Network Working Group
Internet Draft
Intended Status: Standards Track
Expires: April 15, 2016

Sean Turner
IECA

October 13, 2015

EST Extensions
draft-turner-est-extensions-03.txt

Abstract

The EST (Enrollment over Secure Transport) protocol defined a Well-Known URI (Uniform Resource Identifier): /.well-known/est. EST also defined several path components that clients use for PKI (Public Key Infrastructure) services, namely certificate enrollment (e.g., /simpleenroll). In some sense, the services provided by the path components can be thought of as PKI management-related packages. There are additional PKI-related packages a client might need as well as other security-related packages, such as firmware, trust anchors, and symmetric, asymmetric, and encrypted keys. This document also specifies the PAL (Package Availability List), which is an XML (Extensible Markup Language) file 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

Copyright Notice

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 PKI (Public Key Infrastructure) related services with path components such as simple enrollment with /simpleenroll, rekey/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 (Path Components). 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 path component (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], which include /csrattrs, /fullcmc, /simpleenroll, /simplereenroll, and /cacerts. The /pal PC is described in Section 2.

- /eecerts - EE (End-Entity) certificates 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) are also needed by the client when they validate certificate paths. CRLs 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 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. The /symmetrickeys PC is described in Section 5.

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

- /tamp - To control the TAs in client trust anchor database,

servers use the /tamp PC to request that clients retrieve a TAMP query, update, and adjust packages and clients use the same PC to return response, confirm, and error packages. The /tamp PC is defined in Section 7.

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

- Returning asymmetric key package receipts and errors.
- Encapsulating returned asymmetric keys in additional CMS content types.
- Returning server-generated public key pairs encapsulated in PKCS#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 Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages [RFC4108], Certificate Management over CMS (CMC) [RFC5272], Cryptographic Message Syntax (CMS) Encrypted Key Package [RFC6032], Cryptographic Message Syntax (CMS) [RFC5652][RFC6268], Trust Anchor Management Protocol (TAMP) [RFC5934], Cryptographic Message Syntax (CMS) Content Constraints Extension [RFC6010], CMS Symmetric Key Package Content Type [RFC6031], Enrollment over Secure Transport protocol [RFC7030], CMS Key Package Receipt and Error Content Types [RFC7191] is assumed. Also, familiarity with the CMS protecting content types signed data and encrypted data is assumed; CMS signed data and encrypted data are defined in [RFC5652] and 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 an air gap [RFC4949] between themselves and the server; interactions between the agent and client in the last cases are beyond the scope of this document. Before an agent can service clients, the agent must have a trust relationship with the server, 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 end-point 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 CMS content types. There are numerous types of packages: Asymmetric Keys, Symmetric Keys, Encrypted Keys, CRLs, Public Key Certificate Management, Firmware, Public Key Certificates, and TAMP packages. All of these packages except the public key certificates and CRLs, which are already digitally signed, are digitally signed and encapsulated in a CMS signed data [RFC5652][RFC6268]; Firmware receipts and errors, TAMP responses, confirms, and errors, as well as Key Package receipts and errors can be optionally signed. Certificate and CRLs are included in a package that uses signed data, which is often referred to as a degenerate CMS or "certs-only" or "crls-only" message [RFC5751][RFC6268], but no signature or content is present; hence the name certs-only and crls-only.

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 of each of the PCs defined by this document.

The requirements for the TA database is as specified in [RFC7030] as well.

1.3. TLS Cipher Suites

TLS cipher suite 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. Content-Transfer-Encoding

A Content-Transfer encoding of "base64" [RFC2045] is used for all client server interactions.

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 [RFC2119].

2. Locate Available Packages

The PAL (Package Availability List) is an XML (Extensible Markup Language) [XML] file that furnishes information for packages that are currently available and authorized for retrieval by a client. It provides client specific:

- 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.

A client can use this service to determine all of the security-related products for bootstrapping or to periodically poll the server in order to determine if there are updated packages available for it.

To get the /pal PC, the client and server need to mutually authenticate each other with TLS and authorize each other. Clients retrieve their PAL and processes it to determine the packages available for it.

```
                      |                      |
            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

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

The server MUST authenticate the client as specified in [RFC7030] and the server MUST verify client 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 (i.e., minOccurs=0) or more entries, each of which is composed of the following four elements all of which MUST be present (i.e., minOccurs=1):

- The <type> element uniquely identifies each package that a client may retrieve from the server with a 4-digit field. The PAL Package Types are defined in Section 2.1.1.

- The <date> element is a 20-character field that contains either:

  - The date and time (expressed as Generalized Time: YYYY-MM-DDTHH:MM:SSZ) that the client last successfully downloaded the identified package from the server, or

  - 0001-01-01T00:00:00Z (i.e., 0), if:

    - There is no indication 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).

- The <size> element indicates the size in bytes of the package. A package size of zero (i.e., "0" without the quotes) indicates that the client needs to begin a transaction or return an error or receipt.

- The <info> element provides either an SKI (Subject Key Identifier), DN (Distinguished Name), Issuer and Serial Number tuple or a URI. When a URI is included it indicates the location where the identified package can be retrieved. When a DN, SKI, or 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/renewed).

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 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 clients.

When the <date> element is not zero (i.e., 0001-01-01T00:00:00Z) it MUST be represented in a form that matches the dateTime production in "canonical representation" [XMLSCHEMA]. Implementations SHOULD NOT rely on time resolution finer than seconds and MUST NOT generate time instants that specify leap seconds.

2.1.1. PAL Package Types

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

NOTE: DS is Digital Signature and KE is Key Establishment.

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
0007:	DS certificate enrollment (success)
0008:	DS certificate enrollment (failure)
0009:	Start DS certificate re-enrollment
0010:	DS certificate re-enrollment (success)
0011:	DS certificate re-enrollment (failure)
0012:	Start KE certificate enrollment
0013:	KE certificate enrollment (success)
0014:	KE certificate enrollment (failure)
0015:	Start KE certificate re-enrollment
0016:	KE certificate re-enrollment (success)
0017:	KE certificate re-enrollment (failure)

0018:	Asymmetric Key Package (PKCS#8)
0019:	Asymmetric Key Package (CMS)
0020:	Asymmetric Key Package (PKCS#12)
0021:	Asymmetric Key Package Receipt or Error
0022:	Symmetric Key Package
0023:	Symmetric Key Package Receipt or Error
0024:	Firmware Package
0025:	Firmware Package Receipt or Error
0026:	TAMP Status Query
0027:	TAMP Status Query Response or Error
0028:	Trust Anchor Update
0029:	Trust Anchor Update Confirm or Error
0030:	Apex Trust Anchor Update
0031:	Apex Trust Anchor Update Confirm or Error
0032:	Community Update
0033:	Community Update Confirm or Error
0034:	Sequence Number Adjust
0035:	Sequence Number Adjust Confirm or Error

Table 1 - PAL Package Types

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 or error.

The PAL package types have the following meaning:

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.

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 certificate [RFC5280] is 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 ARL (Authority Revocation List) [RFC5280] is 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 CRL (Certificate Revocation List) [RFC5280] is 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 (Digital Signature) certificate enrollment: Indicates that the client begin enrolling their DS certificate. The PAL entry points to a /csrattrs URI, which is defined in [RFC7030].

0007 DS certificate enrollment (success): Indicates that the client retrieve a successful certification response. The PAL entry points to a /simpleenroll or a /fullcmc URI, which are both defined in [RFC7030].

0008 DS certificate enrollment (failure): Indicates that the client retrieve a failed certification response for a DS certificate. This PAL entry points to a /simpleenroll or a /fullcmc URI.

0009 Start DS certificate re-enrollment: Indicates that the client rekey/renew a DS certificate.   The PAL entry points to a /simplereenroll or a /fullcmc URI.

0010 DS certificate re-enrollment (success): See PAL package type 0007.

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

NOTE: The KE (Key Establishment) responses that follow use the same URIs as DS certificates.

0012 Start KE certificate enrollment: See PAL package type 0006.

0013 KE certificate enrollment (success): See PAL package type 0007.

0014 KE certificate enrollment (failure): See PAL package type 0008.

0015 Start KE certificate re-enrollment: See PAL package type 0009.

0016 KE certificate re-enrollment (success): See PAL package type

0007.

0017 KE certificate re-enrollment (failure): See PAL package type 0008.

Note: The variations on the asymmetric key packages is 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 not the private key info type). See Section 8 of this document for additional information. See Section 9 for additional information about server key generation using the /fullcmc URI.

0018 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].

0019 Asymmetric Key Package (CMS): See PAL package type 0018. 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].

0020 Asymmetric Key Package (PKCS#12): See PAL package type 0018. The difference being that the package available is PKCS12 [RFC7292] content type. See Section 8 of this document.

0021 Asymmetric Key Package Receipt or Error: See PAL package type 0018. 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].

0022 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.

0023 Symmetric Key Package Receipt or Error: Indicates that the server wants the client to return a key package receipt or an error [RFC7191] to the /symmetrickeys/return URI, which is defined in Section 5.

0024 Firmware Package: Indicates that a firmware package [RFC4108] is

available for the client using the /firmware URI, which is defined in Section 6.

0025 Firmware Package Receipt or Error: Indicates that the server wants the client to return a firmware package receipt or error [RFC4108] using the /firmware/return URI, which is defined in Section 6.

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

0026 TAMP Status Query: Indicates that a TAMP Status Query package [RFC5934] is available for the client using the /tamp URI.

0027 TAMP Status Query Response or Error: Indicates that the server wants the client to return a TAMP Status Query Response or Error [RFC5934] using the /tamp/return URI.

0028 Trust Anchor Update: Indicates that a Trust Anchor Update package [RFC5934] is available for the client using the /tamp URI.

0029 Trust Anchor Update Confirm or Error: Indicates that the server wants the client to return a TAMP Anchor Update Confirm or Error [RFC5934] using the /tamp/return URI.

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

0031 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] using the /tamp/return URI.

0032 Community Update: Indicates that a TAMP Community Update package [RFC5934] is available for the client using the /tamp URI.

0033 Community Update Confirm or Error: Indicates that the server wants the client to return a Community Update Confirm or Error [RFC5934] using the /tamp/return URI.

0034 Sequence Number Adjust: Indicates that a TAMP Sequence Number package [RFC5934] is available for the client using the /tamp URI.

0035 Sequence Number Adjust Confirm or Error: Indicates that the server wants the client to return a Sequence Number Adjust Confirm or Error [RFC5934] using the /tamp/return URI.

2.1.2. PAL Schema

The name space 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="http://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.
    </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">
        <xsd:annotation>
          <xsd:documentation>
            Contains information about the package and a link that
            the client uses to download 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"
           minOccurs="1" maxOccurs="1">
         </xsd:element>
         <xsd:element name="date" type="pal:GeneralizedTimeType"
           minOccurs="1" maxOccurs="1">
         </xsd:element>
         <xsd:element name="size" type="pal:PackageSizeType"
           minOccurs="1" maxOccurs="1">
         </xsd:element>
         <xsd:element name="info" type="pal:PackageInfoType"
           minOccurs="1" maxOccurs="1">
         </xsd:element>
       </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:integer" />
       </xsd:sequence>
     </xsd:complexType>

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

     <xsd:simpleType name="PackageType">
```

```
       <xsd:annotation>
         <xsd:documentation>
           Identifies each package that a client may retrieve from
           the server with a 4-digit field.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:string">
         <xsd:maxLength value="4" />
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="GeneralizedTimeType">
       <xsd:annotation>
         <xsd:documentation>
           Indicates the date and time (YYYY-MM-DDTHH:MM:SSZ) the
           client last acknowledged successful receipt of the
           package or 0001-01-01T00:00:00Z if there is no indication
           the package has been downloaded or the PAL entry
           corresponds to a pointer to the next PAL.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:restriction base="xsd:dateTime">
         <xsd:pattern value=
           "((000[1-9])|(00[1-9][0-9])|(0[1-9][0-9]{2})|
           ([1-9][0-9]{3}))-((0[1-9])|(1[012]))-((0[1-9])|
           ([12][0-9])|(3[01]))T(([01][0-9])|(2[0-3]))
           ((:[0-5][0-9])(:[0-5][0-9])Z" />
         <xsd:minInclusive value="2013-05-23T00:00:00Z" />
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="PackageSizeType">
       <xsd:annotation>
         <xsd:documentation>
           Indicates the package's size.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:pattern value="[0-9]+" />
     </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: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: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:simpleType>

</xsd:schema>
```

2.2. Request PAL

Clients request their PAL with an HTTP GET [RFC7231] using an operation path of "/pal".

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] and a Content-Transfer-Encoding of "base64".

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

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

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

- /simpleenroll, /simplereenroll, and /fullcmc packages items 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 a certificate management SHOULD accept and process CA-provided transactions as soon as possible to avoid undue delays that might lead to protocol failure.

- /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 to begin a certificate management protocol with a client, the server will provide a notice for one at a time. Pending notices will be serviced in order of the earliest date when the certificate will be used.

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 are already 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 is identical to that in Section 3.1.3 of [RFC7030] except that the certificate provided is not the one issued to the client but is instead one of more client's peer certificates is returned in the certs-only message.

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

4. Distribute CRLs

CRLs 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 at the time of bootstrap would obviate the need for the client to parse certificate and aid 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 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 is already signed by a CA (i.e., the CRLs in a certs-only message are already 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 processing of the response is identical to that in Section 3.1.3 of [RFC7030] except that instead of providing the issued certificate one of more CRLs are returned in the certs-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 client to provide errors and receipts.

Clients MUST authenticate the server and clients MUST check 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 path component; 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 path component.

5.1. Symmetric Keys

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

The TLS cipher suite used to return the symmetric key MUST offer commensurate cryptographic strength with the symmetric key being delivered to the client. As with the /serverkeygen PC defined in [RFC7030], the default distribution method of 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 commensurate cryptographic strength with the symmetric key being delivered to the client. The cipher suite use MUST NOT have 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) |
                      |--------------------->|
                      |<---------------------|
                      | Res 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] and a Content-Transfer-Encoding of "base64". The optional application/cms parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned symmetric key. The returned content varies:

- If additional encryption is not being employed, the content associated with application/cms is a base 64-encoded DER-encoded [X.690] symmetric key package.

- If additional encryption is employed, the content associated with application/cms is a base 64-encoded DER-encoded enveloped data that encapsulates a signed data that further encapsulates a symmetric key package.

- If additional encryption and origin authentication is employed, the content associated with application/cms is a base 64-encoded DER-encoded signed data that encapsulates an enveloped data that encapsulates a signed data that further encapsulates a symmetric key package.

- If CCC (CMS Content Constraints) [RFC6010] is supported and additional encryption is employed, the content associated with application/cms is a base 64-encoded DER-encoded encrypted key package content type [RFC6032] that encapsulates a signed data that further encapsulates a symmetric key package.

- If CCC is supported and additional encryption and additional origin authentication is employed, the content associated with application/cms is a base 64-encoded DER-encoded signed data that encapsulates an encrypted key package content type that encapsulates a signed data that further encapsulates a symmetric key package.

Encrypted key package provides three choices to encapsulate keys, encrypted data, enveloped data, and authenticated data, with enveloped data being the mandatory to implement choice. 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 is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

[RFC3370], [RFC5753], [RFC5754], [RFC6033], [RFC6160], and [RFC6161] provide algorithm details for use when protecting the symmetric 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 are 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 are 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 key receipts and errors to the server with an HTTP POST [RFC7231] using an operation path of "/symmetrickeys/return" and a Content-Transfer-Encoding of "base64". The returned content varies:

- 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.

- If the key package error is not digitally signed, the Content-Type is application/cms and the associated content is 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 200 response code with no content.

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, Firmware Receipts, and Firmware Errors

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

Clients MUST authenticate the server and clients MUST check server's authorization.

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] and a Content-Transfer-Encoding of "base64". The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned firmware. The returned content varies:

- If the firmware is unprotected, then the Content-Type is application/cms and the content is the base 64-encoded DER-encoded [X.690] firmware package.

- If the firmware is encrypted, then the Content-Type is application/cms and the content is the base 64-encoded DER-encoded encrypted data that encapsulates the firmware package.

- If the firmware is signed, then the Content-Type is application/cms and the content is the base 64-encoded DER-encoded signed data that encapsulates the firmware package.

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. 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 Package Receipt or Error

Clients return firmware package receipts and errors to the server with an HTTP POST [RFC7231] using an operation path of "/firmware/return" and a Content-Transfer-Encoding of "base64". The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned firmware. The returned content varies:

- If the firmware load receipt is not digitally signed, the Content-Type is application/cms [RFC7193] and the content is the base 64-encoded DER-encoded firmware load receipt.

- If the firmware load receipt is digitally signed, the Content-Type is application/cms and the content is the base 64-encoded DER-encoded signed data encapsulating the firmware load receipt.

- If the firmware load error is not digitally signed, the Content-Type is application/cms and the content is the base 64-encoded DER-encoded firmware load error.

- If the firmware load error is digitally signed, the Content-Type is application/cms and the content is the base 64-encoded DER-encoded signed data encapsulating the firmware load error.

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

6.2.2. Firmware Receipt or Error Response

If the request is successful, the server response MUST have an HTTP 200 response code with no content.

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

If a firmware load receipt or firmware load 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 client trust anchor databases; TAMP packages are defined in [RFC5934]. TAMP will allow the flexibility for a device to load authorities 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 server's authorization.

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 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 with Content-Transfer-Encoding of "base64" 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, client 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 Response, Confirm, and Errors Packages

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. Return Responses, Confirms, and 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-Transfer-Encoding is "base64" and the Content-Type is:

- o application/tamp-status-query-response for TAMP Status Query Response
- o application/tamp-update-confirm for Trust Anchor Update Confirm
- o application/tamp-apex-update-confirm for Apex Trust Anchor Update Confirm
- o application/tamp-community-update-confirm for Community Update Confirm
- o application/tamp-sequence-adjust-confirm for Sequence Number Adjust Confirm
- o 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. Responses, Confirms, and Errors Response

If the request is successful, the server response MUST have an HTTP 200 response code with no content.

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 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 either as an unprotected PKCS#8 when additional security beyond TLS is not employed or as 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 in Section 8.1.

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

PKCS#12 [RFC7292], sometimes referred to as "PFX" (Personal inFormation eXchange), "P12", and "PKCS#12" files, are often used to distribute asymmetric private keys and the associated certificate. This document extends the /serverkeygen PC to allow servers to distribute server-generated asymmetric private keys and the associated certificate to clients in 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 content type attribute [RFC2985] in the CSR (Certificate Signing Request), aka the certification request, it provides to the server. If the server knows a prior that the client 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 Content-Transfer-Encoding is "base64". The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned asymmetric key package.

If additional encryption and origin authentication is employed, the content associated with application/cms is a base 64-encoded DER-encoded signed data that encapsulates an enveloped data that encapsulates a signed data that further encapsulates an asymmetric key package.

If CCC (CMS Content Constraints) is supported and additional encryption is employed, the content associated with application/cms is a base 64-encoded DER-encoded encrypted key package content type that encapsulates a signed data that further encapsulates an asymmetric key package.

If CCC is supported and additional encryption and additional origin authentication is employed, the content associated with application/cms is a base 64-encoded DER-encoded signed data that encapsulates an encrypted key package content type that encapsulates a signed data that further encapsulates an asymmetric key package.

Encrypted key package provides three choices to encapsulate keys, encrypted data, enveloped data, and authenticated data, with enveloped data being the mandatory to implement choice.

8.2. Asymmetric Key Package Receipts and Errors

Clients are 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 Section 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:

- The certificate request is HTTPS POSTed and is the same format as for the "/simpleenroll" and "/simplereenroll" path extensions with the same content-type and transfer encoding.

- 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 re-use of existing codebases for generating and parsing such requests.

In the wild, only the PBE (password based encryption) shrouding of PKCS#8 is supported and this specification makes no attempt to alter this defacto standard. As such there is no support of the DecryptKeyIdentifier specified in [RFC7030] for use with PKCS#12.

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" that consists of a base64-encoded DER-encoded [X.690] PFX [RFC7292] with a Content-Transfer-Encoding of "base64".

Note that this response is different than the response returned 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. If the content-type is not set, the response data MUST be a plaintext human-readable error message.

9. PAL & Certificate Enrollment

The /fullcmc PC is defined in [RFC7030]; the CMC (Certificate Management over Cryptographic Message Syntax) requirements and

packages 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: for PAL

                  <--------------------
                         GET res: PAL
                         Content-Type: application/xml

                 --------------------->
             GET req:
```

```
                  <--------------------
                         GET res: PAL
                         Content-Type: application/csr-attrs
```

For immediately rejected request, CMC works well. If the server prematurely closes the connection, then the procedures in Section 8.2.4 of [RFC7231] 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/renew request was processed:

- o Clients MUST authenticate the server and clients MUST check server's authorization.

- o 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. 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 is requested to perform 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:TBD" per the guidelines in [RFC3688]:

```
URI:  urn:ietf:params:xml:ns:TBD
Registrant Contact: Sean Turner (turners@ieca.com)
XML:
   BEGIN
      <?xml version="1.0"?>
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
      <html xmlns="http://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:TBD</h2>
         <p>See RFC TBD</p>
      </body>
      </html>
   END
```

11.2. PAL Schema

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

URI: urn:ietf:params:xml:schema:pal

Registrant Contact: Sean Turner turners@ieca.com

XML: See Section 2.1.2.

11.3. PAL Package Types

This section registers the PAL Package Types. Future PAL Package Types registrations are to be subject to Expert Review, as defined in RFC 5226 [RFC5226]. Package types MUST be paired with a media type.

The initial registry values are found in Section 2.1.1.

12. Acknowledgements

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

13. References

13.1. Normative References

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, November 2000.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4073] Housley, R., "Protecting Multiple Contents with the Cryptographic Message Syntax (CMS)", RFC 4073, May 2005.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, August 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, June 2008.

[RFC5273] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC): Transport Protocols", RFC 5273, June 2008.

[RFC5274] Schaad, J. and M. Myers, "Certificate Management Messages over CMS (CMC): Compliance Requirements", RFC 5274, June 2008.

[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, May 2008.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, January 2010.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, January 2010.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, August 2010.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, August 2010.

[RFC5959] Turner, S., "Algorithms for Asymmetric Key Package Content Type", RFC 5959, August 2010.

[RFC6010] Housley, R., Ashmore, S., and C. Wallace, "Cryptographic Message Syntax (CMS) Content Constraints Extension", RFC 6010, September 2010.

[RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, December 2010.

[RFC6032] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6032, December 2010.

[RFC6033] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6033, December 2010.

[RFC6160] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Protection of Symmetric Key Package Content Types", RFC 6160, April 2011.

[RFC6161] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6161, April 2011.

[RFC6162] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Asymmetric Key Package Content Type", RFC 6162, April 2011.

[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, July 2011.

[RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, November 2011.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, July 2014.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, October 2013.

[RFC7191] Housley, R., "Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7191, April 2014.

[RFC7192] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7192, April 2014.

[RFC7193] Turner, S., Housley, R., and J. Schaad, "The application/cms Media Type", RFC 7193, April 2014.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7292] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, July 2014.

[XML] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation, November 2008, <http://www.w3.org/TR/2006/REC-xml-20060816/>.

[XMLSCHEMA] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041082, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[X.690] ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002. Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

13.2. Informative References

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, November 2000.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

[XMLNS] Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", World Wide Web Consortium First Edition REC-xml-names-19990114, January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

Appendix A. Example Use of PAL

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

Steps for using a PAL include:

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=certs-only   |
                                                     |
         --------------------->                     -+
   GET req:                                          | /csrattrs
         <---------------------                      |
                           GET res: attributes       |
         --------------------->                     -+
   POST req: PKIRequest                              | /simpleenroll &
   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 encode ASN.1 objects. If the client knows 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 as part of 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)
    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 CRMF requests.

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

Or, the Content Info can be a signed data content type that further encapsulates a PKIData.

Appendix C. Example ASN.1

TO DO: Include BASE64 encodings of ASN.1 encodings of selected packages. They're a lot smaller than the ASN.1 pretty prints and there are tons of available to tools to convert.

Authors' Addresses

Sean Turner
IECA, Inc.
3057 Nutley Street, Suite 106
Fairfax, VA 22031
USA

EMail: turners@ieca.com