. For example, a router being provisioned that connects to two peers can be provisioned with not only its certificate but also with the peers' certificates.

The server need not authenticate or authorize the client for distributing an EE certificate, because the package contents are already signed by a CA (i.e., the certificate(s) in a certs-only message has already been signed by a CA). The message flow is similar to Figure 1, except that the connection need not be HTTPS:

```
                          |                       |
             Client       |    Establish TLS      |    Server
                          |       Session         |
                          |<--------------------->|
                          |                       |
                          |          Request PAL  |
                          |   (HTTP GET Request)  |
                          |---------------------->|
                          |<----------------------|
                          | Deliver PAL           |
                          | (HTTP GET Response)   |
                          |                       |
                          |   Request EE Cert(s)  |
                          |   (HTTP GET Request)  |
                          |---------------------->|
                          |<----------------------|
                          | Deliver EE Cert(s)    |
                          | (HTTP GET Response)   |
                          |                       |
```

Figure 2: /eecerts Message Sequence

3.1. EE Certificate Request

Clients request EE certificates with an HTTP GET [RFC7231], using an operation path of "/eecerts".

3.2. EE Certificate Response

The response and processing of the returned error codes are identical to what is described in Section 4.1.3 of [RFC7030], except that the certificate provided is not the one issued to the client; instead, one or more client’s peer certificates are returned in the certs-only message.

Clients MUST reject EE certificates that do not validate to an authorized TA.

4. Distribute CRLs and ARLs

CRLs (and ARLs) are needed in many instances to perform certificate path validation [RFC5280]. They can be obtained from repositories if their location is provided in the certificate. However, the client needs to parse the certificate and perform an additional round trip to retrieve them. Providing CRLs during bootstrapping obviates the need for the client to parse the certificate and aids those clients who might be unable to retrieve the CRL. Clients are free to obtain CRLs on which they rely from sources other than the server (e.g., a local directory). The /crls PC allows servers to distribute CRLs at the same time that clients retrieve their certificate(s) and CA certificate(s) as well as peer certificates.

The server need not authenticate or authorize the client for distributing a CRL, because the package contents are already signed by a CA (i.e., the CRLs in a crls-only message have already been signed by a CA). The message flow is as depicted in Figure 2 but with "CRL(s)" instead of "EE Cert(s)".

4.1. CRL Request

Clients request CRLs with an HTTP GET [RFC7231], using an operation path of "/crls".

4.2. CRL Response

The response, and the processing of that response, are identical to what is described in Section 4.1.3 of [RFC7030], except that instead of providing the issued certificate one of more CRLs are returned in the crls-only message.

Clients MUST reject CRLs that do not validate to an authorized TA.

5. Symmetric Keys, Receipts, and Errors

In addition to public keys, clients often need one or more symmetric keys to communicate with their peers. The /symmetrickeys PC allows the server to distribute symmetric keys to clients.

Distribution of keys does not always work as planned, and clients need a way to inform the server that something has gone wrong; they also need a way to inform the server, if asked, that the distribution process has successfully completed. The /symmetrickeys/return PC allows clients to provide errors and receipts.

Clients MUST authenticate the server, and clients MUST check the server’s authorization.

The server MUST authenticate clients, and the server MUST check the client’s authorization.

HTTP GET [RFC7231] is used when the server provides the key to the client (see Section 5.1), using the /symmetrickeys PC; HTTP POST [RFC7231] is used when the client provides a receipt (see Section 5.2) or an error (see Section 5.2) to the server with the /symmetrickeys/return PC.

5.1. Symmetric Keys

Servers use /symmetrickeys to provide symmetric keys to clients; the symmetric key package is defined in [RFC6031].

As with the /serverkeygen PC defined in [RFC7030], the default method for distributing the symmetric key uses the encryption mode of the negotiated TLS cipher suite. Keys are not protected by preferred key-wrapping methods such as AES Key Wrap [RFC3394] or AES Key Wrap with Padding [RFC5649], because encryption of the symmetric key beyond that provided by TLS is OPTIONAL. Therefore, the cipher suite used to return the symmetric key MUST offer cryptographic strength that is commensurate with the symmetric key being delivered to the client. The cipher suite used MUST NOT have the NULL encryption algorithm, as this will disclose the unprotected symmetric key. It is strongly RECOMMENDED that servers always return encrypted symmetric keys.

The following depicts the protocol flow:

```
                         |                       |
               Client    |    Establish TLS      |    Server
                         |       Session         |
                         |<--------------------->|
                         |                       |
                         |          Request PAL  |
                         |   (HTTP GET Request)  |
                         |---------------------->|
                         |<----------------------|
                         | Deliver PAL           |
                         | (HTTP GET Response)   |
                         |                       |
                         |    Req Symmetric Key  |
                         |   (HTTP GET Request)  |
                         |---------------------->|
                         |<----------------------|
                         | Deliver Symmetric Key |
                         | (HTTP GET Response)   |
                         |                       |
```

Figure 3: /symmetrickeys Message Sequence

5.1.1. Distribute Symmetric Keys

Clients request the symmetric key from the server with an HTTP GET [RFC7231], using an operation path of "/symmetrickeys".

5.1.2. Symmetric Key Response

If the request is successful, the server response MUST have an HTTP 200 response code with a Content-Type of "application/cms" [RFC7193]. The optional application/cms encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned symmetric key. The returned content varies:

o If additional encryption is not being employed, the content associated with application/cms is a DER-encoded [X.690] symmetric key package.

o If additional encryption is employed, the content associated with application/cms is DER-encoded enveloped-data that encapsulates a signed-data that further encapsulates a symmetric key package.

- o If additional encryption and origin authentication are employed, the content associated with application/cms is a DER-encoded signed-data that encapsulates an enveloped-data that encapsulates a signed-data that further encapsulates a symmetric key package.

- o If CCC (CMS Content Constraints) [RFC6010] is supported, the content associated with application/cms is a DER-encoded encrypted key package [RFC6032]. The encrypted key package provides three choices to encapsulate keys: EncryptedData, EnvelopedData, and AuthEnvelopedData. Prior to employing one of these three encryption choices, the key package can be encapsulated in a signed-data.

How the server knows whether the client supports the encrypted key package is beyond the scope of this document.

When rejecting a request, the server specifies either an HTTP 4xx error or an HTTP 5xx error.

If a symmetric key package (which might be signed) or an encrypted key package (which might be signed before and after encryption) is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

Note: Absent a policy on the client side requiring a signature, a malicious EST server can simply strip the signature, thus bypassing that check. In that case, this requirement is merely a sanity check, serving to detect mis-signed packages or misconfigured clients.

[RFC3370], [RFC5753], [RFC5754], [RFC6033], [RFC6160], and [RFC6161] provide algorithm details for use when protecting the symmetric key package and encrypted key package.

5.2. Symmetric Key Receipts and Errors

Clients use /symmetrickeys/return to provide symmetric key package receipts; the key package receipt content type is defined in [RFC7191]. Clients can be configured to automatically return receipts after processing a symmetric key package, return receipts based on processing of the key-package-identifier-and-receipt-request attribute [RFC7191], or return receipts when prompted by a PAL entry.

Servers can indicate that clients return a receipt by including the key-package-identifier-and-receipt-request attribute in a signed-data as a signed attribute. However, this attribute only appears when additional encryption is employed (see Section 5.1.2).

Clients also use /symmetrickeys/return to return symmetric key package errors; the key package error content type is defined in [RFC7191]. Clients can be configured to automatically return errors after processing a symmetric key package or based on a PAL entry.

The following depicts the protocol flow:

```
                       |                       |
              Client   |    Establish TLS      |    Server
                       |       Session         |
                       |<--------------------->|
                       |                       |
                       |           Request PAL |
                       |   (HTTP GET Request)  |
                       |---------------------->|
                       |<----------------------|
                       | Deliver PAL           |
                       | (HTTP GET Response)   |
                       |                       |
                       | Return Receipt/Error  |
                       |  (HTTP POST Request)  |
                       |---------------------->|
                       |<----------------------|
                       | (HTTP POST Response)  |
                       | status code only      |
                       | no content            |
                       |                       |
```

Figure 4: /symmetrickeys/return Message Sequence

5.2.1. Provide Symmetric Key Receipt or Error

Clients return symmetric key receipts and errors to the server with an HTTP POST [RFC7231], using an operation path of "/symmetrickeys/return". The returned content varies:

- o The key package receipt is digitally signed [RFC7191]; the Content-Type is "application/cms" [RFC7193]; and the associated content is signed-data, which encapsulates a key package receipt.

- o If the key package error is not digitally signed, the Content-Type is "application/cms" and the associated content is a key package error. If the key package error is digitally signed, the Content-Type is "application/cms" and the associated content is signed-data, which encapsulates a key package error.

The optional application/cms encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the receipt or error.

[RFC3370], [RFC5753], [RFC5754], and [RFC7192] provide algorithm details for use when protecting the key package receipt or key package error.

5.2.2. Symmetric Key Receipt or Error Response

If the client successfully provides a receipt or error, the server response has an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server specifies either an HTTP 4xx error or an HTTP 5xx error.

If a key package receipt or key package error is digitally signed, the server MUST reject it if the digital signature does not validate back to an authorized TA.

6. Firmware, Receipts, and Errors

Servers can distribute object code for cryptographic algorithms and software with the firmware package [RFC4108].

Clients MUST authenticate the server, and clients MUST check the server’s authorization.

The server MUST authenticate the client, and the server MUST check the client’s authorization.

The /firmware PC uses an HTTP GET [RFC7231], and the /firmware/return PC uses an HTTP POST [RFC7231]. GET is used when the client retrieves firmware from the server (see Section 6.1); POST is used when the client provides a receipt (see Section 6.2) or an error (see Section 6.2).

6.1. Firmware

The /firmware URI is used by servers to provide firmware packages to clients.

The message flow is as depicted in Figure 3 modulo replacing "Symmetric Key" with "Firmware Package".

6.1.1. Distribute Firmware

Clients request firmware from the server with an HTTP GET [RFC7231], using an operation path of "/firmware".

6.1.2. Firmware Response

If the request is successful, the server response MUST have an HTTP 200 response code with a Content-Type of "application/cms" [RFC7193]. The optional encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned firmware. The returned content varies:

- o If the firmware is unprotected, then the Content-Type is "application/cms" and the content is the DER-encoded [X.690] firmware package.

- o If the firmware is compressed, then the Content-Type is "application/cms" and the content is the DER-encoded [X.690] compressed data that encapsulates the firmware package.

- o If the firmware is encrypted, then the Content-Type is "application/cms" and the content is the DER-encoded [X.690] encrypted-data that encapsulates the firmware package (which might be compressed prior to encryption).

- o If the firmware is signed, then the Content-Type is "application/cms" and the content is the DER-encoded [X.690] signed-data that encapsulates the firmware package (which might be compressed, encrypted, or compressed and then encrypted prior to signature).

How the server knows whether the client supports the unprotected, signed, compressed, and/or encrypted firmware package is beyond the scope of this document.

When rejecting a request, the server specifies either an HTTP 4xx error or an HTTP 5xx error.

If a firmware package is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the firmware package.

6.2. Firmware Receipts and Errors

Clients use the /firmware/return PC to provide firmware package load receipts and errors [RFC4108]. Clients can be configured to automatically return receipts and errors after processing a firmware package or based on a PAL entry.

The message flow is as depicted in Figure 4 modulo the receipt or error is for a firmware package.

6.2.1. Provide Firmware Receipt or Error

Clients return firmware receipts and errors to the server with an HTTP POST [RFC7231], using an operation path of "/firmware/return". The optional encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned firmware receipt or error. The returned content varies:

- o If the firmware receipt is not digitally signed, the Content-Type is "application/cms" [RFC7193] and the content is the DER-encoded firmware receipt.

- o If the firmware receipt is digitally signed, the Content-Type is "application/cms" and the content is the DER-encoded signed-data encapsulating the firmware receipt.

- o If the firmware error is not digitally signed, the Content-Type is "application/cms" and the content is the DER-encoded firmware error.

- o If the firmware error is digitally signed, the Content-Type is "application/cms" and the content is the DER-encoded signed-data encapsulating the firmware error.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the firmware receipt or firmware error.

6.2.2. Firmware Receipt or Error Response

If the request is successful, the server response MUST have an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server MUST specify either an HTTP 4xx error or an HTTP 5xx error.

If a firmware receipt or firmware error is digitally signed, the server MUST reject it if the digital signature does not validate back to an authorized TA.

7. Trust Anchor Management Protocol

Servers distribute TAMP packages to manage TAs in a client’s trust anchor databases; TAMP packages are defined in [RFC5934]. TAMP will allow the flexibility for a device to load TAs while maintaining an operational state. Unlike other systems that require new software loads when new PKI Roots are authorized for use, TAMP allows for automated management of roots for provisioning or replacement as needed.

Clients MUST authenticate the server, and clients MUST check the server’s authorization.

The server MUST authenticate the client, and the server MUST check the client’s authorization.

The /tamp PC uses an HTTP GET [RFC7231], and the tamp/return PC uses an HTTP POST [RFC7231]. GET is used when the server requests that the client retrieve a TAMP package (see Section 7.1); POST is used when the client provides a confirm (see Section 7.2), provides a response (see Section 7.2), or provides an error (see Section 7.2) for the TAMP package.

7.1. TAMP Status Query, Trust Anchor Update, Apex Trust Anchor Update, Community Update, and Sequence Number Adjust

Clients use the /tamp PC to retrieve the TAMP packages: TAMP Status Query, Trust Anchor Update, Apex Trust Anchor Update, Community Update, and Sequence Number Adjust. Clients can be configured to periodically poll the server for these packages or contact the server based on a PAL entry.

The message flow is as depicted in Figure 3 modulo replacing "Symmetric Key" with the appropriate TAMP message.

7.1.1. Request TAMP Packages

Clients request the TAMP packages from the server with an HTTP GET [RFC7231], using an operation path of "/tamp".

7.1.2. Return TAMP Packages

If the request is successful, the server response MUST have an HTTP 200 response code and a Content-Type of:

- o application/tamp-status-query for TAMP Status Query
- o application/tamp-update for Trust Anchor Update
- o application/tamp-apex-update for Apex Trust Anchor Update
- o application/tamp-community-update for Community Update
- o application/tamp-sequence-adjust for Sequence Number Adjust

As specified in [RFC5934], these content types are digitally signed and clients must support validating the packages directly signed by TAs. For this specification, clients MUST support validation with a certificate and clients MUST reject it if the digital signature does not validate back to an authorized TA.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the TAMP packages.

7.2. TAMP Responses, Confirms, and Errors

Clients return the TAMP Status Query Response, Trust Anchor Update Confirm, Apex Trust Anchor Update Confirm, Community Update Confirm, Sequence Number Adjust Confirm, and TAMP Error to servers, using the /tamp/return PC. Clients can be configured to automatically return responses, confirms, and errors after processing a TAMP package or based on a PAL entry.

The message flow is as depicted in Figure 4 modulo replacing "Receipt/Error" with the appropriate TAMP response, confirm, or error.

7.2.1. Provide TAMP Responses, Confirms, or Errors

Clients provide the TAMP responses, confirms, and errors to the server with an HTTP POST, using an operation path of "/tamp/return". The Content-Type is:

- application/tamp-status-response for TAMP Status Query Response
- application/tamp-update-confirm for Trust Anchor Update Confirm
- application/tamp-apex-update-confirm for Apex Trust Anchor Update Confirm
- application/tamp-community-update-confirm for Community Update Confirm
- application/tamp-sequence-adjust-confirm for Sequence Number Adjust Confirm
- application/tamp-error for TAMP Error

As specified in [RFC5934], these content types should be signed. If signed, a signed-data encapsulates the TAMP content.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the TAMP packages.

7.2.2. TAMP Responses, Confirms, and Error Responses

If the request is successful, the server response MUST have an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server MUST specify either an HTTP 4xx error or an HTTP 5xx error.

If the package is digitally signed, the server MUST reject it if the digital signature does not validate back to an authorized TA.

8. Asymmetric Keys, Receipts, and Errors

[RFC7030] defines the /serverkeygen PC to support server-side generation of asymmetric keys. Keys are returned as either a) an unprotected PKCS #8 when additional security beyond TLS is not employed or b) a CMS asymmetric key package content type that is encapsulated in a signed-data content type that is further encapsulated in an enveloped-data content type when additional security beyond TLS is requested.

Some implementations prefer the use of other CMS content types to encapsulate the asymmetric key package. This document extends the content types that can be returned; see Section 8.1.

[RFC7191] defines content types for key package receipts and errors. This document defines the /serverkeygen/return PC to add support for returning receipts and errors for asymmetric key packages; see Section 8.2.

PKCS #12 [RFC7292] (sometimes referred to as "PFX" (Personal Information Exchange) or "P12") is often used to distribute asymmetric private keys and associated certificates. This document extends the /serverkeygen PC to allow servers to distribute server-generated asymmetric private keys and the associated certificate to clients using PKCS #12; see Section 8.3.

8.1. Asymmetric Key Encapsulation

CMS supports a number of content types to encapsulate other CMS content types; [RFC7030] includes one such possibility. Note that when only relying on TLS the returned key is not a CMS content type. This document extends the CMS content types that can be returned.

If the client supports CCC [RFC6010], then the client can indicate that it supports encapsulated asymmetric keys in the encrypted key package [RFC5958] by including the encrypted key package’s OID in a content type attribute [RFC2985] in the CSR (Certificate Signing Request) -- aka the certification request -- that it provides to the server. If the client knows a priori that the server supports the encrypted key package content type, then the client need not include the content type attribute in the CSR.

In all instances defined herein, the Content-Type is "application/cms" [RFC7193]. The optional encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned asymmetric key package.

If additional encryption and origin authentication are employed, the content associated with application/cms is a DER-encoded signed-data that encapsulates an enveloped-data that encapsulates a signed-data that further encapsulates an asymmetric key package.

If CCC is supported and additional encryption is employed, the content associated with application/cms is a DER-encoded encrypted key package [RFC6032] content type that encapsulates a signed-data that further encapsulates an asymmetric key package.

If CCC is supported and if additional encryption and additional origin authentication are employed, the content associated with application/cms is a DER-encoded signed-data that encapsulates an encrypted key package content type that encapsulates a signed-data that further encapsulates an asymmetric key package.

The encrypted key package [RFC6032] provides three choices to encapsulate keys: EncryptedData, EnvelopedData, and AuthEnvelopedData, with EnvelopedData being the mandatory-to-implement choice.

When rejecting a request, the server specifies either an HTTP 4xx error or an HTTP 5xx error.

If an asymmetric key package or an encrypted key package is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

Note: Absent a policy on the client side requiring a signature, a malicious EST server can simply strip the signature, thus bypassing that check. In that case, this requirement is merely a sanity check, serving to detect mis-signed packages or misconfigured clients.

[RFC3370], [RFC5753], [RFC5754], [RFC6033], [RFC6161], and [RFC6162] provide algorithm details for use when protecting the asymmetric key package and encrypted key package.

8.2. Asymmetric Key Package Receipts and Errors

Clients can be configured to automatically return receipts after processing an asymmetric key package, return receipts based on processing of the key-package-identifier-and-receipt-request attribute [RFC7191], or return receipts when prompted by a PAL entry. Servers can indicate that clients return a receipt by including the key-package-identifier-and-receipt-request attribute [RFC7191] in a signed-data as a signed attribute.

The protocol flow is identical to that depicted in Figure 4 modulo the receipt or error is for asymmetric keys.

The server and client processing is as described in Sections 5.2.1 and 5.2.2 modulo the PC, which, for Asymmetric Key Packages, is "/serverkeygen/return".

8.3. PKCS #12

PFX is widely deployed and supports protecting keys in the same fashion as CMS, but it does so differently.

8.3.1. Server-Side Key Generation Request

Similar to the other server-generated asymmetric keys provided through the /serverkeygen PC:

- o The certificate request is HTTPS POSTed and is the same format as for the "/simpleenroll" and "/simplereenroll" path extensions with the same content type.
- o In all respects, the server SHOULD treat the CSR as it would any enroll or re-enroll CSR; the only distinction here is that the server MUST ignore the public key values and signature in the CSR. These are included in the request only to allow the reuse of existing codebases for generating and parsing such requests.

PBE (password-based encryption) shrouding of PKCS #12 is supported, and this specification makes no attempt to alter this de facto standard. As such, there is no support of the DecryptKeyIdentifier specified in [RFC7030] for use with PKCS #12 (i.e., "enveloping" is not supported). Note: The use of PBE requires that the password be distributed to the client; methods to distribute this password are beyond the scope of this document.

8.3.2. 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 "application/pkcs12" [PKCS12] that consists of a base64-encoded DER-encoded [X.690] PFX [RFC7292].

Note that this response is different than the response returned as described in Section 4.4.2 of [RFC7030], because here the private key and the certificate are included in the same PFX.

When rejecting a request, the server MUST specify either an HTTP 4xx error or an HTTP 5xx error. The response data’s Content-Type MAY be "text/plain" [RFC2046] to convey human-readable error messages.

9. PAL and Certificate Enrollment

The /fullcmc PC is defined in [RFC7030]; the CMC (Certificate Management over Cryptographic Message Syntax) requirements and packages are defined in [RFC5272], [RFC5273], [RFC5274], and [RFC6402]. This section describes PAL interactions.

Under normal circumstances, the client-server interactions for PKI enrollment are as follows:

```
           Client                       Server
                 --------------------->
             POST req: PKIRequest
             Content-Type: application/pkcs10
            or
             POST req: PKIRequest
             Content-Type: application/pkcs7-mime
                            smime-type=CMC-request

                  <--------------------
                         POST res: PKIResponse
                         Content-Type: application/pkcs7-mime
                                        smime-type=certs-only
                        or
                         POST res: PKIResponse
                         Content-Type: application/pkcs7-mime
                                        smime-type=CMC-response
```

If the response is rejected during the same session:

```
           Client                       Server
                  --------------------->
              POST req: PKIRequest
              Content-Type: application/pkcs10
             or
              POST req: PKIRequest
              Content-Type: application/pkcs7-mime
                             smime-type=CMC-request

                  <--------------------
                         POST res: empty
                         HTTPS Status Code
                        or
                         POST res: PKIResponse
                         Content-Type: application/pkcs7-mime
                                        smime-type=CMC-response
```

   If the request is to be filled later:

```
          Client                       Server
                 --------------------->
             POST req: PKIRequest
             Content-Type: application/pkcs10
            or
             POST req: PKIRequest
             Content-Type: application/pkcs7-mime
                           smime-type=CMC-request

                 <--------------------
                        POST res: empty
                        HTTPS Status Code
                        + Retry-After
                       or
                        POST res: PKIResponse (pending)
                        Content-Type: application/pkcs7-mime
                                      smime-type=CMC-response

                 --------------------->
             POST req: PKIRequest (same request)
             Content-Type: application/pkcs10
            or
             POST req: PKIRequest (CMC Status Info only)
             Content-Type: application/pkcs7-mime
                           smime-type=CMC-request

                 <--------------------
                        POST res: PKIResponse
                        Content-Type: application/pkcs7-mime
                                      smime-type=certs-only
                       or
                        POST res: PKIResponse
                        Content-Type: application/pkcs7-mime
                                      smime-type=CMC-response
```

With the PAL, the client begins after pulling the PAL and a Start Issuance PAL package type, essentially adding the following before the request:

```
        Client                       Server
              --------------------->
          GET req: PAL
               <--------------------
                      GET res: PAL
                      Content-Type: application/xml
```

The client then proceeds as above with a simple PKI enrollment or a full CMC enrollment, or it begins enrollment assisted by a CSR:

```
        Client                       Server
              --------------------->
          GET req: DS certificate with CSR

               <--------------------
                      GET res: PAL
                      Content-Type: application/csrattrs
```

For immediately rejected requests, CMC works well. If the server prematurely closes the connection, then the procedures in Section 6.3.1 of [RFC7230] apply. But this might leave the client and server in a different state. The client could merely resubmit the request, but another option, documented herein, is for the client to instead download the PAL to see if the server has processed the request. Clients might also use this process when they are unable to remain connected to the server for the entire enrollment process; if the server does not or is not able to return a PKIData indicating a status of pending, then the client will not know whether the request was received. If a client uses the PAL and reconnects to determine if the certification or rekey or renew request was processed:

o Clients MUST authenticate the server, and clients MUST check the server’s authorization.

o The server MUST authenticate the client, and the server MUST check the client’s authorization.

o Clients retrieve the PAL, using the /pal URI.

o Clients and servers use the operation path of "/simpleenroll", "simplereenroll", or "/fullcmc", based on the PAL entry, with an HTTP GET [RFC7231] to get the success or failure response.

Responses are as specified in [RFC7030].

10. Security Considerations

This document relies on many other specifications; however, all of the security considerations in [RFC7030] apply. Refer also to the following:

- For HTTP, HTTPS, and TLS security considerations, see [RFC7231], [RFC2818], and [RFC5246].

- For URI security considerations, see [RFC3986].

- For content type security considerations, see [RFC4073], [RFC4108], [RFC5272], [RFC5652], [RFC5751], [RFC5934], [RFC5958], [RFC6031], [RFC6032], [RFC6268], [RFC6402], [RFC7191], and [RFC7292].

- For algorithms used to protect packages, see [RFC3370], [RFC5649], [RFC5753], [RFC5754], [RFC5959], [RFC6033], [RFC6160], [RFC6161], [RFC6162], and [RFC7192].

- For random numbers, see [RFC4086].

- For server-generated asymmetric key pairs, see [RFC7030].

11. IANA Considerations

IANA has created the "PAL Package Types" registry and performed three registrations: PAL Name Space, PAL XML Schema, and PAL Package Types.

11.1. PAL Name Space

This section registers a new XML namespace [XMLNS], "urn:ietf:params:xml:ns:pal", per the guidelines in [RFC3688]:

```
   URI: urn:ietf:params:xml:ns:pal
   Registrant Contact: Sean Turner (sean@sn3rd.com)
   XML:
      BEGIN
         <?xml version="1.0"?>
         <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
            "https://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
         <html xmlns="https://www.w3.org/1999/xhtml" xml:lang="en">
         <head>
            <title>Package Availability List</title>
         </head>
         <body>
            <h1>Namespace for Package Availability List</h1>
            <h2>urn:ietf:params:xml:ns:pal</h2>
            <p>See RFC 8295</p>
         </body>
         </html>
      END
```

11.2. PAL XML Schema

This section registers an XML schema as per the guidelines in [RFC3688].

```
   URI: urn:ietf:params:xml:schema:pal
   Registrant Contact: Sean Turner (sean@sn3rd.com)
   XML: See Section 2.1.2.
```

11.3. PAL Package Types

IANA has created a new registry named "PAL Package Types". This registry is for PAL package types whose initial values are found in Section 2.1.1. Future registrations of PAL package types are subject to Expert Review, as defined in RFC 8126 [RFC8126]. Package types MUST be paired with a media type; package types specify the path components to be used that in turn specify the media type used.

12. References

12.1. Normative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, DOI 10.17487/RFC2585, May 1999, <https://www.rfc-editor.org/info/rfc2585>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <https://www.rfc-editor.org/info/rfc2985>.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, DOI 10.17487/RFC3370, August 2002, <https://www.rfc-editor.org/info/rfc3370>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/info/rfc3394>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4073] Housley, R., "Protecting Multiple Contents with the Cryptographic Message Syntax (CMS)", RFC 4073, DOI 10.17487/RFC4073, May 2005, <https://www.rfc-editor.org/info/rfc4073>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, DOI 10.17487/RFC4514, June 2006, <https://www.rfc-editor.org/info/rfc4514>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/info/rfc5272>.

[RFC5273] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC): Transport Protocols", RFC 5273, DOI 10.17487/RFC5273, June 2008, <https://www.rfc-editor.org/info/rfc5273>.

[RFC5274] Schaad, J. and M. Myers, "Certificate Management Messages over CMS (CMC): Compliance Requirements", RFC 5274, DOI 10.17487/RFC5274, June 2008, <https://www.rfc-editor.org/info/rfc5274>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://www.rfc-editor.org/info/rfc5649>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, DOI 10.17487/RFC5753, January 2010, <https://www.rfc-editor.org/info/rfc5753>.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <https://www.rfc-editor.org/info/rfc5754>.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, DOI 10.17487/RFC5934, August 2010, <https://www.rfc-editor.org/info/rfc5934>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC5959] Turner, S., "Algorithms for Asymmetric Key Package Content Type", RFC 5959, DOI 10.17487/RFC5959, August 2010, <https://www.rfc-editor.org/info/rfc5959>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <https://www.rfc-editor.org/info/rfc5967>.

[RFC6010] Housley, R., Ashmore, S., and C. Wallace, "Cryptographic Message Syntax (CMS) Content Constraints Extension", RFC 6010, DOI 10.17487/RFC6010, September 2010, <https://www.rfc-editor.org/info/rfc6010>.

[RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, DOI 10.17487/RFC6031, December 2010, <https://www.rfc-editor.org/info/rfc6031>.

[RFC6032] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6032, DOI 10.17487/RFC6032, December 2010, <https://www.rfc-editor.org/info/rfc6032>.

[RFC6033] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6033, DOI 10.17487/RFC6033, December 2010, <https://www.rfc-editor.org/info/rfc6033>.

[RFC6160] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Protection of Symmetric Key Package Content Types", RFC 6160, DOI 10.17487/RFC6160, April 2011, <https://www.rfc-editor.org/info/rfc6160>.

[RFC6161] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6161, DOI 10.17487/RFC6161, April 2011, <https://www.rfc-editor.org/info/rfc6161>.

[RFC6162] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Asymmetric Key Package Content Type", RFC 6162, DOI 10.17487/RFC6162, April 2011, <https://www.rfc-editor.org/info/rfc6162>.

[RFC6268] Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", RFC 6268, DOI 10.17487/RFC6268, July 2011, <https://www.rfc-editor.org/info/rfc6268>.

[RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <https://www.rfc-editor.org/info/rfc6402>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <https://www.rfc-editor.org/info/rfc7303>.

[RFC7191] Housley, R., "Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7191, DOI 10.17487/RFC7191, April 2014, <https://www.rfc-editor.org/info/rfc7191>.

[RFC7192] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7192, DOI 10.17487/RFC7192, April 2014, <https://www.rfc-editor.org/info/rfc7192>.

[RFC7193] Turner, S., Housley, R., and J. Schaad, "The application/cms Media Type", RFC 7193, DOI 10.17487/RFC7193, April 2014, <https://www.rfc-editor.org/info/rfc7193>.

[RFC7230] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7292] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, DOI 10.17487/RFC7292, July 2014, <https://www.rfc-editor.org/info/rfc7292>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[XMLSCHEMA]
Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <https://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/en>.

12.2. Informative References

[PKCS12] IANA, "PKCS #12", <https://www.iana.org/assignments/media-types/application/pkcs12>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[XMLNS] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208/, December 2009, <https://www.w3.org/TR/2009/REC-xml-names-20091208/>.

Appendix A. Example Use of PAL

This is an informative appendix. It includes examples of protocol flows.

Steps for using a PAL include the following:

1. Access PAL

2. Process PAL entries
   2.1. Get CA certificates
   2.2. Get CRLs
   2.3. Get CSR attributes
   2.4. Enroll: simple enrollment, re-enrollment, or full CMC
   2.5. Get Firmware, TAMP, Symmetric Keys, or EE certificates

```
   Client                      Server
         --------------------->                     -+
   GET req:                                          | /pal
         <---------------------                      |
                       GET res: PAL                  |
                       Content-Type: application/xml |
                                                     |
         --------------------->                     -+
   GET req:                                          | /cacerts
         <---------------------                      |
               GET res: CA Certificates              |
               Content-Type: application/pkcs7-smime |
                             smime-type=certs-only   |
                                                     |
         --------------------->                     -+
   GET req:                                          | /crls
         <---------------------                      |
               GET res: CRLs                         |
               Content-Type: application/pkcs7-smime |
                             smime-type=crls-only    |
                                                     |
         --------------------->                     -+
   GET req:                                          | /csrattrs
         <---------------------                      |
                           GET res: attributes       |
         --------------------->                     -+
   POST req: PKIRequest                              | /simpleenroll and
   Content-Type: application/pkcs10                  | /simplereenroll
                                                     |
   Content-Type: application/pkcs7-mime              | /fullcmc
                 smime-type=CMC-request              |
                                                     |
         <--------------------                       |
              (success or failure)                   |
              POST res: PKIResponse                  | /simpleenroll
              Content-Type: application/pkcs7-mime   | /simplereenroll
                            smime-type=certs-only    | /fullcmc
                                                     |
              Content-Type: application/pkcs7-mime   | /fullcmc
                            smime-type=CMC-response  |
                                                     |
         -------------------->                      -+
   GET req:                                          | /firmware
         <--------------------                       | /tamp
               GET res: Firmware, TAMP Query         | /symmetrickeys
                        + Updates, Symmetric Keys    |
                Content-Type: application/cms        |
                                                     |
         --------------------->                     -+
   POST res: Firmware Receipts or Errors,            | /firmware/return
   TAMP Response or Confirms or Errors,              | /tamp/return
   Symmetric Key Receipts or Errors                  | /symmetrickeys/
                                                     |      return
                                                     |
   Content-Type: application/cms                     |
         <--------------------                       |
               POST res: empty                       |
                (success or failure)                 |
         -------------------->                      -+
   GET req:                                          | /eecerts
         <--------------------                       |
               GET res: Other EE certificates        |
                Content-Type: application/pkcs7-mime |
                              smime-type=certs-only  |
```

The figure above shows /eecerts after /*/return, but this is for illustrative purposes only.

Appendix B. Additional CSR Attributes

This is an informative appendix.

In some cases, the client is severely limited in its ability to encode and decode ASN.1 objects. If the client knows that a "csr" template is being provided during enrollment, then it can peel the returned CSR attribute, generate its keys, place the public key in the certification request, and then sign the request. To accomplish this, the server returns a pKCS7PDU attribute [RFC2985] in the /csrattrs (the following is "pseudo ASN.1" and is only meant to show the fields needed to accomplish returning a template certification request):

```
  pKCS7PDU ATTRIBUTE ::= {
    WITH SYNTAX ContentInfo
    ID pkcs-9-at-pkcs7PDU
    }

  pkcs-9-at-pkcs7PDU OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    pkcs-9-at(25) 5
    }
```

The ContentInfo is a PKIData:

```
  PKIData ::= SEQUENCE {
    reqSequence        SEQUENCE SIZE(0..MAX) OF TaggedRequest
    }
```

Where TaggedRequest is a choice between the PKCS #10 or Certificate Request Message Format (CRMF) requests.

```
  TaggedRequest ::= CHOICE {
    tcr               [0] TaggedCertificationRequest,
    crm               [1] CertReqMsg
    }
```

Or, the ContentInfo can be a signed-data content type that further encapsulates a PKIData.

Acknowledgements

Thanks in no particular order go to Alexey Melnikov, Paul Hoffman, Brad McInnis, Max Pritikin, Francois Rousseau, Chris Bonatti, and Russ Housley for taking time to provide comments.

Author’s Address

Sean Turner
sn3rd

Email: sean@sn3rd.com