                 Trust Anchor Management Protocol (TAMP)

Abstract

   This document describes a transport independent protocol for the
   management of trust anchors (TAs) and community identifiers stored in
   a trust anchor store. The protocol makes use of the Cryptographic
   Message Syntax (CMS), and a digital signature is used to provide
   integrity protection and data origin authentication. The protocol
   can be used to manage trust anchor stores containing trust anchors
   represented as Certificate, TBSCertificate, or TrustAnchorInfo
   objects.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc5934.

Table of Contents

1. Introduction

   This document describes the Trust Anchor Management Protocol (TAMP).
   TAMP may be used to manage the trust anchors and community
   identifiers in any device that uses digital signatures; however, this
   specification was written with the requirements of cryptographic
   modules in mind. For example, TAMP can support signed firmware
   packages [RFC4108], where the trust anchor public key can be used to
   validate digital signatures on firmware packages or validate the
   X.509 certification path [RFC5280][X.509] of the firmware package
   signer.

   Most TAMP messages are digitally signed to provide integrity
   protection and data origin authentication. Both signed and unsigned
   TAMP messages employ the Cryptographic Message Syntax (CMS)
   [RFC5652]. The CMS is a data protection encapsulation syntax that
   makes use of ASN.1 [X.680].

   This specification does not provide for confidentiality of TAMP
   messages. If confidentiality is required, then the communications
   environment that is used to transfer TAMP messages must provide it.
   This specification is intended to satisfy the protocol-related
   requirements expressed in "Trust Anchor Management Requirements"
   [TA-MGMT-REQS] and uses vocabulary from that document.

   TAMP messages may be exchanged in real time over a network, such as
   via HTTP as described in Appendix A, or may be stored and transferred
   using other means. TAMP exchanges consist of a request message that
   includes instructions for a trust anchor store and, optionally, a
   corresponding response message that reports the result of carrying
   out the instructions in the request. Response messages need not be
   propagated in all cases. For example, a GPS receiver may be unable
   to transmit a response and may instead use an attached display to
   indicate the results of processing a TAMP request.

1.1. Terminology

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Trust Anchors

   TAMP manages trust anchors. A trust anchor contains a public key
   that is used to validate digital signatures. TAMP recognizes three
   formats for representing trust anchor information: Certificate
   [RFC5280], TBSCertificate [RFC5280], and TrustAnchorInfo [RFC5914].

   All trust anchors are distinguished by the public key, and all trust
   anchors consist of the following components:

   o A public key signature algorithm identifier and associated public
      key, which MAY include parameters

   o A public key identifier

   Other information may appear in a trust anchor, including
   certification path processing controls and a human readable name.

   TAMP recognizes three types of trust anchors based on functionality:
   apex trust anchors, management trust anchors, and identity trust
   anchors.

   In addition to the information described above, apex trust anchors
   and management trust anchors that sign TAMP messages have an
   associated sequence number that is used for replay detection.

The public key is used to name a trust anchor, and the public key
identifier is used to identify the trust anchor as a signer of a
particular object, such as a SignedData object or a public key
certificate. This public key identifier can be stored with the trust
anchor, or in most public key identifier assignment methods, it can
be computed from the public key whenever needed.

A trust anchor public key can be used in two different ways to
support digital signature validation. In the first approach, the
trust anchor public key is used directly to validate the digital
signature. In the second approach, the trust anchor public key is
used to validate an X.509 certification path, and then the subject
public key in the final certificate in the certification path is used
to validate the digital signature. When the second approach is
employed, the certified public key may be used for things other than
digital signature validation; the other possible actions are
constrained by the key usage certificate extension.

TAMP implementations MUST support validation of TAMP messages that
are directly validated using a trust anchor. Support for TAMP
messages validated using an X.509 certificate validated using a trust
anchor, or using longer certification paths, is OPTIONAL. The CMS
provides a location to carry X.509 certificates, and this facility
can be used to transfer certificates to aid in the construction of
the certification path.

1.2.1. Apex Trust Anchors

Within the context of a single trust anchor store, one trust anchor
is superior to all others. This trust anchor is referred to as the
apex trust anchor. This trust anchor represents the ultimate
authority over the trust anchor store. Much of this authority can be
delegated to other trust anchors.

The apex trust anchor private key is expected to be controlled by an
entity with information assurance responsibility for the trust anchor
store. The apex trust anchor is by definition unconstrained and
therefore does not have explicit authorization information associated
with it.

Due to the special nature of the apex trust anchor, TAMP includes
separate facilities to change it. In particular, TAMP includes a
facility to securely replace the apex trust anchor. This action
might be taken for one or more of the following reasons:

o The crypto period for the apex trust anchor public/private key
   pair has come to an end

o The apex trust anchor private key is no longer available

o The apex trust anchor public/private key pair needs to be revoked

o The authority has decided to use a different digital signature
   algorithm or the same digital signature algorithm with different
   parameters, such as a different elliptic curve

o The authority has decided to use a different key size

o The authority has decided to transfer control to another authority

To accommodate these requirements, the apex trust anchor MAY include
two public keys. Whenever the apex trust anchor is updated, both
public keys will be replaced. The first public key, called the
operational public key, is used in the same manner as other trust
anchors. Any type of TAMP message, including an Apex Trust Anchor
Update message, can be validated with the operational public key.
The second public key, called the contingency public key, can only be
used to update the apex trust anchor. The contingency private key
SHOULD be used at only one point in time; it is used only to sign an
Apex Trust Anchor Update message that results in its own replacement
(as well as the replacement of the operational public key). The
contingency public key is distributed in encrypted form. When the
contingency public key is used to validate an Apex Trust Anchor
Update message, the symmetric key needed to decrypt the contingency
public key is provided as part of the signed Apex Trust Anchor Update
message that is to be verified with the contingency public key.

1.2.2. Management Trust Anchors

Management trust anchors are used in the management of cryptographic
modules. For example, the TAMP messages specified in this document
are validated to a management trust anchor. Likewise, a signed
firmware package as specified in [RFC4108] is validated to a
management trust anchor.

1.2.3. Identity Trust Anchors

Identity trust anchors are used to validate certification paths, and
they represent the trust anchor for a public key infrastructure.
They are most often used in the validation of certificates associated
with non-management applications.

1.3. Architectural Elements

   TAMP does not assume any particular architecture. However, TAMP
   REQUIRES the following architectural elements: a cryptographic
   module, a trust anchor store, TAMP protocol processing, and other
   application-specific protocol processing.

   A globally unique algorithm identifier MUST be assigned for each one-
   way hash function, digital signature generation/validation algorithm,
   and symmetric key unwrapping algorithm that is implemented. To
   support CMS, an object identifier (OID) is assigned to name a one-way
   hash function, and another OID is assigned to name each combination
   of a one-way hash function when used with a digital signature
   algorithm. Similarly, certificates associate OIDs assigned to public
   key algorithms with subject public keys, and certificates make use of
   an OID that names both the one-way hash function and the digital
   signature algorithm for the certificate issuer digital signature.
   [RFC3279], [RFC3370], [RFC5753], and [RFC5754] provide OIDs for a
   number of commonly used algorithms; however, OIDs may be defined in
   later or different specifications.

1.3.1. Cryptographic Module

   The cryptographic module MUST include the following capabilities:

   o The cryptographic module SHOULD support the secure storage of a
      digital signature private key to sign TAMP responses and either a
      certificate containing the associated public key or a certificate
      designator. In the latter case, the certificate is stored
      elsewhere but is available to parties that need to validate
      cryptographic module digital signatures. The designator is a
      public key identifier.

   o The cryptographic module MUST support at least one one-way hash
      function, one digital signature validation algorithm, one digital
      signature generation algorithm, and, if contingency keys are
      supported, one symmetric key unwrapping algorithm. If only one
      one-way hash function is present, it MUST be consistent with the
      digital signature validation and digital signature generation
      algorithms. If only one digital signature validation algorithm is
      present, it MUST be consistent with the apex trust anchor
      operational public key. If only one digital signature generation
      algorithm is present, it MUST be consistent with the cryptographic
      module digital signature private key. These algorithms MUST be
      available for processing TAMP messages, including the content
      types defined in [RFC5652], and for validation of X.509

certification paths. As with similar specifications, such as
RFC 5280, this specification does not mandate support for any
cryptographic algorithms. However, algorithm requirements may be
imposed by specifications that use trust anchors managed via TAMP.

1.3.2. Trust Anchor Store

The trust anchor store MUST include the following capabilities:

o Each trust anchor store MUST have a unique name. For example, a
   cryptographic module containing a single trust anchor store may be
   identified by a unique serial number with respect to other modules
   within the same family where the family is represented as an ASN.1
   object identifier (OID) and the unique serial number is
   represented as a string of octets. Other means of establishing a
   unique name are also possible.

o Each trust anchor store SHOULD have the capability to securely
   store one or more community identifiers. The community identifier
   is an OID, and it identifies a collection of cryptographic modules
   that can be the target of a single TAMP message or the intended
   recipients for a particular management message.

o The trust anchor store SHOULD support the use of an apex trust
   anchor. If apex support is provided, the trust anchor store MUST
   support the secure storage of exactly one apex trust anchor. The
   trust anchor store SHOULD support the secure storage of at least
   one additional trust anchor. Each trust anchor MUST contain a
   unique public key. A public key MUST NOT appear more than once in
   a trust anchor store.

o The trust anchor store MUST have the capability to securely store
   a sequence number for each trust anchor authorized to generate
   TAMP messages and be able to report the sequence number along with
   the key identifier of the trust anchor.

1.3.3. TAMP Processing Dependencies

TAMP processing MUST include the following capabilities:

o TAMP processing MUST have a means of locating an appropriate trust
   anchor. Two mechanisms are available. The first mechanism is
   based on the public key identifier for digital signature
   verification, and the second mechanism is based on the trust
   anchor X.500 distinguished name and other X.509 certification path
   controls for certificate path discovery and validation. The first
   mechanism MUST be supported, but the second mechanism MAY be
   supported.

o   TAMP processing MUST be able to invoke the digital signature
    validation algorithm using the public key held in secure storage
    for trust anchors.

o   TAMP processing MUST have read and write access to secure storage
    for sequence numbers associated with each TAMP message signer as
    described in Section 6.

o   TAMP processing MUST have read and write access to secure storage
    for trust anchors in order to update them. Update operations
    include adding trust anchors, removing trust anchors, and
    modifying trust anchors. Application-specific constraints MUST be
    securely stored with each management trust anchor as described in
    Section 1.3.4.

o   TAMP processing MUST have read access to secure storage for the
    community membership list, if any, to determine whether a targeted
    message ought to be accepted.

o   To implement the OPTIONAL community identifier update feature,
    TAMP processing MUST have read and write access to secure storage
    for the community membership list.

o   To generate signed confirmation messages, TAMP processing MUST be
    able to invoke the digital signature generation algorithm using
    the cryptographic module digital signature private key, and it
    MUST have read access to the cryptographic module certificate or
    its designator. TAMP uses X.509 certificates [RFC5280].

o   The TAMP processing MUST have read access to the trust anchor
    store unique name.

1.3.4. Application-Specific Protocol Processing

   The apex trust anchor and management trust anchors managed with TAMP
   can be used by the TAMP application. Other management applications
   MAY make use of all three types of trust anchors, but non-management
   applications SHOULD only make use of identity trust anchors.
   Applications MUST ensure that usage of a trust anchor is consistent
   with any constraints associated with the trust anchor. For example,
   if name constraints are associated with a trust anchor, certification
   paths that start with the trust anchor and contain certificates with
   names that violate the name constraints MUST be rejected.

   The application-specific protocol processing MUST be provided with
   the following services:

o The application-specific protocol processing MUST have a means of
   locating an appropriate trust anchor. Two mechanisms are
   available to applications. The first mechanism is based on the
   public key identifier for digital signature verification, and the
   second mechanism is based on the trust anchor X.500 distinguished
   name and other X.509 certification path controls for certificate
   path discovery and validation.

o The application-specific protocol processing MUST be able to
   invoke the digital signature validation algorithm using the public
   key held in secure storage for trust anchors.

o The application-specific protocol processing MUST have read access
   to data associated with trust anchors to ensure that constraints
   can be enforced appropriately. For example, an application MUST
   have read access to any name constraints associated with a TA to
   ensure that certification paths terminated by that TA do not
   include certificates issued to entities outside the TA manager-
   designated namespace.

o The application-specific protocol processing MUST have read access
   to secure storage for the community membership list, if any, to
   determine whether a targeted message ought to be accepted.

o If the application-specific protocol requires digital signatures
   on confirmation messages or receipts, then the application-
   specific protocol processing MUST be able to invoke the digital
   signature generation algorithm with the cryptographic module
   digital signature private key and its associated certificate or
   certificate designator. Digital signature generation MUST be
   controlled in a manner that ensures that the content type of
   signed confirmation messages or receipts is appropriate for the
   application-specific protocol processing.

o The application-specific protocol processing MUST have read access
   to the trust anchor store unique name.

1.4. ASN.1 Encoding

   The CMS uses Abstract Syntax Notation One (ASN.1) [X.680]. ASN.1 is
   a formal notation used for describing data protocols, regardless of
   the programming language used by the implementation. Encoding rules
   describe how the values defined in ASN.1 will be represented for
   transmission. The Basic Encoding Rules (BER) [X.690] are the most
   widely employed rule set, but they offer more than one way to
   represent data structures. For example, definite-length encoding and
   indefinite-length encoding are supported. This flexibility is not
   desirable when digital signatures are used. As a result, the

   Distinguished Encoding Rules (DER) [X.690] were invented. DER is a
   subset of BER that ensures a single way to represent a given value.
   For example, DER always employs definite-length encoding.

   Digitally signed structures MUST be encoded with DER. In other
   specifications, structures that are not digitally signed do not
   require DER, but in this specification, DER is REQUIRED for all
   structures. By always using DER, the TAMP processor will have fewer
   options to implement.

   ASN.1 is used throughout the text of this document for illustrative
   purposes. The authoritative source of ASN.1 for the structures
   defined in this document is Appendix A.

2. Cryptographic Message Syntax Profile

   TAMP makes use of signed and unsigned messages. The Cryptographic
   Message Syntax (CMS) is used in both cases. A digital signature is
   used to protect the message from undetected modification and provide
   data origin authentication. TAMP makes no general provision for
   encryption of content.

   CMS is used to construct a signed TAMP message. The CMS ContentInfo
   content type MUST always be present. For signed messages,
   ContentInfo MUST encapsulate the CMS SignedData content type; for
   unsigned messages, ContentInfo MUST encapsulate the TAMP message
   directly. The CMS SignedData content type MUST encapsulate the TAMP
   message. A unique content type identifier identifies the particular
   type of TAMP message. The CMS encapsulation of a signed TAMP message
   is summarized by:

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

    SignedData {
      version   CMSVersion, -- Always set to 3
      digestAlgorithms DigestAlgorithmIdentifiers, -- Only one
      encapContentInfo EncapsulatedContentInfo,
      certificates   CertificateSet, -- OPTIONAL signer certificates
      crls   CertificateRevocationLists, -- OPTIONAL
      signerInfos   SET OF SignerInfo -- Only one
    }

```
   SignerInfo {
     version              CMSVersion, -- Always set to 3
     sid                  SignerIdentifier,
     digestAlgorithm      DigestAlgorithmIdentifer,
     signedAttrs          SignedAttributes,
                                    -- REQUIRED in TAMP messages
     signatureAlgorithm   SignatureAlgorithmIdentifier,
     signature            SignatureValue,
     unsignedAttrs        UnsignedAttributes -- OPTIONAL; may only be
   }                                    -- present in Apex Trust
                                        -- Anchor Update messages

  EncapsulatedContentInfo {
     eContentType   OBJECT IDENTIFIER, -- Names TAMP message type
     eContent       OCTET STRING       -- Contains TAMP message
  }
```

   When a TAMP message is used to update the apex trust anchor, this
   same structure is used; however, the digital signature will be
   validated with either the apex trust anchor operational public key or
   the contingency public key. When the contingency public key is used,
   the symmetric key needed to decrypt the previously stored contingency
   public key is provided as a contingency-public-key-decrypt-key
   unsigned attribute. Section 4.5 of this document describes the Apex
   Trust Anchor Update message.

   CMS is also used to construct an unsigned TAMP message. The CMS
   ContentInfo structure MUST always be present, and it MUST be the
   outermost layer of encapsulation. A unique content type identifier
   identifies the particular TAMP message. The CMS encapsulation of an
   unsigned TAMP message is summarized by:

```
  ContentInfo {
     contentType  OBJECT IDENTIFIER, -- Names TAMP message type
     content      OCTET STRING       -- Contains TAMP message
  }
```

2.1. ContentInfo

   CMS requires the outermost encapsulation to be ContentInfo [RFC5652].
   The fields of ContentInfo are used as follows:

   o contentType indicates the type of the associated content, and for
      TAMP, the encapsulated type is either SignedData or the content
      type identifier associated with an unsigned TAMP message. When
      the id-signedData (1.2.840.113549.1.7.2) object identifier is
      present in this field, then a signed TAMP message is in the
      content. Otherwise, an unsigned TAMP message is in the content.

   o content holds the content, and for TAMP, the content is either a
      SignedData content or an unsigned TAMP message.

2.2. SignedData Info

   The SignedData content type [RFC5652] contains the signed TAMP
   message and a digital signature value; the SignedData content type
   MAY also contain the certificates needed to validate the digital
   signature. The fields of SignedData are used as follows:

   o version is the syntax version number, and for TAMP, the version
      number MUST be set to 3.

   o digestAlgorithms is a collection of one-way hash function
      identifiers, and for TAMP, it contains a single one-way hash
      function identifier. The one-way hash function employed by the
      TAMP message originator in generating the digital signature MUST
      be present.

   o encapContentInfo is the signed content, consisting of a content
      type identifier and the content itself. The use of the
      EncapsulatedContentInfo type is discussed further in
      Section 2.2.2.

   o certificates is an OPTIONAL collection of certificates. It MAY be
      omitted, or it MAY include the X.509 certificates needed to
      construct the certification path of the TAMP message originator.
      For TAMP messages sent to a trust anchor store where an apex trust
      anchor or management trust anchor is used directly to validate the
      TAMP message digital signature, this field SHOULD be omitted.
      When an apex trust anchor or management trust anchor is used to
      validate an X.509 certification path [RFC5280], and the subject
      public key from the final certificate in the certification path is
      used to validate the TAMP message digital signature, the
      certificate of the TAMP message originator SHOULD be included, and
      additional certificates to support certification path construction
      MAY be included. For TAMP messages sent by a trust anchor store,
      this field SHOULD include only the signer's certificate or should
      be omitted. A TAMP message recipient MUST NOT reject a valid TAMP
      message that contains certificates that are not needed to validate
      the digital signature. PKCS#6 extended certificates [PKCS#6] and
      attribute certificates (either version 1 or version 2) [RFC5755]
      MUST NOT be included in the set of certificates; these certificate
      formats are not used in TAMP. Certification authority (CA)
      certificates and end entity certificates MUST conform to the
      profiles defined in [RFC5280].

   o crls is an OPTIONAL collection of certificate revocation lists
      (CRLs).

   o signerInfos is a collection of per-signer information, and for
      TAMP, the collection MUST contain exactly one SignerInfo. The use
      of the SignerInfo type is discussed further in Section 2.2.1.

2.2.1. SignerInfo

   The TAMP message originator is represented in the SignerInfo type.
   The fields of SignerInfo are used as follows:

   o version is the syntax version number. With TAMP, the version MUST
      be set to 3.

   o sid identifies the TAMP message originator's public key. The
      subjectKeyIdentifier alternative is always used with TAMP, which
      identifies the public key directly. When the public key is
      included in a TrustAnchorInfo object, this identifier is included
      in the keyId field. When the public key is included in a
      Certificate or TBSCertificate, this identifier is included in the
      subjectKeyIdentifier certificate extension.

   o digestAlgorithm identifies the one-way hash function, and any
      associated parameters, used by the TAMP message originator. It
      MUST contain the one-way hash functions employed by the
      originator. This message digest algorithm identifier MUST match
      the one carried in the digestAlgorithms field in SignedData. The
      message digest algorithm identifier is carried in two places to
      facilitate stream processing by the receiver.

   o signedAttrs is an OPTIONAL set of attributes that are signed along
      with the content. The signedAttrs are OPTIONAL in the CMS, but
      signedAttrs is REQUIRED for all signed TAMP messages. The SET OF
      Attribute MUST be encoded with the Distinguished Encoding Rules
      (DER) [X.690]. Section 2.2.3 of this document lists the signed
      attributes that MUST be included in the collection. Other signed
      attributes MAY be included, but any unrecognized signed attributes
      MUST be ignored.

   o signatureAlgorithm identifies the digital signature algorithm, and
      any associated parameters, used by the TAMP message originator to
      generate the digital signature.

   o signature is the digital signature value generated by the TAMP
      message originator.

   o unsignedAttrs is an OPTIONAL set of attributes that are not
      signed. For TAMP, this field is usually omitted. It is present
      only in Apex Trust Anchor Update messages that are to be validated
      using the apex trust anchor contingency public key. In this case,
      the SET OF Attribute MUST include the symmetric key needed to
      decrypt the contingency public key in the contingency-public-key-
      decrypt-key unsigned attribute. Section 2.2.4 of this document
      describes this unsigned attribute.

2.2.2. EncapsulatedContentInfo

   The EncapsulatedContentInfo structure contains the TAMP message. The
   fields of EncapsulatedContentInfo are used as follows:

   o eContentType is an object identifier that uniquely specifies the
      content type, and for TAMP, the value identifies the TAMP message.
      The list of TAMP message content types is provided in Section 4.

   o eContent is the TAMP message, encoded as an octet string. In
      general, the CMS does not require the eContent to be DER-encoded
      before constructing the octet string. However, TAMP messages MUST
      be DER-encoded.

2.2.3. Signed Attributes

   The TAMP message originator MUST digitally sign a collection of
   attributes along with the TAMP message. Each attribute in the
   collection MUST be DER-encoded. The syntax for attributes is defined
   in [RFC5912].

   Each of the attributes used with this CMS profile has a single
   attribute value. Even though the syntax is defined as a SET OF
   AttributeValue, there MUST be exactly one instance of AttributeValue
   present.

   The SignedAttributes syntax within SignerInfo is defined as a SET OF
   Attribute. The SignedAttributes MUST include only one instance of
   any particular attribute. TAMP messages that violate this rule MUST
   be rejected as malformed.

   The TAMP message originator MUST include the content-type and
   message-digest attributes. The TAMP message originator MAY also
   include the binary-signing-time attribute.

The TAMP message originator MAY include any other attribute that it
deems appropriate. The intent is to allow additional signed
attributes to be included if a future need is identified. This does
not cause an interoperability concern because unrecognized signed
attributes MUST be ignored.

The following summarizes the signed attribute requirements for TAMP
messages:

o content-type MUST be supported.

o message-digest MUST be supported.

o binary-signing-time MAY be supported. When present, it is
   generally ignored by the recipient.

o other attributes MAY be supported. Unrecognized attributes MUST
   be ignored by the recipient.

2.2.3.1. Content-Type Attribute

The TAMP message originator MUST include a content-type attribute; it
is an object identifier that uniquely specifies the content type.
Section 11.1 of [RFC5652] defines the content-type attribute. For
TAMP, the value identifies the TAMP message. The list of TAMP
message content types and their identifiers is provided in Section 4.

A content-type attribute MUST contain the same object identifier as
the content type contained in the EncapsulatedContentInfo.

2.2.3.2. Message-Digest Attribute

The TAMP message originator MUST include a message-digest attribute,
having as its value the output of a one-way hash function computed on
the TAMP message that is being signed. Section 11.2 of [RFC5652]
defines the message-digest attribute.

2.2.3.3. Binary-Signing-Time Attribute

The TAMP message originator MAY include a binary-signing-time
attribute, specifying the time at which the digital signature was
applied to the TAMP message. The binary-signing-time attribute is
defined in [RFC4049].

No processing of the binary-signing-time attribute is REQUIRED of a
TAMP message recipient; however, the binary-signing-time attribute
MAY be included by the TAMP message originator as a form of message
identifier.

2.2.4. Unsigned Attributes

   For TAMP, unsigned attributes are usually omitted. An unsigned
   attribute is present only in Apex Trust Anchor Update messages that
   are to be validated by the apex trust anchor contingency public key.
   In this case, the symmetric key to decrypt the previous contingency
   public key is provided in the contingency-public-key-decrypt-key
   unsigned attribute. This attribute MUST be supported, and it is
   described in Section 2.2.4.1.

   The TAMP message originator SHOULD NOT include other unsigned
   attributes, and any unrecognized unsigned attributes MUST be ignored.

   The UnsignedAttributes syntax within SignerInfo is defined as a SET
   OF Attribute. The UnsignedAttributes MUST include only one instance
   of any particular attribute. TAMP messages that violate this rule
   MUST be rejected as malformed.

2.2.4.1. Contingency-Public-Key-Decrypt-Key Attribute

   The contingency-public-key-decrypt-key attribute provides the
   plaintext symmetric key needed to decrypt the previously distributed
   apex trust anchor contingency public key. The symmetric key MUST be
   useable with the symmetric algorithm used to previously encrypt the
   contingency public key.

   The contingency-public-key-decrypt-key attribute has the following
   syntax:

     contingency-public-key-decrypt-key ATTRIBUTE ::= {
        WITH SYNTAX PlaintextSymmetricKey
        SINGLE VALUE TRUE
        ID id-aa-TAMP-contingencyPublicKeyDecryptKey }

     id-aa-TAMP-contingencyPublicKeyDecryptKey
        OBJECT IDENTIFIER ::= { id-attributes 63 }

     PlaintextSymmetricKey ::= OCTET STRING

3. Trust Anchor Formats

   TAMP recognizes three formats for representing trust anchor
   information within the protocol itself: Certificate [RFC5280],
   TBSCertificate [RFC5280], and TrustAnchorInfo [RFC5914]. The
   TrustAnchorChoice structure, defined in [RFC5914], is used to select
   one of these options.

```
   TrustAnchorChoice ::= CHOICE {
     certificate  Certificate,
     tbsCert      [1] EXPLICIT TBSCertificate,
     taInfo       [2] EXPLICIT TrustAnchorInfo }
```

   The Certificate structure is commonly used to represent trust
   anchors. Certificates include a signature, which removes the ability
   for relying parties to customize the information within the structure
   itself. TBSCertificate contains all of the information of the
   Certificate structure except for the signature, enabling tailoring of
   the information. TrustAnchorInfo is intended to serve as a
   minimalist representation of trust anchor information for scenarios
   where storage or bandwidth is highly constrained.

   Implementations are not required to support all three options. The
   unsupportedTrustAnchorFormat error code should be indicated when
   generating a TAMPError due to receipt of an unsupported trust anchor
   format.

4. Trust Anchor Management Protocol Messages

   TAMP makes use of signed and unsigned messages. The CMS is used in
   both cases. An object identifier is assigned to each TAMP message
   type, and this object identifier is used as a content type in the
   CMS.

   TAMP specifies eleven message types. The following provides the
   content type identifier for each TAMP message type, and it indicates
   whether a digital signature is required. If the following indicates
   that the TAMP message MUST be signed, then implementations MUST
   reject a message of that type that is not signed.

   o The TAMP Status Query message MUST be signed. It uses the
      following object identifier: { id-tamp 1 }.

   o The TAMP Status Response message SHOULD be signed. It uses the
      following object identifier: { id-tamp 2 }.

   o The Trust Anchor Update message MUST be signed. It uses the
      following object identifier: { id-tamp 3 }.

   o The Trust Anchor Update Confirm message SHOULD be signed. It uses
      the following object identifier: { id-tamp 4 }.

   o The Apex Trust Anchor Update message MUST be signed. It uses the
      following object identifier: { id-tamp 5 }.

o The Apex Trust Anchor Update Confirm message SHOULD be signed. It
   uses the following object identifier: { id-tamp 6 }.

o The Community Update message MUST be signed. It uses the
   following object identifier: { id-tamp 7 }.

o The Community Update Confirm message SHOULD be signed. It uses
   the following object identifier: { id-tamp 8 }.

o The Sequence Number Adjust MUST be signed. It uses the following
   object identifier: { id-tamp 10 }.

o The Sequence Number Adjust Confirm message SHOULD be signed. It
   uses the following object identifier: { id-tamp 11 }.

o The TAMP Error message SHOULD be signed. It uses the following
   object identifier: { id-tamp 9 }.

Trust anchor managers generate TAMP Status Query, Trust Anchor
Update, Apex Trust Anchor Update, Community Update, and Sequence
Number Adjust messages. Trust anchor stores generate TAMP Status
Response, Trust Anchor Update Confirm, Apex Trust Anchor Update
Confirm, Community Update Confirm, Sequence Number Adjust Confirm,
and TAMP Error messages.

Support for Trust Anchor Update messages is REQUIRED. Support for
all other message formats is RECOMMENDED. Implementations that
support the HTTP binding described in Appendix C MUST additionally
support Trust Anchor Update Confirm and TAMP Error messages and MAY
support 0 or more of the following pairs of messages: TAMP Status
Query and TAMP Status Query Response; Apex Trust Anchor Update and
Apex Trust Anchor Update Confirm; Community Update and Community
Update Confirm; Sequence Number Adjust and Sequence Number Adjust
Confirm. Implementations that operate in a disconnected manner MUST
NOT assume a response will be received from each consumer of a TAMP
message.

A typical interaction between a trust anchor manager and a trust
anchor store will follow the message flow shown in Figure 1. Figure
1 does not illustrate a flow where an error occurs.

```
      +---------+       Trust Anchor Status Query    +---------+
      |         |     |------------------------------>|         |
      |         |     |                               |         |
      |         |     |   Trust Anchor Status Response|         |
      | Trust   |     |<------------------------------| Trust   |
      | Anchor  |     |                               | Anchor  |
      | Manager |     |   Trust Anchor Update         | Store   |
      |         |     |------------------------------>|         |
      |         |     |                               |         |
      |         |     |   Trust Anchor Update Confirm |         |
      |         |     |<------------------------------|         |
      |         |     |                               |         |
      +---------+       |                             +---------+
```
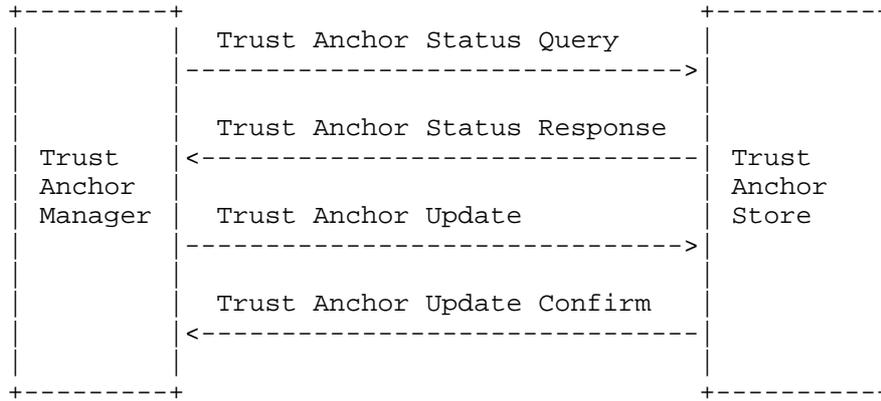
                 Figure 1. Typical TAMP Message Flow

   Each TAMP query and update message includes an indication of the type
   of response that is desired. The response can either be terse or
   verbose. All trust anchor stores MUST support both the terse and
   verbose responses and SHOULD generate a response of the type
   indicated in the corresponding request. TAMP response processors
   MUST support processing of both terse and verbose responses.

   Trust anchor stores SHOULD be able to process and properly act upon
   the valid payload of the TAMP Status Query message, the Trust Anchor
   Update message, the Apex Trust Anchor Update message, and the
   Sequence Number Adjust message. TAMP implementations MAY also
   process and act upon the valid payload of the Community Update
   message.

   TAMP implementations SHOULD support generation of the TAMP Status
   Response message, the Trust Anchor Update Confirm message, the Apex
   Trust Anchor Update Confirm message, the Sequence Number Adjust
   Confirm message, and the TAMP Error message. If a TAMP
   implementation supports the Community Update message, then generation
   of Community Update Confirm messages SHOULD also be supported.

4.1. TAMP Status Query

   The TAMP Status Query message is used to request information about
   the trust anchors that are currently installed in a trust anchor
   store, and for the list of communities to which the store belongs.
   The TAMP Status Query message MUST be signed. For the query message
   to be valid, the trust anchor store MUST be an intended recipient of
   the query; the sequence number checking described in Section 6 MUST
   be successful when the TAMP message signer is a trust anchor; and the
   digital signature MUST be validated by the apex trust anchor

   operational public key, an authorized management trust anchor, or via
   an authorized X.509 certification path originating with such a trust
   anchor.

   If the digital signature on the TAMP Status Query message is valid,
   sequence number checking is successful, the signer is authorized, and
   the trust anchor store is an intended recipient of the TAMP message,
   then a TAMP Status Response message SHOULD be returned. If a TAMP
   Status Response message is not returned, then a TAMP Error message
   SHOULD be returned.

   The TAMP Status Query content type has the following syntax:

```
CONTENT-TYPE  ::= TYPE-IDENTIFIER

tamp-status-query CONTENT-TYPE  ::=
   { TAMPStatusQuery IDENTIFIED BY id-ct-TAMP-statusQuery }

id-ct-TAMP-statusQuery OBJECT IDENTIFIER ::= { id-tamp 1 }

TAMPStatusQuery ::= SEQUENCE {
  Version  [0] TAMPVersion DEFAULT v2,
  terse    [1] TerseOrVerbose DEFAULT verbose,
  query    TAMPMsgRef }

TAMPVersion ::= INTEGER { v1(1), v2(2) }

TerseOrVerbose ::= ENUMERATED { terse(1), verbose(2) }

TAMPMsgRef ::= SEQUENCE {
  target  TargetIdentifier,
  seqNum  SeqNumber }

SeqNumber ::= INTEGER (0..9223372036854775807)

TargetIdentifier ::= CHOICE {
  hwModules     [1] HardwareModuleIdentifierList,
  communities   [2] CommunityIdentifierList,
  allModules    [3] NULL,
  uri           [4] IA5String,
  otherName     [5] AnotherName }

HardwareModuleIdentifierList ::= SEQUENCE SIZE (1..MAX) OF
                                   HardwareModules

HardwareModules ::= SEQUENCE {
  hwType            OBJECT IDENTIFIER,
  hwSerialEntries   SEQUENCE SIZE (1..MAX) OF HardwareSerialEntry }
```

```
   HardwareSerialEntry ::= CHOICE {
      all     NULL,
      single  OCTET STRING,
      block   SEQUENCE {
         low     OCTET STRING,
         high    OCTET STRING } }

   CommunityIdentifierList ::= SEQUENCE SIZE (0..MAX) OF Community

   Community ::= OBJECT IDENTIFIER
```

The fields of TAMPStatusQuery are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o terse indicates the type of response that is desired. A terse
   response is indicated by a value of 1, and a verbose response is
   indicated by a value of 2, which is omitted during encoding since
   it is the default value.

o query contains two items: the target and the seqNum. target
   identifies the target(s) of the query message. seqNum is a
   single-use value that will be used to match the TAMP Status Query
   message with the TAMP Status Response message. The sequence
   number is also used to detect TAMP message replay. The sequence
   number processing described in Section 6 MUST successfully
   complete before a response is returned.

The fields of TAMPMsgRef are used as follows:

o target identifies the target(s) of the query. Several
   alternatives for naming a target are provided. To identify a
   cryptographic module, a combination of a cryptographic type and
   serial number are used. The cryptographic type is represented as
   an ASN.1 object identifier, and the unique serial number is
   represented as a string of octets. To facilitate compact
   representation of serial numbers, a contiguous block can be
   specified by the lowest included serial number and the highest
   included serial number. When present, the high and low octet
   strings MUST have the same length. The
   HardwareModuleIdentifierList sequence MUST NOT contain duplicate
   hwType values, so that each member of the sequence names all of
   the cryptographic modules of this type. Object identifiers are
   also used to identify communities of trust anchor stores. A
   sequence of these object identifiers is used if more than one
   community is the target of the message. A trust anchor store is
   considered a target if it is a member of any of the listed

communities. An explicit NULL value is used to identify all
modules that consider the signer of the TAMP message to be an
authorized source for that message type. The uri field can be
used to identify a target, i.e., a trust anchor store, using a
Uniform Resource Identifier [RFC3986]. Additional name types are
supported via the otherName field, which is of type AnotherName.
AnotherName is defined in [RFC5280]. The format and semantics of
the name are indicated through the OBJECT IDENTIFIER in the type-
id field. The name itself is conveyed as a value field in
otherName. Implementations MUST support the allModules option and
SHOULD support all TargetIdentifier options.

o seqNum contains a single-use value that will be used to match the
   TAMP Status Query message with the successful TAMP Status Response
   message. The sequence number processing described in Section 6
   MUST successfully complete before a response is returned.

To determine whether a particular cryptographic module serial number
is considered part of a specified block, all of the following
conditions MUST be met. First, the cryptographic module serial
number MUST be the same length as both the high and low octet
strings. Second, the cryptographic module serial number MUST be
greater than or equal to the low octet string. Third, the
cryptographic module serial number MUST be less than or equal to the
high octet string.

One octet string is equal to another if they are of the same length
and are the same at each octet position. An octet string, S1, is
greater than another, S2, where S1 and S2 have the same length, if
and only if S1 and S2 have different octets in one or more positions,
and in the first such position, the octet in S1 is greater than that
in S2, considering the octets as unsigned binary numbers. Note that
these octet string comparison definitions are consistent with those
in clause 6 of [X.690].

4.2. TAMP Status Query Response

The TAMP Status Response message is a reply by a trust anchor store
to a valid TAMP Status Query message. The TAMP Status Response
message provides information about the trust anchors that are
currently installed in the trust anchor store and the list of
communities to which the trust anchor store belongs, if any. The
TAMP Status Response message MAY be signed or unsigned. A TAMP
Status Response message MUST be signed if the implementation is
capable of signing it.

   The TAMP Status Response content type has the following syntax:

```
   tamp-status-response CONTENT-TYPE  ::=
      { TAMPStatusResponse IDENTIFIED BY id-ct-TAMP-statusResponse }

   id-ct-TAMP-statusResponse OBJECT IDENTIFIER ::= { id-tamp 2 }

   TAMPStatusResponse ::= SEQUENCE {
      version    [0] TAMPVersion DEFAULT v2,
      query      TAMPMsgRef,
      response   StatusResponse,
      usesApex   BOOLEAN DEFAULT TRUE }

   StatusResponse ::= CHOICE {
      terseResponse           [0] TerseStatusResponse,
      verboseResponse         [1] VerboseStatusResponse }

   TerseStatusResponse ::= SEQUENCE {
      taKeyIds              KeyIdentifiers,
      communities           CommunityIdentifierList OPTIONAL }

   KeyIdentifiers ::= SEQUENCE SIZE (1..MAX) OF KeyIdentifier

   VerboseStatusResponse ::= SEQUENCE {
      taInfo                TrustAnchorChoiceList,
      continPubKeyDecryptAlg [0] AlgorithmIdentifier OPTIONAL,
      communities           [1] CommunityIdentifierList OPTIONAL,
      tampSeqNumbers        [2] TAMPSequenceNumbers OPTIONAL }

   TrustAnchorChoiceList ::= SEQUENCE SIZE (1..MAX) OF
       TrustAnchorChoice

   TAMPSequenceNumbers ::= SEQUENCE SIZE (1..MAX) OF TAMPSequenceNumber

   TAMPSequenceNumber ::= SEQUENCE {
      keyId       KeyIdentifier,
      seqNumber   SeqNumber }
```

   The fields of TAMPStatusResponse are used as follows:

   o version identifies version of TAMP. For this version of the
      specification, the default value, v2, MUST be used.

   o query identifies the TAMPStatusQuery to which the trust anchor
      store is responding. The query structure repeats the TAMPMsgRef
      from the TAMP Status Query message (see Section 4.1). The
      sequence number processing described in Section 6 MUST
      successfully complete before any response is returned.

o response contains either a terse response or a verbose response.
   The terse response is represented by TerseStatusResponse, and the
   verbose response is represented by VerboseStatusResponse.

o usesApex is a Boolean value that indicates whether the first item
   in the TerseStatusResponse.taKeyIds or
   VerboseStatusResponse.taInfo field identifies the apex TA.

The fields of TerseStatusResponse are used as follows:

o taKeyIds contains a sequence of key identifiers. Each trust
   anchor contained in the trust anchor store is represented by one
   key identifier. When TAMPStatusResponse.usesApex is TRUE, the
   apex trust anchor is represented by the first key identifier in
   the sequence, which contains the key identifier of the operational
   public key.

o communities is OPTIONAL. When present, it contains a sequence of
   object identifiers. Each object identifier names one community to
   which this trust anchor store belongs. When the trust anchor
   store belongs to no communities, this field is omitted.

The fields of VerboseStatusResponse are used as follows:

o taInfo contains a sequence of TrustAnchorChoice structures. One
   entry in the sequence is provided for each trust anchor contained
   in the trust anchor store. When TAMPStatusResponse.usesApex is
   TRUE, the apex trust anchor is the first trust anchor in the
   sequence.

o continPubKeyDecryptAlg is OPTIONAL. When present, it indicates
   the decryption algorithm needed to decrypt the currently installed
   apex trust anchor contingency public key, if a contingency key is
   associated with the apex trust anchor. When present,
   TAMPStatusResponse.usesApex MUST be TRUE.

o communities is OPTIONAL. When present, it contains a sequence of
   object identifiers. Each object identifier names one community to
   which this trust anchor store belongs. When the trust anchor
   store belongs to no communities, this field is omitted.

o tampSeqNumbers is OPTIONAL. When present, it is used to indicate
   the currently held sequence number for each trust anchor
   authorized to sign TAMP messages. The keyId field identifies the
   trust anchor, and the seqNumber field provides the current
   sequence number associated with the trust anchor.

4.3. Trust Anchor Update

   The Trust Anchor Update message is used to add, remove, and change
   management and identity trust anchors. The Trust Anchor Update
   message cannot be used to update the apex trust anchor. The Trust
   Anchor Update message MUST be signed. For a Trust Anchor Update
   message to be valid, the trust anchor store MUST be an intended
   recipient of the update; the sequence number checking described in
   Section 6 MUST be successful when the TAMP message signer is a trust
   anchor; and the digital signature MUST be validated using the apex
   trust anchor operational public key, an authorized management trust
   anchor, or via an authorized X.509 certification path originating
   with such a trust anchor.

   If the digital signature on the Trust Anchor Update message is valid,
   sequence number checking is successful, the signer is authorized, and
   the trust anchor store is an intended recipient of the TAMP message,
   then the trust anchor store MUST perform the specified updates and
   return a Trust Anchor Update Confirm message. If a Trust Anchor
   Update Confirm message is not returned, then a TAMP Error message
   SHOULD be returned.

   The Trust Anchor Update content type has the following syntax:

    tamp-update CONTENT-TYPE ::=
       { TAMPUpdate IDENTIFIED BY id-ct-TAMP-update }

    id-ct-TAMP-update OBJECT IDENTIFIER ::= { id-tamp 3 }

    TAMPUpdate ::= SEQUENCE {
      version   [0] TAMPVersion DEFAULT v2,
      terse   [1] TerseOrVerbose DEFAULT verbose,
      msgRef   TAMPMsgRef,
      updates   SEQUENCE SIZE (1..MAX) OF TrustAnchorUpdate,
      tampSeqNumbers [2]TAMPSequenceNumbers OPTIONAL }

    TrustAnchorUpdate ::= CHOICE {
      add   [1] TrustAnchorChoice,
      remove [2] SubjectPublicKeyInfo,
      change [3] EXPLICIT TrustAnchorChangeInfoChoice }

    TrustAnchorChangeInfoChoice ::= CHOICE {
      tbsCertChange [0] TBSCertificateChangeInfo,
      taChange   [1] TrustAnchorChangeInfo }

```
TBSCertificateChangeInfo  ::=  SEQUENCE  {
   serialNumber         CertificateSerialNumber OPTIONAL,
   signature            [0] AlgorithmIdentifier OPTIONAL,
   issuer               [1] Name OPTIONAL,
   validity             [2] Validity OPTIONAL,
   subject              [3] Name OPTIONAL,
   subjectPublicKeyInfo [4] SubjectPublicKeyInfo,
   exts                 [5] EXPLICIT Extensions OPTIONAL }

TrustAnchorChangeInfo ::= SEQUENCE {
   pubKey          SubjectPublicKeyInfo,
   keyId           KeyIdentifier OPTIONAL,
   taTitle         TrustAnchorTitle OPTIONAL,
   certPath        CertPathControls OPTIONAL,
   exts            [1] Extensions OPTIONAL }
```

The fields of TAMPUpdate are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o terse indicates the type of response that is desired. A terse
   response is indicated by a value of 1, and a verbose response is
   indicated by a value of 2, which is omitted during encoding since
   it is the default value.

o msgRef contains two items: the target and the seqNum. target
   identifies the target(s) of the update message. The
   TargetIdentifier syntax is described in Section 4.1. seqNum is a
   single-use value that will be used to match the Trust Anchor
   Update message with the Trust Anchor Update Confirm message. The
   sequence number is also used to detect TAMP message replay. The
   sequence number processing described in Section 6 MUST
   successfully complete before any of the updates are processed.

o updates contains a sequence of updates, which are used to add,
   remove, and change management or identity trust anchors. Each
   entry in the sequence represents one of these actions, and is
   indicated by an instance of TrustAnchorUpdate. The actions are a
   batch of updates that MUST be processed in the order that they
   appear, but each of the updates is processed independently. Each
   of the updates MUST satisfy the subordination checks described in
   Section 7. Even if one or more of the updates fail, then the
   remaining updates MUST be processed. These updates MUST NOT make
   any changes to the apex trust anchor.

o tampSeqNumbers MAY be included to provide the initial or new
   sequence numbers for trust anchors added or changed by the updates
   field. Elements included in the tampSeqNumbers field that do not
   correspond to an element in the updates field are ignored.
   Elements included in the tampSeqNumbers field that do correspond
   to an element in the updates field and contain a sequence number
   less than or equal to the most recently stored sequence number for
   the trust anchor are ignored. Elements included in the
   tampSeqNumbers field that do correspond to an element in the
   updates field and contain a sequence number greater than the most
   recently stored sequence number for the indicated trust anchor are
   processed by setting the stored sequence number for the trust
   anchor equal to the new value.

The TrustAnchorUpdate is a choice of three structures, and each
alternative represents one of the three possible actions: add,
remove, and change. A description of the syntax associated with each
of these actions follows:

o add is used to insert a new management or identity trust anchor
   into the trust anchor store. The TrustAnchorChoice structure is
   used to provide the trusted public key and all of the information
   associated with it. However, the action MUST fail with the error
   code notAuthorized if the subordination checks described in
   Section 7 are not satisfied. See Section 3 for a discussion of
   the TrustAnchorChoice structure. The apex trust anchor cannot be
   introduced into a trust anchor store using this action; therefore,
   the id-pe-wrappedApexContinKey MUST NOT be present in the
   extensions field. The constraints of the existing trust anchors
   are unchanged by this action. An attempt to add a management or
   identity trust anchor that is already in place with the same
   values for every field in the TrustAnchorChoice structure MUST be
   treated as a successful addition. An attempt to add a management
   or identity trust anchor that is already present with the same
   pubKey values, but with different values for any of the fields in
   the TrustAnchorChoice structure, MUST fail with the error code
   improperTAAddition. This means a trust anchor may not be added
   twice using different TrustAnchorChoice options. If a different
   format is desired, the existing trust anchor must be removed and
   the new format added.

o remove is used to delete an existing management or identity trust
   anchor from the trust anchor store, including the deletion of the
   management trust anchor associated with the TAMP message signer.
   However, the action MUST fail with the error code notAuthorized if
   the subordination checks described in Section 7 are not satisfied.
   The public key contained in SubjectPublicKeyInfo names the
   management or identity trust anchor to be deleted. An attempt to

delete a trust anchor that is not present MUST be treated as a
successful deletion. The constraints of the deleted trust anchor
are not distributed to other trust anchors in any manner. The
apex trust anchor cannot be removed using this action, which
ensures that this action cannot place the trust anchor store in an
unrecoverable configuration.

o change is used to update the information associated with an
   existing management or identity trust anchor in the trust anchor
   store. Attempts to change a trust anchor added as a Certificate
   MUST fail with the error code improperTAChange. The public key
   contained in the SubjectPublicKeyInfo field of
   TrustAnchorChangeInfo or in the subjectPublicKeyInfo field of a
   TBSCertificateChangeInfo names the to-be-updated trust anchor.
   However, the action MUST fail with the error code notAuthorized if
   the subordination checks described in Section 7 are not satisfied.
   An attempt to change a trust anchor that is not present MUST
   result in a failure with the trustAnchorNotFound status code. The
   TrustAnchorChangeInfo structure or the TBSCertificateChangeInfo
   structure is used to provide the revised configuration of the
   management or identity trust anchor. If the update fails for any
   reason, then the original trust anchor configuration MUST be
   preserved. The apex trust anchor information cannot be changed
   using this action. Attempts to change a trust anchor added as a
   TBSCertificate using a TrustAnchorChangeInfo MUST fail with an
   improperTAChange error. Attempts to change a trust anchor added
   as a TrustAnchorInfo using a TBSCertificateChangeInfo MUST fail
   with an improperTAChange error.

The fields of TrustAnchorChangeInfo are used as follows:

o pubKey contains the algorithm identifier and the public key of the
   management or identity trust anchor. It is used to locate the
   to-be-updated trust anchor in the trust anchor store.

o keyId is OPTIONAL, and when present, it contains the public key
   identifier of the trust anchor public key, which is contained in
   the pubKey field. If this field is not present, then the public
   key identifier remains unchanged. If this field is present, the
   provided public key identifier replaces the previous one.

o taTitle is OPTIONAL, and when present, it provides a human
   readable name for the management or identity trust anchor. When
   absent in a change trust anchor update, any title that was
   previously associated with the trust anchor is removed.
   Similarly, when present in a change trust anchor update, the title

      in the message is associated with the trust anchor. If a previous
      title was associated with the trust anchor, then the title is
      replaced. If a title was not previously associated with the trust
      anchor, then the title from the update message is added.

   o certPath is OPTIONAL, and when present, it provides the controls
      needed to construct and validate an X.509 certification path.
      When absent in a change trust anchor update, any controls that
      were previously associated with the management or identity trust
      anchor are removed, which means that delegation is no longer
      permitted. Similarly, when present in a change trust anchor
      update, the controls in the message are associated with the
      management or identity trust anchor. If previous controls,
      including the trust anchor distinguished name, were associated
      with the trust anchor, then the controls are replaced, which means
      that delegation continues to be supported, but that different
      certification paths will be valid. If controls were not
      previously associated with the management or identity trust
      anchor, then the controls from the update message are added, which
      enables delegation. The syntax and semantics of CertPathControls
      are discussed in [RFC5914].

   o exts is OPTIONAL, and when present, it provides the extensions
      values that are associated with the trust anchor. When absent in
      a change trust anchor update, any extensions that were previously
      associated with the trust anchor are removed. Similarly, when
      present in a change trust anchor update, the extensions in the
      message are associated with the trust anchor. Any extensions
      previously associated with the trust anchor are replaced or
      removed.

   The fields of TBSCertificateChangeInfo are used to alter the fields
   within a TBSCertificate structure. TBSCertificate is described in
   [RFC5280]. For all fields except exts, if the field is absent in a
   change trust anchor update, then any previous value associated with a
   trust anchor is unchanged. For the exts field, if the field is
   absent in a change trust anchor update, then any previous value
   associated with a trust anchor is removed. For all fields, if the
   field is present in a change trust anchor update, then any previous
   value associated with a trust anchor is replaced with the value from
   the update message.

4.3.1. Trust Anchor List

   [RFC5914] defines the TrustAnchorList structure to convey a list of
   trust anchors. TAMP implementations MAY process TrustAnchorList
   objects (with eContentType (or contentType) using the id-ct-
   trustAnchorList OID defined in [RFC5914]) as equivalent to TAMPUpdate

objects with terse set to terse, msgRef set to allModules (with a
suitable sequence number), and all elements within the list contained
within the add field. This alternative to TrustAnchorUpdate is
provided for implementations that perform integrity and authorization
checks out-of-band as a simple means of transferring trust anchors
from one trust anchor store to another. It does not provide a means
of removing or changing trust anchors and has no HTTP binding.

4.4. Trust Anchor Update Confirm

   The Trust Anchor Update Confirm message is a reply by a trust anchor
   store to a valid Trust Anchor Update message. The Trust Anchor
   Update Confirm message provides success and failure information for
   each of the requested updates. The Trust Anchor Update Confirm
   message MAY be signed or unsigned. A Trust Anchor Update Confirm
   message MUST be signed if the implementation is capable of
   signing it.

   The Trust Anchor Update Confirm content type has the following
   syntax:

```
 tamp-update-confirm CONTENT-TYPE  ::=
    { TAMPUpdateConfirm IDENTIFIED BY id-ct-TAMP-updateConfirm }

 id-ct-TAMP-updateConfirm OBJECT IDENTIFIER ::= { id-tamp 4 }

 TAMPUpdateConfirm ::= SEQUENCE {
   version  [0] TAMPVersion DEFAULT v2,
   update   TAMPMsgRef,
   confirm  UpdateConfirm }

 UpdateConfirm ::= CHOICE {
   terseConfirm    [0] TerseUpdateConfirm,
   verboseConfirm  [1] VerboseUpdateConfirm }

 TerseUpdateConfirm ::= StatusCodeList

 StatusCodeList ::= SEQUENCE SIZE (1..MAX) OF StatusCode

 VerboseUpdateConfirm ::= SEQUENCE {
   status          StatusCodeList,
   taInfo          TrustAnchorChoiceList,
   tampSeqNumbers  TAMPSequenceNumbers OPTIONAL,
   usesApex        BOOLEAN DEFAULT TRUE }
```

The fields of TAMPUpdateConfirm are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o update identifies the TAMPUpdate message to which the trust anchor
   store is responding. The update structure repeats the TAMPMsgRef
   from the Trust Anchor Update message (see Section 4.3). The
   sequence number processing described in Section 6 MUST
   successfully complete before any of the updates are processed.

o confirm contains either a terse update confirmation or a verbose
   update confirmation. The terse update confirmation is represented
   by TerseUpdateConfirm, and the verbose response is represented by
   VerboseUpdateConfirm.

The TerseUpdateConfirm contains a sequence of status codes, one for
each TrustAnchorUpdate structure in the Trust Anchor Update message.
The status codes MUST appear in the same order as the
TrustAnchorUpdate structures to which they apply, and the number of
elements in the status code list MUST be the same as the number of
elements in the trust anchor update list. Each of the status codes
is discussed in Section 5.

The fields of VerboseUpdateConfirm are used as follows:

o status contains a sequence of status codes, one for each
   TrustAnchorUpdate structure in the Trust Anchor Update message.
   The status codes appear in the same order as the TrustAnchorUpdate
   structures to which they apply, and the number of elements in the
   status code list MUST be the same as the number of elements in the
   trust anchor update list. Each of the status codes is discussed
   in Section 5.

o taInfo contains a sequence of TrustAnchorChoice structures. One
   entry in the sequence is provided for each trust anchor contained
   in the trust anchor store. These represent the state of the trust
   anchors after the updates have been processed. When usesApex is
   true, the apex trust anchor is the first trust anchor in the
   sequence.

o tampSeqNumbers is used to indicate the currently held sequence
   number for each trust anchor authorized to sign TAMP messages.
   The keyId field identifies the trust anchor, and the seqNumber
   field provides the current sequence number associated with the
   trust anchor.

   o usesApex is a Boolean value that indicates whether the first item
      in the taInfo field identifies the apex TA.

4.5. Apex Trust Anchor Update

   The Apex Trust Anchor Update message replaces the operational public
   key and, optionally, the contingency public key associated with the
   apex trust anchor. Each trust anchor store has exactly one apex
   trust anchor. No constraints are associated with the apex trust
   anchor. The public key identifier of the operational public key is
   used to identify the apex trust anchor in subsequent TAMP messages.
   The digital signature on the Apex Trust Anchor Update message is
   validated with either the current operational public key or the
   current contingency public key. For the Apex Trust Anchor Update
   message that is validated with the operational public key to be
   valid, the trust anchor store MUST be a target of the update, the
   sequence number MUST be larger than the most recently stored sequence
   number for the operational public key, and the digital signature MUST
   be validated directly with the operational public key. That is, no
   delegation via a certification path is permitted. For the Apex Trust
   Anchor Update message that is validated with the contingency public
   key to be valid, the trust anchor store MUST be a target of the
   update, the provided decryption key MUST properly decrypt the
   contingency public key, and the digital signature MUST be validated
   directly with the decrypted contingency public key. Again, no
   delegation via a certification path is permitted.

   If the Apex Trust Anchor Update message is validated using the
   operational public key, then sequence number processing is handled
   normally, as described in Section 6. If the Apex Trust Anchor Update
   message is validated using the contingency public key, then the
   TAMPMsgRef sequence number MUST contain a zero value. A sequence
   number for subsequent messages that will be validated with the new
   operational public key can optionally be provided. If no value is
   provided, then the trust anchor store MUST be prepared to accept any
   sequence number in the next TAMP message validated with the newly
   installed apex trust anchor operational public key. If the Apex
   Trust Anchor Update message is valid and the clearTrustAnchors flag
   is set to TRUE, then all of the management and identity trust anchors
   stored in the trust anchor store MUST be deleted. That is, the new
   apex trust anchor MUST be the only trust anchor remaining in the
   trust anchor store. If the Apex Trust Anchor Update message is valid
   and the clearCommunities flag is set to TRUE, then all community
   identifiers stored in the trust anchor store MUST be deleted.

   The SignedData structure includes a SignerInfo.sid value, and it
   identifies the apex trust anchor public key that will be used to
   validate the digital signature on this TAMP message. The public key

identifier for the operational public key is known in advance, and it
is stored as part of the apex trust anchor. The public key
identifier for the contingency public key is not known in advance;
however, the presence of the unsigned attribute containing the
symmetric key needed to decrypt the contingency public key
unambiguously indicates that the TAMP message signer used the
contingency private key to sign the Apex Trust Anchor Update message.

If the digital signature on the Apex Trust Anchor Update message is
valid using either the apex trust anchor operational public key or
the apex trust anchor contingency public key, sequence number
checking is successful, and the trust anchor store is an intended
recipient of the TAMP message, then the trust anchor store MUST
update the apex trust anchor and return an Apex Trust Anchor Update
Confirm message. If an Apex Trust Anchor Update Confirm message is
not returned, then a TAMP Error message SHOULD be returned. Note
that the sequence number MUST be zero if the Apex Trust Anchor Update
message is validated with the apex trust anchor contingency public
key.

The Apex Trust Anchor Update content type has the following syntax:

```
tamp-apex-update CONTENT-TYPE  ::=
   { TAMPApexUpdate IDENTIFIED BY id-ct-TAMP-apexUpdate }

id-ct-TAMP-apexUpdate OBJECT IDENTIFIER ::= { id-tamp 5 }

TAMPApexUpdate ::= SEQUENCE {
   version            [0] TAMPVersion DEFAULT v2,
   terse              [1] TerseOrVerbose DEFAULT verbose,
   msgRef             TAMPMsgRef,
   clearTrustAnchors  BOOLEAN,
   clearCommunities   BOOLEAN,
   seqNumber          SeqNumber OPTIONAL,
   apexTA             TrustAnchorChoice }
```

The fields of TAMPApexUpdate are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o terse indicates the type of response that is desired. A terse
   response is indicated by a value of 1, and a verbose response is
   indicated by a value of 2, which is omitted during encoding since
   it is the default value.

o msgRef contains two items: the target and the seqNum. target
   identifies the target(s) of the Apex Trust Anchor Update message.

The TargetIdentifier syntax as described in Section 4.1 is used.
seqNum is a single-use value that will be used to match the Apex
Trust Anchor Update message with the Apex Trust Anchor Update
Confirm message. The sequence number is also used to detect TAMP
message replay if the message is validated with the apex trust
anchor operational public key. The sequence number processing
described in Section 6 MUST successfully complete before any
action is taken. However, seqNum MUST contain a zero value if the
message is validated with the apex trust anchor contingency
public key.

o clearTrustAnchors is a Boolean. If the value is set to TRUE, then
   all of the management and identity trust anchors stored in the
   trust anchor store MUST be deleted, leaving the newly installed
   apex trust anchor as the only trust anchor in the trust anchor
   store. If the value is set to FALSE, the other trust anchors MUST
   NOT be changed.

o clearCommunities is a Boolean. If the value is set to TRUE, then
   all of the community identifiers stored in the trust anchor store
   MUST be deleted, leaving none. If the value is set to FALSE, the
   list of community identifiers MUST NOT be changed.

o seqNumber is OPTIONAL, and when present, it provides the initial
   sequence number for the apex trust anchor. If seqNumber is
   absent, the trust anchor store is prepared to accept any sequence
   number value for the apex trust anchor operational public key.

o apexTA provides the information for the replacement apex trust
   anchor. The TrustAnchorChoice structure is used to provide the
   trusted public key and all of the information associated with it.
   The pubKey, keyId, taTitle, certPath, and exts fields apply to the
   operational public key of the apex trust anchor. The
   ApexTrustAnchorInfo certificate extension MAY appear as an
   extension. Section 9 describes the WrappedApexContingencyKey
   certificate extension.

4.6. Apex Trust Anchor Update Confirm

   The Apex Trust Anchor Update Confirm message is a reply by a trust
   anchor store to a valid Apex Trust Anchor Update message. The Apex
   Trust Anchor Update Confirm message provides success or failure
   information for the apex trust anchor update. The Apex Trust Anchor
   Update Confirm message MAY be signed or unsigned. An Apex Trust
   Anchor Update Confirm message MUST be signed if the trust anchor
   store is capable of signing it.

The Apex Trust Anchor Update Confirm content type has the following
syntax:

```
tamp-apex-update-confirm CONTENT-TYPE  ::=
   { TAMPApexUpdateConfirm IDENTIFIED BY
     id-ct-TAMP-apexUpdateConfirm }

id-ct-TAMP-apexUpdateConfirm OBJECT IDENTIFIER ::= { id-tamp 6 }

TAMPApexUpdateConfirm ::= SEQUENCE {
   version     [0] TAMPVersion DEFAULT v2,
   apexReplace  TAMPMsgRef,
   apexConfirm  ApexUpdateConfirm }

ApexUpdateConfirm ::= CHOICE {
   terseApexConfirm   [0] TerseApexUpdateConfirm,
   verboseApexConfirm [1] VerboseApexUpdateConfirm }

TerseApexUpdateConfirm ::= StatusCode

VerboseApexUpdateConfirm ::= SEQUENCE {
   status              StatusCode,
   taInfo              TrustAnchorChoiceList,
   communities         [0] CommunityIdentifierList OPTIONAL,
   tampSeqNumbers      [1] TAMPSequenceNumbers OPTIONAL }
```

The fields of TAMPApexUpdateConfirm are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o apexReplace identifies the Apex Trust Anchor Update message to
   which the trust anchor store is responding. The apexReplace
   structure repeats the TAMPMsgRef from the beginning of the Apex
   Trust Anchor Update message (see Section 4.5). When the Apex
   Trust Anchor Update message is validated with the operational
   public key, the sequence number processing described in Section 6
   MUST successfully complete before an Apex Trust Anchor Update
   Confirm message is generated. When the Apex Trust Anchor Update
   message is validated with the contingency public key, normal
   sequence number processing is ignored, but the seqNum MUST be
   zero.

o apexConfirm contains either a terse update confirmation or a
   verbose update confirmation. The terse update confirmation is
   represented by TerseApexUpdateConfirm, and the verbose response is
   represented by VerboseApexUpdateConfirm.

   The TerseApexUpdateConfirm contains a single status code, indicating
   the success or failure of the apex trust anchor update. If the apex
   trust anchor update failed, then the status code provides the reason
   for the failure. Each of the status codes is discussed in Section 5.

   The fields of VerboseApexUpdateConfirm are used as follows:

   o status contains a single status code, indicating the success or
      failure of the apex trust anchor update. If the apex trust anchor
      update failed, then the status code provides the reason for the
      failure. Each of the status codes is discussed in Section 5.

   o taInfo contains a sequence of TrustAnchorChoice structures. One
      entry in the sequence is provided for each trust anchor contained
      in the trust anchor store. These represent the state of the trust
      anchors after the apex trust anchor update has been processed.
      See [RFC5914] for a description of the TrustAnchorInfo structure.
      The apex trust anchor is the first trust anchor in the sequence.

   o communities is OPTIONAL. When present, it contains a sequence of
      object identifiers. Each object identifier names one community to
      which this trust anchor store belongs. When the trust anchor
      store belongs to no communities, this field is omitted.

   o tampSeqNumbers is used to indicate the currently held sequence
      number for each trust anchor authorized to sign TAMP messages.
      The keyId field identifies the trust anchor, and the seqNumber
      field provides the current sequence number associated with the
      trust anchor.

4.7. Community Update

   The trust anchor store maintains a list of identifiers for the
   communities of which it is a member. The Community Update message
   can be used to remove or add community identifiers from this list.
   The Community Update message MUST be signed. For the Community
   Update message to be valid, the trust anchor store MUST be a target
   of the update; the sequence number checking described in Section 6
   MUST be successful when the TAMP message signer is a trust anchor;
   and the digital signature MUST be validated by the apex trust anchor
   operational public key, an authorized management trust anchor, or via
   an authorized X.509 certification path originating with such a trust
   anchor.

   If the trust anchor store supports the Community Update message, the
   digital signature on the Community Update message is valid, sequence
   number checking is successful, the signer is authorized, and the
   trust anchor store is an intended recipient of the TAMP message, then

the trust anchor store MUST make the specified updates and return a
Community Update Confirm message. If a Community Update Confirm
message is not returned, then a TAMP Error message SHOULD be
returned.

The Community Update message contains a batch of updates, and all of
the updates MUST be accepted for the trust anchor store to return a
successful Community Update Confirm message. The remove updates, if
present, MUST be processed before the add updates. Where remove is
present with an empty list, all community identifiers MUST be
removed. This approach prevents community identifiers that are
intended to be mutually exclusive from being installed by a
successful addition and a failed removal. Where add is present, at
least one community identifier MUST appear in the list.

The Community Update content type has the following syntax:

```
 tamp-community-update CONTENT-TYPE  ::=
    { TAMPCommunityUpdate IDENTIFIED BY id-ct-TAMP-communityUpdate }

 id-ct-TAMP-communityUpdate OBJECT IDENTIFIER ::= { id-tamp 7 }

 TAMPCommunityUpdate ::= SEQUENCE {
    version  [0] TAMPVersion DEFAULT v2,
    terse    [1] TerseOrVerbose DEFAULT verbose,
    msgRef   TAMPMsgRef,
    updates  CommunityUpdates }

 CommunityUpdates ::= SEQUENCE {
    remove     [1] CommunityIdentifierList OPTIONAL,
    add        [2] CommunityIdentifierList OPTIONAL }
     -- At least one MUST be present
```

The fields of TAMPCommunityUpdate are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o terse indicates the type of response that is desired. A terse
   response is indicated by a value of 1, and a verbose response is
   indicated by a value of 2, which is omitted during encoding since
   it is the default value.

o msgRef contains two items: the target and the seqNum. target
   identifies the target(s) of the update message. The
   TargetIdentifier syntax as described in Section 4.1 is used.
   seqNum is a single-use value that will be used to match the
   Community Update message with the Community Update Confirm

      message. The sequence number is also used to detect TAMP message
      replay. The sequence number processing described in Section 6
      MUST successfully complete before any of the updates are
      processed.

   o updates contains a sequence of community identifiers to be removed
      and a sequence of community identifiers to be added. These are
      represented by the CommunityUpdates structure.

   The CommunityUpdates is a sequence of two OPTIONAL sequences, but at
   least one of these sequences MUST be present. The first sequence
   contains community identifiers to be removed, and if there are none,
   it is absent. Where remove is present with an empty list, all
   community identifiers MUST be removed. The second sequence contains
   community identifiers to be added, and if there are none, it is
   absent. The remove updates, if present, MUST be processed before the
   add updates. An error is generated if any of the requested removals
   or additions cannot be accomplished. However, requests to remove
   community identifiers that are not present are treated as successful
   removals. Likewise, requests to add community identifiers that are
   already present are treated as successful additions. If an error is
   generated, the trust anchor store community list MUST NOT be changed.

   A description of the syntax associated with each of these actions
   follows:

   o remove is used to remove one, multiple, or all community
      identifiers from the trust anchor store.

   o add is used to insert one or more new community identifiers into
      the trust anchor store.

4.8. Community Update Confirm

   The Community Update Confirm message is a reply by a trust anchor
   store to a valid Community Update message. The Community Update
   Confirm message provides success or failure information for the
   requested updates. Success is returned only if the whole batch of
   updates is successfully processed. If any of the requested updates
   cannot be performed, then a failure is indicated, and the set of
   community identifiers stored in the trust anchor store is unchanged.
   The Community Update Confirm message MAY be signed or unsigned. A
   Community Update Confirm message MUST be signed if the trust anchor
   store is capable of signing it.

The Community Update Confirm content type has the following syntax:

```
tamp-community-update-confirm CONTENT-TYPE  ::=
   { TAMPCommunityUpdateConfirm IDENTIFIED BY
     id-ct-TAMP-communityUpdateConfirm }

id-ct-TAMP-communityUpdateConfirm OBJECT IDENTIFIER ::=
   { id-tamp 8 }

TAMPCommunityUpdateConfirm ::= SEQUENCE {
   version      [0] TAMPVersion DEFAULT v2,
   update       TAMPMsgRef,
   commConfirm  CommunityConfirm }

CommunityConfirm ::= CHOICE {
   terseCommConfirm     [0] TerseCommunityConfirm,
   verboseCommConfirm   [1] VerboseCommunityConfirm }

TerseCommunityConfirm ::= StatusCode

VerboseCommunityConfirm ::= SEQUENCE {
   status       StatusCode,
   communities  CommunityIdentifierList OPTIONAL }
```

The fields of TAMPCommunityUpdateConfirm are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o update identifies the Community Update message to which the trust
   anchor store is responding. The update structure repeats the
   TAMPMsgRef from the Community Update message (see Section 4.7).
   The sequence number processing described in Section 6 MUST
   successfully complete before any of the updates are processed.

o commConfirm contains either a terse community update confirmation
   or a verbose community update confirmation. The terse response is
   represented by TerseCommunityConfirm, and the verbose response is
   represented by VerboseCommunityConfirm.

The TerseCommunityConfirm contains a single status code, indicating
the success or failure of the Community Update message processing.
If the community update failed, then the status code indicates the
reason for the failure. Each of the status codes is discussed in
Section 5.

   The fields of VerboseCommunityConfirm are used as follows:

   o status contains a single status code, indicating the success or
      failure of the Community Update message processing. If the
      community update failed, then the status code indicates the reason
      for the failure. Each of the status codes is discussed in
      Section 5.

   o communities is OPTIONAL. When present, it contains the sequence
      of community identifiers present in the trust anchor store after
      the update is processed. When the trust anchor store belongs to
      no communities, this field is omitted.

4.9. Sequence Number Adjust

   The trust anchor store maintains the current sequence number for the
   apex trust anchor and each management trust anchor authorized for
   TAMP messages. Sequence number processing is discussed in Section 6.
   The Sequence Number Adjust message can be used to provide the most
   recently used sequence number to one or more targets, thereby
   reducing the possibility of replay. The Sequence Number Adjust
   message MUST be signed. For the Sequence Number Adjust message to be
   valid, the trust anchor store MUST be an intended recipient of the
   Sequence Number Adjust message, the sequence number MUST be equal to
   or larger than the most recently stored sequence number for the
   originating trust anchor, and the digital signature MUST be validated
   by the apex trust anchor operational public key or an authorized
   management trust anchor.

   If the digital signature on the Sequence Number Adjust message is
   valid, the sequence number is equal to or larger than the most
   recently stored sequence number for the originating trust anchor, the
   signer is authorized, and the trust anchor store is an intended
   recipient of the TAMP message, then the trust anchor store MUST
   update the sequence number associated with the originating trust
   anchor and return a Sequence Number Adjust Confirm message. If a
   Sequence Number Adjust Confirm message is not returned, then a TAMP
   Error message SHOULD be returned.

   The Sequence Number Adjust message contains an adjustment for the
   sequence number of the TAMP message signer.

The Sequence Number Adjust content type has the following syntax:

```
tamp-sequence-number-adjust CONTENT-TYPE  ::=
   { SequenceNumberAdjust IDENTIFIED BY id-ct-TAMP-seqNumAdjust }

id-ct-TAMP-seqNumAdjust OBJECT IDENTIFIER ::= { id-tamp 10 }

SequenceNumberAdjust ::= SEQUENCE {
   Version  [0] TAMPVersion DEFAULT v2,
   msgRef   TAMPMsgRef }
```

The fields of SequenceNumberAdjust are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o msgRef contains two items: the target and the seqNum. target
   identifies the target(s) of the sequence number adjust message.
   The TargetIdentifier syntax as described in Section 4.1 is used.
   The allModules target is expected to be used for Sequence Number
   Adjust messages. seqNum MUST be equal to or larger than the most
   recently stored sequence number for this TAMP message signer, and
   the value will be used to match the Sequence Number Adjust message
   with the Sequence Number Adjust Confirm message. The sequence
   number processing described in Section 6 applies, except that the
   sequence number in a Sequence Number Adjust message is acceptable
   if it matches the most recently stored sequence number for this
   TAMP message signer. If sequence number checking completes
   successfully, then the sequence number is adjusted; otherwise, it
   remains unchanged.

4.10. Sequence Number Adjust Confirm

   The Sequence Number Adjust Confirm message is a reply by a trust
   anchor store to a valid Sequence Number Adjust message. The Sequence
   Number Adjust Confirm message provides success or failure
   information. Success is returned only if the sequence number for the
   trust anchor that signed the Sequence Number Adjust message
   originator is adjusted. If the sequence number cannot be adjusted,
   then a failure is indicated, and the sequence number stored in the
   trust anchor store is unchanged. The Sequence Number Adjust Confirm
   message MAY be signed or unsigned. A Sequence Number Adjust Confirm
   message MUST be signed if the trust anchor store is capable of
   signing it.

   The Sequence Number Adjust Confirm content type has the following
   syntax:

     tamp-sequence-number-adjust-confirm CONTENT-TYPE ::=
        { SequenceNumberAdjustConfirm IDENTIFIED BY
          id-ct-TAMP-seqNumAdjustConfirm }

     id-ct-TAMP-seqNumAdjustConfirm OBJECT IDENTIFIER ::=
        { id-tamp 11 }

     SequenceNumberAdjustConfirm ::= SEQUENCE {
        version [0] TAMPVersion DEFAULT v2,
        adjust   TAMPMsgRef,
        status   StatusCode }

   The fields of SequenceNumberAdjustConfirm are used as follows:

   o version identifies version of TAMP. For this version of the
      specification, the default value, v2, MUST be used.

   o adjust identifies the Sequence Number Adjust message to which the
      trust anchor store is responding. The adjust structure repeats
      the TAMPMsgRef from the Sequence Number Adjust message (see
      Section 4.9). The sequence number processing described in
      Section 6 MUST successfully complete to adjust the sequence number
      associated with the Sequence Number Adjust message originator.

   o status contains a single status code, indicating the success or
      failure of the Sequence Number Adjust message processing. If the
      adjustment failed, then the status code indicates the reason for
      the failure. Each of the status codes is discussed in Section 5.

4.11. TAMP Error

   The TAMP Error message is a reply by a trust anchor store to any
   invalid TAMP message. The TAMP Error message provides an indication
   of the reason for the error. The TAMP Error message MAY be signed or
   unsigned. A TAMP Error message MUST be signed if the trust anchor
   store is capable of signing it. For the request types defined in
   this specification, TAMP Error messages MUST NOT be used to indicate
   a request message was successfully processed. Each TAMP Error
   message identifies the type of TAMP message that caused the error.
   In cases where the TAMP message type cannot be determined, errors MAY
   be returned via other means, such as at the protocol level, via an
   attached display, etc.

The TAMP Error message content type has the following syntax:

```
tamp-error CONTENT-TYPE  ::=
   { TAMPError IDENTIFIED BY id-ct-TAMP-error }

id-ct-TAMP-error OBJECT IDENTIFIER ::= { id-tamp 9 }

TAMPError ::= SEQUENCE {
  version  [0] TAMPVersion DEFAULT v2,
  msgType  OBJECT IDENTIFIER,
  status   StatusCode,
  msgRef   TAMPMsgRef OPTIONAL }
```

The fields of TAMPError are used as follows:

o version identifies version of TAMP. For this version of the
   specification, the default value, v2, MUST be used.

o msgType indicates the content type of the TAMP message that caused
   the error.

o status contains a status code that indicates the reason for the
   error. Each of the status codes is discussed in Section 5.

o msgRef is OPTIONAL, but whenever possible it SHOULD be present.
   It identifies the TAMP message that caused the error. It repeats
   the target and seqNum from the TAMP message that caused the error
   (see Sections 4.1, 4.3, 4.5, 4.7, and 4.9).

5. Status Codes

   The Trust Anchor Update Confirm, the Apex Trust Anchor Update
   Confirm, the Community Update Confirm, the Sequence Number Adjust
   Confirm, and the TAMP Error messages include status codes. The
   syntax for the status codes is:

```
   StatusCode ::= ENUMERATED {
     success                        (0),
     decodeFailure                  (1),
     badContentInfo                 (2),
     badSignedData                  (3),
     badEncapContent                (4),
     badCertificate                 (5),
     badSignerInfo                  (6),
     badSignedAttrs                 (7),
     badUnsignedAttrs               (8),
     missingContent                 (9),
     noTrustAnchor                  (10),
```

```
      notAuthorized                       (11),
      badDigestAlgorithm                  (12),
      badSignatureAlgorithm               (13),
      unsupportedKeySize                  (14),
      unsupportedParameters               (15),
      signatureFailure                    (16),
      insufficientMemory                  (17),
      unsupportedTAMPMsgType              (18),
      apexTAMPAnchor                      (19),
      improperTAAddition                  (20),
      seqNumFailure                       (21),
      contingencyPublicKeyDecrypt         (22),
      incorrectTarget                     (23),
      communityUpdateFailed               (24),
      trustAnchorNotFound                 (25),
      unsupportedTAAlgorithm              (26),
      unsupportedTAKeySize                (27),
      unsupportedContinPubKeyDecryptAlg (28),
      missingSignature                    (29),
      resourcesBusy                       (30),
      versionNumberMismatch               (31),
      missingPolicySet                    (32),
      revokedCertificate                  (33),
      unsupportedTrustAnchorFormat        (34),
      improperTAChange                    (35),
      malformed                           (36),
      cmsError                            (37),
      unsupportedTargetIdentifier         (38),
      other                             (127) }
```

   The various values of StatusCode are used as follows:

   o success is used to indicate that an update, portion of an update,
      or adjust was processed successfully.

   o decodeFailure is used to indicate that the trust anchor store was
      unable to successfully decode the provided message. The specified
      content type and the provided content do not match.

   o badContentInfo is used to indicate that the ContentInfo syntax is
      invalid or that the contentType carried within the ContentInfo is
      unknown or unsupported.

   o badSignedData is used to indicate that the SignedData syntax is
      invalid, the version is unknown or unsupported, or more than one
      entry is present in digestAlgorithms.

o badEncapContent is used to indicate that the
   EncapsulatedContentInfo syntax is invalid. This error can be
   generated due to problems located in SignedData.

o badCertificate is used to indicate that the syntax for one or more
   certificates in CertificateSet is invalid.

o badSignerInfo is used to indicate that the SignerInfo syntax is
   invalid, or the version is unknown or unsupported.

o badSignedAttrs is used to indicate that the signedAttrs syntax
   within SignerInfo is invalid.

o badUnsignedAttrs is used to indicate that the unsignedAttrs syntax
   within SignerInfo is invalid.

o missingContent is used to indicate that the OPTIONAL eContent is
   missing in EncapsulatedContentInfo, which is REQUIRED in this
   specification. This error can be generated due to problems
   located in SignedData.

o noTrustAnchor is used to indicate one of two possible error
   situations. In one case, the subjectKeyIdentifier does not
   identify the public key of a trust anchor or a certification path
   that terminates with an installed trust anchor. In the other
   case, the issuerAndSerialNumber is used to identify the TAMP
   message signer, which is prohibited by this specification.

o notAuthorized is used to indicate one of two possible error
   situations. In one case, the sid within SignerInfo leads to an
   installed trust anchor, but that trust anchor is not an authorized
   signer for the received TAMP message content type. Identity trust
   anchors are not authorized signers for any of the TAMP message
   content types. In the other case, the signer of a Trust Anchor
   Update message is not authorized to manage the to-be-updated trust
   anchor as determined by a failure of the subordination processing
   in Section 7.

o badDigestAlgorithm is used to indicate that the digestAlgorithm in
   either SignerInfo or SignedData is unknown or unsupported.

o badSignatureAlgorithm is used to indicate that the
   signatureAlgorithm in SignerInfo is unknown or unsupported.

o unsupportedKeySize is used to indicate that the signatureAlgorithm
   in SignerInfo is known and supported, but the TAMP message digital
   signature could not be validated because an unsupported key size
   was employed by the signer.

o unsupportedParameters is used to indicate that the
   signatureAlgorithm in SignerInfo is known, but the TAMP message
   digital signature could not be validated because unsupported
   parameters were employed by the signer.

o signatureFailure is used to indicate that the signatureAlgorithm
   in SignerInfo is known and supported, but the digital signature in
   the signature field within SignerInfo could not be validated.

o insufficientMemory indicates that the update could not be
   processed because the trust anchor store did not have sufficient
   memory to store the resulting trust anchor configuration or
   community identifier.

o unsupportedTAMPMsgType indicates that the TAMP message could not
   be processed because the trust anchor store does not support the
   provided TAMP message type. This code will be used if the
   id-ct-TAMP-communityUpdate content type is provided and the trust
   anchor store does not support the Community Update message. This
   status code will also be used if the contentType value within
   eContentType is not one that is defined in this specification.

o apexTAMPAnchor indicates that the update could not be processed
   because the Trust Anchor Update message tried to remove the apex
   trust anchor.

o improperTAAddition indicates that a trust anchor update is trying
   to add a new trust anchor that may already exist, but some
   attributes of the to-be-added trust anchor are being modified in
   an improper manner. The desired trust anchor configuration may be
   attainable with a change operation instead of an add operation.

o seqNumFailure indicates that the TAMP message could not be
   processed because the processing of the sequence number, which is
   described in Section 6, resulted in an error.

o contingencyPublicKeyDecrypt indicates that the update could not be
   processed because an error occurred while decrypting the
   contingency public key.

o incorrectTarget indicates that the query, update, or adjust
   message could not be processed because the trust anchor store is
   not the intended recipient.

o communityUpdateFailed indicates that the community update
   requested the addition of a community identifier or the removal of
   a community identifier, but the request could not be honored.

o trustAnchorNotFound indicates that a change to a trust anchor was
   requested, but the referenced trust anchor is not represented in
   the trust anchor store.

o unsupportedTAAlgorithm indicates that an update message would
   result in the trust anchor with a public key associated with a
   digital signature validation algorithm that is not implemented.
   In addition, this status code is used if the algorithm is
   supported, but the parameters associated with the algorithm are
   not supported.

o unsupportedTAKeySize indicates that the trust anchor would include
   a public key of a size that is not supported.

o unsupportedContinPubKeyDecryptAlg indicates that the decryption
   algorithm for the apex trust anchor contingency public key is not
   supported.

o missingSignature indicates that an unsigned TAMP message was
   received, but the received TAMP message type MUST be signed.

o resourcesBusy indicates that the resources necessary to process
   the TAMP message are not available at the present time, but the
   resources might be available at some point in the future.

o versionNumberMismatch indicates that the version number in a
   received TAMP message is not acceptable.

o missingPolicySet indicates that the policyFlags associated with a
   trust anchor are set in a fashion that requires the policySet to
   be present, but the policySet is missing.

o revokedCertificate indicates that one or more of the certificates
   needed to properly process the TAMP message have been revoked.

o unsupportedTrustAnchorFormat indicates that an unsupported trust
   anchor format was presented or the version is unknown or
   unsupported.

o improperTAChange indicates that a trust anchor update is trying to
   change a new trust anchor using a format different than the format
   of the existing trust anchor.

o malformed indicates an error in the composition of the CMS
   structure encapsulating a TAMP message.

   o cmsError indicates an error processing a CMS structure that
      encapsulated a TAMP message, such as an error processing
      ContentType or MessageDigest attributes.

   o unsupportedTargetIdentifier indicates that a msgRef with an
      unsupported TargetIdentifier option was encountered.

   o other indicates that the update could not be processed, but the
      reason is not covered by any of the assigned status codes. Use of
      this status code SHOULD be avoided.

6. Sequence Number Processing

   The sequence number processing facilities in TAMP represent a balance
   between replay protection, operational considerations, and trust
   anchor store memory management. The goal is to provide replay
   protection without making TAMP difficult to use, creating an
   environment where surprising error conditions occur on a regular
   basis, or imposing onerous memory management requirements on
   implementations. This balance is achieved by performing sequence
   number checking on TAMP messages that are validated directly using a
   trust anchor, and allowing these checks to be skipped whenever the
   TAMP message originator is not represented by a trust anchor.
   Implementations MUST perform sequence number checking on TAMP
   messages that are validated directly using a trust anchor and MAY
   perform sequence number checking for TAMP messages validated using a
   certification path.

   The TAMP Status Query, Trust Anchor Update, Apex Trust Anchor Update,
   Community Update, and Sequence Number Adjust messages include a
   sequence number. This single-use identifier is used to match a TAMP
   message with the response to that TAMP message. When the TAMP
   message is validated directly using a trust anchor, the sequence
   number is also used to detect TAMP message replay.

   To provide replay protection, each TAMP message originator MUST treat
   the sequence number as a monotonically increasing non-negative
   integer. The sequence number counter is associated with the signing
   operation performed by the private key. The trust anchor store MUST
   ensure that a newly received TAMP message that is validated directly
   by a trust anchor public key contains a sequence number that is
   greater than the most recent successfully processed TAMP message from
   that originator. Note that the Sequence Number Adjust message is
   considered valid if the sequence number is greater than or equal to
   the most recent successfully processed TAMP message from that

originator. If the sequence number in a received TAMP message does
not meet these conditions, then the trust anchor store MUST reject
the TAMP message, returning a sequence number failure (seqNumFailure)
error.

Whenever a trust anchor is authorized for TAMP messages, either as a
newly installed trust anchor or as a modification to an existing
trust anchor, if a sequence number value is not provided in the Trust
Anchor Update message, memory MUST be allocated for the sequence
number and set to zero. The first TAMP message received that is
validated using that trust anchor is not rejected based on sequence
number checks, and the sequence number from that first TAMP message
is stored. The TAMP message recipient MUST maintain a database of
the most recent sequence number from a successfully processed TAMP
message from a trust anchor. The index for this database is the
trust anchor public key. This could be the apex trust anchor
operational public key or a management trust anchor public key. In
the first case, the apex trust anchor operational public key is used
directly to validate the TAMP message digital signature. In the
second case, a management trust anchor public key is used directly to
validate the TAMP message digital signature.

Sequence number values MUST be 64-bit non-negative integers. Since
ASN.1 encoding of an INTEGER always includes a sign bit, a TAMP
message signer can generate 9,223,372,036,854,775,807 TAMP messages
before exhausting the 64-bit sequence number space, before which the
TAMP message signer MUST transition to a different public/private key
pair. The ability to reset a sequence number provided by the Trust
Anchor Update and Sequence Number Adjust messages is not intended to
avoid the transition to a different key pair; rather, it is intended
to aid recovery from operational errors. A relatively small non-
volatile storage requirement is imposed on the trust anchor store for
the apex trust anchor and each management trust anchor authorized for
TAMP messages.

When the apex trust anchor or a management trust anchor is replaced
or removed from the trust anchor store, the associated sequence
number storage SHOULD be reclaimed.

7. Subordination Processing

   When a TAMP update message is processed, several checks are
   performed:

   o TAMP message authentication is checked including, if necessary,
      building and validating a certification path to the signer.

o The signer's authorization is checked, including authorization to
   manage trust anchors included in the update message.

o Calculation of the trust anchor information to be stored.

This section describes how to perform the second and third steps.
Section 1.2 discusses authentication of TAMP messages. Where a trust
anchor is represented as a certificate and the calculation of the
trust anchor information to be stored is different than the
information in the certificate, the TAMP update fails. The TAMP
message signer may then wrap the certificate inside a TrustAnchorInfo
structure to assert the intended information.

The apex trust anchor is unconstrained, which means that
subordination checking need not be performed on Trust Anchor Update
messages signed with the apex trust anchor operational public key and
that trust anchor information can be stored as it appears in the
update message. Subordination checking is performed as part of the
validation process of all other Trust Anchor Update messages.

For a Trust Anchor Update message that is not signed with the apex
trust anchor operational public key to be valid, the digital
signature MUST be validated using an authorized trust anchor, either
directly or via an X.509 certification path originating with the apex
trust anchor operational public key or an authorized management trust
anchor. The following subordination checks MUST also be performed as
part of validation of the update message.

Each Trust Anchor Update message contains one or more individual
updates, each of which is used to add, modify, or remove a trust
anchor. For each individual update, the constraints of the TAMP
message signer MUST be greater than or equal to the constraints of
the trust anchor in the update. Specifically, constraints included
in the CertPathControls field of a TrustAnchorInfo object (or
equivalent extensions in Certificate or TBSCertificate objects) must
be checked as described below. [RFC5280] describes how the
intersection and union operations referenced below are performed.

o The values of the policy flags stored with a trust anchor as the
   result of a TAMPUpdate are either true or equal to the value of
   the policy flags associated with the TAMP message signer, i.e., an
   update may set a flag to false only if the value associated with
   the TAMP message signer is false. The policy flags associated
   with the TAMP message signer are read from the policyFlags field
   or policyConstraints and inhibitAnyPolicy extensions if the signer

is represented as a trust anchor or from the explicit_policy,
policy_mapping, and inhibit_anyPolicy state variables following
path validation if the signer is not represented as a trust
anchor.

o The certificate policies stored with a trust anchor as the result
   of a TAMPUpdate are equal to the intersection of the value of the
   certificate policies associated with the TAMP message signer and
   the value of the policySet field or certificatePolicies extension
   from the update. The certificate policies associated with the
   TAMP message signer are read from the policySet field in a
   TrustAnchorInfo or certificatePolicies extension in a Certificate
   or TBSCertificate if the signer is represented as a trust anchor
   or from the valid_policy_tree returned following path validation
   if the signer is not represented by a trust anchor. Where the
   TAMP message signer is represented as a trust anchor, no policy
   mapping is performed. If the intersection is NULL and the
   to-be-stored requireExplicitPolicy value is true, the TAMP update
   fails.

o The excluded names stored with a trust anchor as the result of a
   TAMPUpdate are equal to the union of the excluded names associated
   with the TAMP message signer and the value from the nameConstr
   field or nameConstraints extension from the update. The name
   constraints associated with the TAMP message signer are read from
   the nameConstr field in a TrustAnchorInfo or nameConstraints
   extension in a Certificate or TBSCertificate if the signer is a
   trust anchor or from the excludedSubtrees state variable following
   path validation if the signer is not a trust anchor. The name of
   the trust anchor included in the update MUST NOT fall within the
   excluded name space of the TAMP signer. If the name of the trust
   anchor falls within the excluded name space of the TAMP signer,
   the TAMP update fails.

o The permitted names stored with a trust anchor as the result of a
   TAMPUpdate are equal to the intersection of the permitted names
   associated with the TAMP message signer and the value from the
   nameConstr field or nameConstraints extension from the update.
   The name constraints associated with the TAMP message signer are
   read from the nameConstr field in a TrustAnchorInfo or
   nameConstraints extension in a Certificate or TBSCertificate if
   the signer is a trust anchor or from the permittedSubtrees state
   variable following path validation if the signer is not a trust
   anchor. The name of the trust anchor included in the update MUST
   fall within the permitted name space of the TAMP signer. If the
   name of the trust anchor does not fall within the permitted name
   space of the TAMP signer, the TAMP update fails. If the
   intersection is NULL for all name forms, the TAMP update fails.

No other extensions defined in [RFC5280] must be processed as part of
subordination processing. Other extensions may define subordination
rules.

8. Implementation Considerations

   A public key identifier is used to identify a TAMP message signer.
   Since there is no guarantee that the same public key identifier is
   not associated with more than one public key, implementations MUST be
   prepared for one or more trust anchors to have the same public key
   identifier. In practical terms, this means that when a digital
   signature validation fails, the implementation MUST see if there is
   another trust anchor with the same public key identifier that can be
   used to validate the digital signature. While duplicate public key
   identifiers are expected to be rare, implementations MUST NOT fail to
   find the correct trust anchor when they do occur.

   An X.500 distinguished name is used to identify certificate issuers
   and certificate subjects. The same X.500 distinguished name can be
   associated with more than one trust anchor. However, the trust
   anchor public key will be different. The probability that two trust
   anchors will have the same X.500 distinguished name and the same
   public key identifier but a different public key is diminishingly
   small. Therefore, the authority key identifier certificate extension
   can be used to resolve X.500 distinguished name collisions.

   TAMP assumes a reliable underlying transport protocol.

9. Wrapped Apex Contingency Key Certificate Extension

   An apex trust anchor MAY contain contingency key information using
   the WrappedApexContingencyKey extension. The extension uses the
   ApexContingencyKey structure as defined below.

     ApexContingencyKey ::= SEQUENCE {
        wrapAlgorithm   AlgorithmIdentifier OPTIONAL,
        wrappedContinPubKey   OCTET STRING OPTIONAL }

   The fields of ApexContingencyKey are used as described below. When
   one field is present, both MUST be present. When one field is
   absent, both MUST be absent. The fields are allowed to be absent to
   enable usage of this extension as a means of indicating that the
   corresponding public key is recognized as an apex trust anchor by
   some relying parties.

   o wrapAlgorithm identifies the symmetric algorithm used to encrypt
      the apex trust anchor contingency public key. If this public key
      is ever needed, the symmetric key needed to decrypt it will be

provided in the message that is to be validated using it. The
algorithm identifier is an AlgorithmIdentifier, which contains an
object identifier and OPTIONAL parameters. The object identifier
indicates the syntax of the parameters, if present.

o wrappedContinPubKey is the encrypted apex trust anchor contingency
   public key. Once decrypted, it yields the PublicKeyInfo
   structure, which consists of the algorithm identifier followed by
   the public key itself. The algorithm identifier is an
   AlgorithmIdentifier that contains an object identifier and
   OPTIONAL parameters. The object identifier indicates the format
   of the public key and the syntax of the parameters, if present.
   The public key is encoded as a BIT STRING.

The WrappedApexContingencyKey certificate extension MAY be critical,
and it MUST appear at most one time in a set of extensions. The apex
trust anchor info extension is identified by the
id-pe-wrappedApexContinKey object identifier:

        id-pe-wrappedApexContinKey OBJECT IDENTIFIER ::=
            { iso(1) identified-organization(3) dod(6) internet(1)
              security(5) mechanisms(5) pkix(7) pe(1) 20 }

10. Security Considerations

The majority of this specification is devoted to the syntax and
semantics of TAMP messages. It relies on other specifications,
especially [RFC5914], [RFC3852], and [RFC5280], for the syntax and
semantics of trust anchors, intermediate CMS content types, and X.509
certificates, respectively. Since TAMP messages that change the
trust anchor state of a trust anchor store are always signed by a
Trust Anchor Manager, no further data integrity or data origin
authentication mechanisms are needed; however, no confidentiality for
these messages is provided. Similarly, certificates are digitally
signed, and no additional data integrity or data origin
authentication mechanisms are needed. Trust anchor configurations,
Trust Anchor Manager certificates, and trust anchor store
certificates are not intended to be sensitive. As a result, this
specification does not provide for confidentiality of TAMP messages.

Security factors outside the scope of this specification greatly
affect the assurance provided. The procedures used by certification
authorities (CAs) to validate the binding of the subject identity to
their public key greatly affect the assurance associated with the
resulting certificate. This is particularly important when issuing
certificates to other CAs. In the context of TAMP, the issuance of
an end entity certificate under a management trust anchor is an act
of delegation. However, such end entities cannot further delegate.

On the other hand, issuance of a CA certificate under a management
trust anchor is an act of delegation where the CA can perform further
delegation. The scope of the delegation can be constrained by
including appropriate certificate extensions in a CA certificate.

X.509 certification path construction involves comparison of X.500
distinguished names. Inconsistent application of name comparison
rules can result in acceptance of invalid X.509 certification paths
or rejection of valid ones. Name comparison can be extremely
complex. To avoid imposing this complexity on trust anchor stores,
any certificate profile used with TAMP SHOULD employ simple name
structures and impose rigorous restrictions on acceptable
distinguished names, including the way that they are encoded. The
goal of that certificate profile should be to enable simple binary
comparison. That is, case conversion, character set conversion,
white space compression, and leading and trailing white space
trimming SHOULD be avoided.

Some digital signature algorithms (DSAs) require the generation of
random one-time values. For example, when generating a DSA digital
signature, the signer MUST generate a random k value [DSS]. Also,
the generation of public/private key pairs relies on random numbers.

The use of an inadequate random number generator (RNG) or an
inadequate pseudo-random number generator (PRNG) to generate such
cryptographic values can result in little or no security. An
attacker may find it much easier to reproduce the random number
generation environment, searching the resulting small set of
possibilities, rather than brute-force searching the whole space.

Compromise of an identity trust anchor private key permits
unauthorized parties to issue certificates that will be acceptable to
all trust anchor stores configured with the corresponding identity
trust anchor. The unauthorized private key holder will be limited by
the certification path controls associated with the identity trust
anchor. For example, clearance constraints in the identity trust
anchor will determine the clearances that will be accepted in
certificates that are issued by the unauthorized private key holder.

Compromise of a management trust anchor private key permits
unauthorized parties to generate signed messages that will be
acceptable to all trust anchor stores configured with the
corresponding management trust anchor. All devices that include the
compromised management trust anchor can be configured as desired by
the unauthorized private key holder within the limits of the
subordination checks described in Section 7. If the management trust
anchor is associated with content types other than TAMP, then the
unauthorized private key holder can generate signed messages of that

type. For example, if the management trust anchor is associated with
firmware packages, then the unauthorized private key holder can
install different firmware.

Compromise of the apex trust anchor operational private key permits
unauthorized parties to generate signed messages that will be
acceptable to all trust anchor stores configured with the
corresponding apex trust anchor. All devices that include that apex
trust anchor can be configured as desired by the unauthorized private
key holder, and the unauthorized private key holder can generate
signed messages of any content type. The optional contingency
private key offers a potential way to recover from such a compromise.

The compromise of a CA's private key leads to the same type of
problems as the compromise of an identity or a management trust
anchor private key. The unauthorized private key holder will be
limited by the certification path controls and extensions associated
with the trust anchor.

The compromise of an end entity private key leads to the same type of
problems as the compromise of an identity or a management trust
anchor private key, except that the end entity is unable to issue any
certificates. The unauthorized private key holder will be limited by
the certification path controls and extensions associated with the
trust anchor.

Compromise of a trust anchor store's digital signature private key
permits unauthorized parties to generate signed TAMP response
messages, masquerading as the trust anchor store.

Premature disclosure of the key-encryption key used to encrypt the
apex trust anchor contingency public key may result in early exposure
of the apex trust anchor contingency public key.

TAMP implementations need to be able to parse messages and
certificates. Care must be taken to ensure that there are no
implementation defects in the TAMP message parser or the processing
that acts on the message content. A validation suite is one way to
increase confidence in the parsing of TAMP messages, CMS content
types, attributes, certificates, and extensions.

TrustAnchorList messages do not provide a replay detection mechanism.
Where TrustAnchorList messages are accepted as an alternative means
of adding trust anchors to a trust anchor store, applications may
require additional mechanisms to address the risks associated with
replay of old TrustAnchorList messages.

As sequence number values are used to detect replay attempts, trust
anchor store managers must take care to maintain their own sequence
number state, i.e., knowledge of which sequence number to include in
the next TAMP message generated by the trust anchor store manager.
Loss of sequence number state can result in generation of TAMP
messages that cannot be processed due to seqNumFailure. In the event
of loss, sequence number state can be restored by inspecting the most
recently generated TAMP message, provided the messages are logged, or
in collaboration with a trust anchor store manager who can
successfully issue a TAMPStatusQuery message.

11. IANA Considerations

The details of TAMP requests and responses are communicated using
object identifiers (OIDs). The objects are defined in an arc
delegated by IANA to the PKIX working group. This document also
includes eleven media type registrations in Appendix B. No further
action by IANA is necessary for this document or any anticipated
updates.

12. References

12.1. Normative References

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

[RFC2616]   Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
                 Masinter, L., Leach, P., and T. Berners-Lee,
                 "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616,
                 June 1999.

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

[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.

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

[RFC5912]   Hoffman, P. and J. Schaad, "New ASN.1 Modules for the
                 Public Key Infrastructure Using X.509 (PKIX)", RFC
                 5912, June 2010.

   [RFC5914]   Housley, R., Ashmore, S., and C. Wallace, "Trust
                      Anchor Format", RFC 5914, June 2010.

   [X.680]   "ITU-T Recommendation X.680 - Information Technology
                      - Abstract Syntax Notation One", 1997.

   [X.690]   "ITU-T Recommendation X.690 - Information Technology
                      - ASN.1 encoding rules: Specification of Basic
                      Encoding Rules (BER), Canonical Encoding Rules (CER)
                      and Distinguished Encoding Rules (DER)", 1997.

12.2. Informative References

   [DSS]   "FIPS Pub 186: Digital Signature Standard", May 1994.

   [PKCS#6]   "PKCS #6: Extended-Certificate Syntax Standard,
                      Version 1.5", November 1993.

   [RFC3279]   Bassham, L., Polk, W., and R. Housley, "Algorithms
                      and Identifiers for the Internet X.509 Public Key
                      Infrastructure Certificate and Certificate Revocation
                      List (CRL) Profile", RFC 3279, April 2002.

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

   [RFC4049]   Housley, R., "BinaryTime: An Alternate Format for
                      Representing Date and Time in ASN.1", RFC 4049, April
                      2005.

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

   [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.

   [RFC5755]   Farrell, S., Housley, R., and S. Turner, "An Internet
                      Attribute Certificate Profile for Authorization", RFC
                      5755, January 2010.

   [TA-MGMT-REQS]   Reddy, R. and C. Wallace, "Trust Anchor Management
                      Requirements", Work in Progress, March 2010.

   [X.208]   "ITU-T Recommendation X.208 - Specification of
                      Abstract Syntax Notation One (ASN.1)", 1988.

   [X.509]   "ITU-T Recommendation X.509 - The Directory -
                      Authentication Framework", 2000.

Appendix A. ASN.1 Modules

   Appendix A.1 provides the normative ASN.1 definitions for the
   structures described in this specification using ASN.1 as defined in
   [X.680]. Appendix A.2 provides a module using ASN.1 as defined in
   [X.208]. The module in Appendix A.2 removes usage of newer ASN.1
   features that provide support for limiting the types of elements that
   may appear in certain SEQUENCE and SET constructions. Otherwise, the
   modules are compatible in terms of encoded representation, i.e., the
   modules are bits-on-the-wire compatible aside from the limitations on
   SEQUENCE and SET constituents. Extension markers are not used due to
   lack of support in [X.208]. Appendix A.2 is included as a courtesy
   to developers using ASN.1 compilers that do not support current
   ASN.1. Appendix A.1 includes definitions imported from [RFC5280],
   [RFC5912], and [RFC5914].

A.1. ASN.1 Module Using 1993 Syntax

   TAMP-Protocol-v2
   { joint-iso-ccitt(2) country(16) us(840) organization(1)
     gov(101) dod(2) infosec(1) modules(0) 30 }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   IMPORTS
   TrustAnchorChoice, TrustAnchorTitle, CertPathControls
   FROM TrustAnchorInfoModule
      { joint-iso-ccitt(2) country(16) us(840)
      organization(1) gov(101) dod(2) infosec(1)
      modules(0) 33 }
   AlgorithmIdentifier{}, SIGNATURE-ALGORITHM, KEY-WRAP
   FROM AlgorithmInformation-2009
      {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58)}
   Certificate, Name, TBSCertificate,
   CertificateSerialNumber, Validity, SubjectPublicKeyInfo
   FROM PKIX1Explicit-2009 -- from [RFC5912]
      {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-explicit-02(51)}
   KeyIdentifier, OTHER-NAME
   FROM PKIX1Implicit-2009 -- from [RFC5912]
      {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-implicit-02(59)}
   EXTENSION, Extensions {}, ATTRIBUTE, SingleAttribute{}

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

   -- Object Identifier Arc for TAMP Message Content Types

   id-tamp OBJECT IDENTIFIER ::= {
   joint-iso-ccitt(2) country(16) us(840) organization(1)
   gov(101) dod(2) infosec(1) formats(2) 77 }

   SupportedSigAlgorithms SIGNATURE-ALGORITHM ::= {
      -- add any locally defined algorithms here
      ...
   }

   SupportedWrapAlgorithms KEY-WRAP ::= {
      -- add any locally defined algorithms here
      ...
   }

   -- CMS Content Types

   CONTENT-TYPE   ::= TYPE-IDENTIFIER

   TAMPContentTypes CONTENT-TYPE  ::= {
     tamp-status-query |
     tamp-status-response |
     tamp-update |
     tamp-update-confirm |
     tamp-apex-update |
     tamp-apex-update-confirm |
     tamp-community-update |
     tamp-community-update-confirm |
     tamp-sequence-number-adjust |
     tamp-sequence-number-adjust-confirm |
     tamp-error,
     ... -- Expect additional content types --
     }

   -- TAMP Status Query Message
   tamp-status-query CONTENT-TYPE  ::=
     { TAMPStatusQuery IDENTIFIED BY id-ct-TAMP-statusQuery }

   id-ct-TAMP-statusQuery OBJECT IDENTIFIER ::= { id-tamp 1 }
```

```
   TAMPStatusQuery ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     terse            [1] TerseOrVerbose DEFAULT verbose,
     query            TAMPMsgRef }

   TAMPVersion ::= INTEGER { v1(1), v2(2) }

   TerseOrVerbose ::= ENUMERATED { terse(1), verbose(2) }

   SeqNumber ::= INTEGER (0..9223372036854775807)

   TAMPMsgRef ::= SEQUENCE {
     target           TargetIdentifier,
     seqNum           SeqNumber }

   TargetIdentifier ::= CHOICE {
     hwModules    [1] HardwareModuleIdentifierList,
     communities  [2] CommunityIdentifierList,
     allModules   [3] NULL,
     uri          [4] IA5String,
     otherName    [5] INSTANCE OF OTHER-NAME }

   HardwareModuleIdentifierList ::= SEQUENCE SIZE (1..MAX) OF
                                      HardwareModules

   HardwareModules ::= SEQUENCE {
     hwType           OBJECT IDENTIFIER,
     hwSerialEntries SEQUENCE SIZE (1..MAX) OF HardwareSerialEntry }

   HardwareSerialEntry ::= CHOICE {
     all              NULL,
     single           OCTET STRING,
     block            SEQUENCE {
       low              OCTET STRING,
       high             OCTET STRING } }

   CommunityIdentifierList ::= SEQUENCE SIZE (0..MAX) OF Community

   Community ::= OBJECT IDENTIFIER

   -- TAMP Status Response Message

   tamp-status-response CONTENT-TYPE  ::=
     { TAMPStatusResponse IDENTIFIED BY id-ct-TAMP-statusResponse }

   id-ct-TAMP-statusResponse OBJECT IDENTIFIER ::= { id-tamp 2 }
```

```
   TAMPStatusResponse ::= SEQUENCE {
     version   [0] TAMPVersion DEFAULT v2,
     query     TAMPMsgRef,
     response  StatusResponse,
     usesApex  BOOLEAN DEFAULT TRUE }

   StatusResponse ::= CHOICE {
     terseResponse          [0] TerseStatusResponse,
     verboseResponse        [1] VerboseStatusResponse }

   TerseStatusResponse ::= SEQUENCE {
     taKeyIds              KeyIdentifiers,
     communities           CommunityIdentifierList OPTIONAL }

   KeyIdentifiers ::= SEQUENCE SIZE (1..MAX) OF KeyIdentifier

   VerboseStatusResponse ::= SEQUENCE {
     taInfo                TrustAnchorChoiceList,
     continPubKeyDecryptAlg [0] AlgorithmIdentifier
                  {KEY-WRAP, {SupportedWrapAlgorithms}} OPTIONAL,
     communities           [1] CommunityIdentifierList OPTIONAL,
     tampSeqNumbers        [2] TAMPSequenceNumbers OPTIONAL }

   TrustAnchorChoiceList ::= SEQUENCE SIZE (1..MAX) OF
       TrustAnchorChoice

   TAMPSequenceNumber ::= SEQUENCE {
     keyId       KeyIdentifier,
     seqNumber   SeqNumber }

   TAMPSequenceNumbers ::= SEQUENCE SIZE (1..MAX) OF TAMPSequenceNumber

   -- Trust Anchor Update Message

   tamp-update CONTENT-TYPE  ::=
     { TAMPUpdate IDENTIFIED BY id-ct-TAMP-update }

   id-ct-TAMP-update OBJECT IDENTIFIER ::= { id-tamp 3 }

   TAMPUpdate ::= SEQUENCE {
     version  [0] TAMPVersion DEFAULT v2,
     terse    [1] TerseOrVerbose DEFAULT verbose,
     msgRef   TAMPMsgRef,
     updates  SEQUENCE SIZE (1..MAX) OF TrustAnchorUpdate,
     tampSeqNumbers [2]TAMPSequenceNumbers OPTIONAL }
```

```
   TrustAnchorUpdate ::= CHOICE {
     add               [1] TrustAnchorChoice,
     remove            [2] SubjectPublicKeyInfo,
     change            [3] EXPLICIT TrustAnchorChangeInfoChoice }

   TrustAnchorChangeInfoChoice ::= CHOICE {
     tbsCertChange  [0] TBSCertificateChangeInfo,
     taChange       [1] TrustAnchorChangeInfo }

   TBSCertificateChangeInfo  ::=  SEQUENCE  {
     serialNumber         CertificateSerialNumber OPTIONAL,
     signature            [0] AlgorithmIdentifier
             {SIGNATURE-ALGORITHM, {SupportedSigAlgorithms}} OPTIONAL,
     issuer               [1] Name OPTIONAL,
     validity             [2] Validity OPTIONAL,
     subject              [3] Name OPTIONAL,
     subjectPublicKeyInfo [4] SubjectPublicKeyInfo,
     exts                 [5] EXPLICIT Extensions{{...}} OPTIONAL }

   TrustAnchorChangeInfo ::= SEQUENCE {
     pubKey           SubjectPublicKeyInfo,
     keyId            KeyIdentifier OPTIONAL,
     taTitle          TrustAnchorTitle OPTIONAL,
     certPath         CertPathControls OPTIONAL,
     exts             [1] Extensions{{...}} OPTIONAL }

   -- Trust Anchor Update Confirm Message

   tamp-update-confirm CONTENT-TYPE  ::=
     { TAMPUpdateConfirm IDENTIFIED BY id-ct-TAMP-updateConfirm }

   id-ct-TAMP-updateConfirm OBJECT IDENTIFIER ::= { id-tamp 4 }

   TAMPUpdateConfirm ::= SEQUENCE {
     version             [0] TAMPVersion DEFAULT v2,
     update              TAMPMsgRef,
     confirm             UpdateConfirm }

   UpdateConfirm ::= CHOICE {
     terseConfirm        [0] TerseUpdateConfirm,
     verboseConfirm      [1] VerboseUpdateConfirm }

   TerseUpdateConfirm ::= StatusCodeList

   StatusCodeList ::= SEQUENCE SIZE (1..MAX) OF StatusCode
```

```
   VerboseUpdateConfirm ::= SEQUENCE {
     status          StatusCodeList,
     taInfo          TrustAnchorChoiceList,
     tampSeqNumbers  TAMPSequenceNumbers OPTIONAL,
     usesApex        BOOLEAN DEFAULT TRUE }

   -- Apex Trust Anchor Update Message

   tamp-apex-update CONTENT-TYPE  ::=
       { TAMPApexUpdate IDENTIFIED BY id-ct-TAMP-apexUpdate }

   id-ct-TAMP-apexUpdate OBJECT IDENTIFIER ::= { id-tamp 5 }

   TAMPApexUpdate ::= SEQUENCE {
     version            [0] TAMPVersion DEFAULT v2,
     terse              [1] TerseOrVerbose DEFAULT verbose,
     msgRef             TAMPMsgRef,
     clearTrustAnchors  BOOLEAN,
     clearCommunities   BOOLEAN,
     seqNumber          SeqNumber OPTIONAL,
     apexTA             TrustAnchorChoice }

   -- Apex Trust Anchor Update Confirm Message

   tamp-apex-update-confirm CONTENT-TYPE  ::=
     { TAMPApexUpdateConfirm IDENTIFIED BY
         id-ct-TAMP-apexUpdateConfirm }

   id-ct-TAMP-apexUpdateConfirm OBJECT IDENTIFIER ::= { id-tamp 6 }

   TAMPApexUpdateConfirm ::= SEQUENCE {
     version             [0] TAMPVersion DEFAULT v2,
     apexReplace         TAMPMsgRef,
     apexConfirm         ApexUpdateConfirm }

   ApexUpdateConfirm ::= CHOICE {
     terseApexConfirm    [0] TerseApexUpdateConfirm,
     verboseApexConfirm  [1] VerboseApexUpdateConfirm }

   TerseApexUpdateConfirm ::= StatusCode

   VerboseApexUpdateConfirm ::= SEQUENCE {
     status              StatusCode,
     taInfo              TrustAnchorChoiceList,
     communities         [0] CommunityIdentifierList OPTIONAL,
     tampSeqNumbers      [1] TAMPSequenceNumbers OPTIONAL }
```

```
   -- Community Update Message

   tamp-community-update CONTENT-TYPE   ::=
     { TAMPCommunityUpdate IDENTIFIED BY id-ct-TAMP-communityUpdate }

   id-ct-TAMP-communityUpdate OBJECT IDENTIFIER ::= { id-tamp 7 }

   TAMPCommunityUpdate ::= SEQUENCE {
     version         [0] TAMPVersion DEFAULT v2,
     terse           [1] TerseOrVerbose DEFAULT verbose,
     msgRef          TAMPMsgRef,
     updates         CommunityUpdates }

   CommunityUpdates ::= SEQUENCE {
     remove      [1] CommunityIdentifierList OPTIONAL,
     add         [2] CommunityIdentifierList OPTIONAL }
     -- At least one must be present

   -- Community Update Confirm Message

   tamp-community-update-confirm CONTENT-TYPE   ::=
     { TAMPCommunityUpdateConfirm IDENTIFIED BY
       id-ct-TAMP-communityUpdateConfirm }

   id-ct-TAMP-communityUpdateConfirm OBJECT IDENTIFIER ::=
     { id-tamp 8 }

   TAMPCommunityUpdateConfirm ::= SEQUENCE {
     version         [0] TAMPVersion DEFAULT v2,
     update          TAMPMsgRef,
     commConfirm     CommunityConfirm }

   CommunityConfirm ::= CHOICE {
     terseCommConfirm      [0] TerseCommunityConfirm,
     verboseCommConfirm    [1] VerboseCommunityConfirm }

   TerseCommunityConfirm ::= StatusCode

   VerboseCommunityConfirm ::= SEQUENCE {
     status          StatusCode,
     communities     CommunityIdentifierList OPTIONAL }

   -- Sequence Number Adjust Message

   tamp-sequence-number-adjust CONTENT-TYPE   ::=
     { SequenceNumberAdjust IDENTIFIED BY id-ct-TAMP-seqNumAdjust }

   id-ct-TAMP-seqNumAdjust OBJECT IDENTIFIER ::= { id-tamp 10 }
```

```
   SequenceNumberAdjust ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,

     msgRef           TAMPMsgRef }

   -- Sequence Number Adjust Confirm Message

   tamp-sequence-number-adjust-confirm CONTENT-TYPE  ::=
     { SequenceNumberAdjustConfirm IDENTIFIED BY
       id-ct-TAMP-seqNumAdjustConfirm }

   id-ct-TAMP-seqNumAdjustConfirm OBJECT IDENTIFIER ::= { id-tamp 11 }

   SequenceNumberAdjustConfirm ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     adjust           TAMPMsgRef,
     status           StatusCode }

   -- TAMP Error Message

   tamp-error CONTENT-TYPE  ::=
     { TAMPError IDENTIFIED BY id-ct-TAMP-error }

   id-ct-TAMP-error OBJECT IDENTIFIER ::= { id-tamp 9 }

   TAMPError ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     msgType          OBJECT IDENTIFIER,
     status           StatusCode,
     msgRef           TAMPMsgRef OPTIONAL }

   -- Status Codes

   StatusCode ::= ENUMERATED {
     success                        (0),
     decodeFailure                  (1),
     badContentInfo                 (2),
     badSignedData                  (3),
     badEncapContent                (4),
     badCertificate                 (5),
     badSignerInfo                  (6),
     badSignedAttrs                 (7),
     badUnsignedAttrs               (8),
     missingContent                 (9),
     noTrustAnchor                 (10),
     notAuthorized                 (11),
     badDigestAlgorithm            (12),
     badSignatureAlgorithm         (13),
```

```
      unsupportedKeySize                 (14),
      unsupportedParameters              (15),
      signatureFailure                   (16),
      insufficientMemory                 (17),
      unsupportedTAMPMsgType             (18),
      apexTAMPAnchor                     (19),
      improperTAAddition                 (20),
      seqNumFailure                      (21),
      contingencyPublicKeyDecrypt        (22),
      incorrectTarget                    (23),
      communityUpdateFailed              (24),
      trustAnchorNotFound                (25),
      unsupportedTAAlgorithm             (26),
      unsupportedTAKeySize               (27),
      unsupportedContinPubKeyDecryptAlg  (28),
      missingSignature                   (29),
      resourcesBusy                      (30),
      versionNumberMismatch              (31),
      missingPolicySet                   (32),
      revokedCertificate                 (33),
      unsupportedTrustAnchorFormat       (34),
      improperTAChange                   (35),
      malformed                          (36),
      cmsError                           (37),
      unsupportedTargetIdentifier        (38),
      other                             (127) }

   -- Object Identifier Arc for Attributes

   id-attributes OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) country(16)
     us(840) organization(1) gov(101) dod(2) infosec(1) 5 }

   -- TAMP Unsigned Attributes
   -- These attributes are unsigned attributes and go into the
   --    UnsignedAttributes set in [RFC5652]

   TAMPUnsignedAttributes ATTRIBUTE ::= {
     contingency-public-key-decrypt-key,
     ... -- Expect additional attributes --
     }

   -- contingency-public-key-decrypt-key unsigned attribute

   contingency-public-key-decrypt-key ATTRIBUTE ::= {
     TYPE PlaintextSymmetricKey IDENTIFIED BY
     id-aa-TAMP-contingencyPublicKeyDecryptKey }
```

```
   id-aa-TAMP-contingencyPublicKeyDecryptKey OBJECT IDENTIFIER ::= {
     id-attributes 63 }

   PlaintextSymmetricKey ::= OCTET STRING

   -- id-pe-wrappedApexcontinKey extension

   wrappedApexContinKey EXTENSION ::= {
        SYNTAX          ApexContingencyKey
        IDENTIFIED BY   id-pe-wrappedApexContinKey }

   id-pe-wrappedApexContinKey OBJECT IDENTIFIER ::=
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) pe(1) 20 }

   ApexContingencyKey ::= SEQUENCE {
     wrapAlgorithm
        AlgorithmIdentifier{KEY-WRAP, {SupportedWrapAlgorithms}},
     wrappedContinPubKey  OCTET STRING }

   END

A.2.  ASN.1 Module Using 1988 Syntax

   TAMP-Protocol-v2-88
      { joint-iso-ccitt(2) country(16) us(840) organization(1)
        gov(101) dod(2) infosec(1) modules(0) 31 }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   IMPORTS
     TrustAnchorChoice, TrustAnchorTitle, CertPathControls
     FROM TrustAnchorInfoModule-88 -- from [RFC5914]
          { joint-iso-ccitt(2) country(16) us(840) organization(1)
            gov(101) dod(2) infosec(1) modules(0) 37 }
     AlgorithmIdentifier, Certificate, Name, Attribute, TBSCertificate,
     SubjectPublicKeyInfo, CertificateSerialNumber, Validity, Extensions
       FROM PKIX1Explicit88 -- from [RFC5280]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-pkix1-explicit(18) }
     KeyIdentifier, AnotherName
       FROM PKIX1Implicit88 -- from [RFC5280]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-pkix1-implicit(19) } ;
```

```
-- Object Identifier Arc for TAMP Message Content Types

id-tamp OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) country(16)
us(840) organization(1) gov(101) dod(2) infosec(1) formats(2) 77 }

-- CMS Content Types

-- TAMP Status Query Message

id-ct-TAMP-statusQuery OBJECT IDENTIFIER ::= { id-tamp 1 }

TAMPStatusQuery ::= SEQUENCE {
  version        [0] TAMPVersion DEFAULT v2,
  terse          [1] TerseOrVerbose DEFAULT verbose,
  query              TAMPMsgRef }

TAMPVersion ::= INTEGER { v1(1), v2(2) }

TerseOrVerbose ::= ENUMERATED { terse(1), verbose(2) }

SeqNumber ::= INTEGER (0..9223372036854775807)

TAMPMsgRef ::= SEQUENCE {
  target            TargetIdentifier,
  seqNum            SeqNumber }

TargetIdentifier ::= CHOICE {
  hwModules    [1] HardwareModuleIdentifierList,
  communities  [2] CommunityIdentifierList,
  allModules   [3] NULL,
  uri          [4] IA5String,
  otherName    [5] AnotherName }

HardwareModuleIdentifierList ::= SEQUENCE SIZE (1..MAX) OF
                                  HardwareModules

HardwareModules ::= SEQUENCE {
  hwType         OBJECT IDENTIFIER,
  hwSerialEntries SEQUENCE SIZE (1..MAX) OF HardwareSerialEntry }

HardwareSerialEntry ::= CHOICE {
  all            NULL,
  single         OCTET STRING,
  block          SEQUENCE {
    low            OCTET STRING,
    high           OCTET STRING } }
```

```
CommunityIdentifierList ::= SEQUENCE SIZE (0..MAX) OF Community

Community ::= OBJECT IDENTIFIER

-- TAMP Status Response Message

id-ct-TAMP-statusResponse OBJECT IDENTIFIER ::= { id-tamp 2 }

TAMPStatusResponse ::= SEQUENCE {
  version   [0] TAMPVersion DEFAULT v2,
  query       TAMPMsgRef,
  response  StatusResponse,
  usesApex  BOOLEAN DEFAULT TRUE }

StatusResponse ::= CHOICE {
  terseResponse        [0] TerseStatusResponse,
  verboseResponse      [1] VerboseStatusResponse }

TerseStatusResponse ::= SEQUENCE {
  taKeyIds             KeyIdentifiers,
  communities          CommunityIdentifierList OPTIONAL }

KeyIdentifiers ::= SEQUENCE SIZE (1..MAX) OF KeyIdentifier

VerboseStatusResponse ::= SEQUENCE {
  taInfo                 TrustAnchorChoiceList,
  continPubKeyDecryptAlg [0] AlgorithmIdentifier OPTIONAL,
  communities            [1] CommunityIdentifierList OPTIONAL,
  tampSeqNumbers         [2] TAMPSequenceNumbers OPTIONAL }

TrustAnchorChoiceList ::= SEQUENCE SIZE (1..MAX) OF
     TrustAnchorChoice

TAMPSequenceNumber ::= SEQUENCE {
  keyId       KeyIdentifier,
  seqNumber   SeqNumber }

TAMPSequenceNumbers ::= SEQUENCE SIZE (1..MAX) OF
     TAMPSequenceNumber

-- Trust Anchor Update Message

id-ct-TAMP-update OBJECT IDENTIFIER ::= { id-tamp 3 }
```

```
   TAMPUpdate ::= SEQUENCE {
     version  [0] TAMPVersion DEFAULT v2,
     terse    [1] TerseOrVerbose DEFAULT verbose,
     msgRef    TAMPMsgRef,
     updates   SEQUENCE SIZE (1..MAX) OF TrustAnchorUpdate,
     tampSeqNumbers [2]TAMPSequenceNumbers OPTIONAL }

   TrustAnchorUpdate ::= CHOICE {
     add             [1] TrustAnchorChoice,
     remove          [2] SubjectPublicKeyInfo,
     change          [3] EXPLICIT TrustAnchorChangeInfoChoice }

   TrustAnchorChangeInfoChoice ::= CHOICE {
     tbsCertChange [0] TBSCertificateChangeInfo,
     taChange      [1] TrustAnchorChangeInfo }

   TBSCertificateChangeInfo  ::=  SEQUENCE  {
     serialNumber          CertificateSerialNumber OPTIONAL,
     signature           [0] AlgorithmIdentifier OPTIONAL,
     issuer              [1] Name OPTIONAL,
     validity            [2] Validity OPTIONAL,
     subject             [3] Name OPTIONAL,
     subjectPublicKeyInfo [4] SubjectPublicKeyInfo,
     exts                [5] EXPLICIT Extensions OPTIONAL }

   TrustAnchorChangeInfo ::= SEQUENCE {
     pubKey          SubjectPublicKeyInfo,
     keyId           KeyIdentifier OPTIONAL,
     taTitle         TrustAnchorTitle OPTIONAL,
     certPath        CertPathControls OPTIONAL,
     exts            [1] Extensions OPTIONAL }

   -- Trust Anchor Update Confirm Message

   id-ct-TAMP-updateConfirm OBJECT IDENTIFIER ::= { id-tamp 4 }

   TAMPUpdateConfirm ::= SEQUENCE {
     version             [0] TAMPVersion DEFAULT v2,
     update              TAMPMsgRef,
     confirm             UpdateConfirm }

   UpdateConfirm ::= CHOICE {
     terseConfirm        [0] TerseUpdateConfirm,
     verboseConfirm      [1] VerboseUpdateConfirm }

   TerseUpdateConfirm ::= StatusCodeList

   StatusCodeList ::= SEQUENCE SIZE (1..MAX) OF StatusCode
```

```
   VerboseUpdateConfirm ::= SEQUENCE {
     status          StatusCodeList,
     taInfo          TrustAnchorChoiceList,
     tampSeqNumbers  TAMPSequenceNumbers OPTIONAL,
     usesApex        BOOLEAN DEFAULT TRUE }

   -- Apex Trust Anchor Update Message

   id-ct-TAMP-apexUpdate OBJECT IDENTIFIER ::= { id-tamp 5 }

   TAMPApexUpdate ::= SEQUENCE {
     version            [0] TAMPVersion DEFAULT v2,
     terse              [1] TerseOrVerbose DEFAULT verbose,
     msgRef             TAMPMsgRef,
     clearTrustAnchors  BOOLEAN,
     clearCommunities   BOOLEAN,
     seqNumber          SeqNumber OPTIONAL,
     apexTA             TrustAnchorChoice }

   -- Apex Trust Anchor Update Confirm Message

   id-ct-TAMP-apexUpdateConfirm OBJECT IDENTIFIER ::= { id-tamp 6 }

   TAMPApexUpdateConfirm ::= SEQUENCE {
     version            [0] TAMPVersion DEFAULT v2,
     apexReplace        TAMPMsgRef,
     apexConfirm        ApexUpdateConfirm }

   ApexUpdateConfirm ::= CHOICE {
     terseApexConfirm   [0] TerseApexUpdateConfirm,
     verboseApexConfirm [1] VerboseApexUpdateConfirm }

   TerseApexUpdateConfirm ::= StatusCode

   VerboseApexUpdateConfirm ::= SEQUENCE {
     status             StatusCode,
     taInfo             TrustAnchorChoiceList,
     communities        [0] CommunityIdentifierList OPTIONAL,
     tampSeqNumbers     [1] TAMPSequenceNumbers OPTIONAL }

   -- Community Update Message

   id-ct-TAMP-communityUpdate OBJECT IDENTIFIER ::= { id-tamp 7 }
```

```
   TAMPCommunityUpdate ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     terse            [1] TerseOrVerbose DEFAULT verbose,
     msgRef           TAMPMsgRef,
     updates          CommunityUpdates }

   CommunityUpdates ::= SEQUENCE {
     remove           [1] CommunityIdentifierList OPTIONAL,
     add              [2] CommunityIdentifierList OPTIONAL }
     -- At least one must be present

   -- Community Update Confirm Message

   id-ct-TAMP-communityUpdateConfirm OBJECT IDENTIFIER ::= { id-tamp 8 }

   TAMPCommunityUpdateConfirm ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     update           TAMPMsgRef,
     commConfirm      CommunityConfirm }

   CommunityConfirm ::= CHOICE {
     terseCommConfirm      [0] TerseCommunityConfirm,
     verboseCommConfirm    [1] VerboseCommunityConfirm }

   TerseCommunityConfirm ::= StatusCode

   VerboseCommunityConfirm ::= SEQUENCE {
     status           StatusCode,
     communities      CommunityIdentifierList OPTIONAL }

   -- Sequence Number Adjust Message

   id-ct-TAMP-seqNumAdjust OBJECT IDENTIFIER ::= { id-tamp 10 }

   SequenceNumberAdjust ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     msgRef           TAMPMsgRef }

   -- Sequence Number Adjust Confirm Message

   id-ct-TAMP-seqNumAdjustConfirm OBJECT IDENTIFIER ::= { id-tamp 11 }

   SequenceNumberAdjustConfirm ::= SEQUENCE {
     version          [0] TAMPVersion DEFAULT v2,
     adjust           TAMPMsgRef,
     status           StatusCode }
```

```
   -- TAMP Error Message

   id-ct-TAMP-error OBJECT IDENTIFIER ::= { id-tamp 9 }

   TAMPError ::= SEQUENCE {
     version         [0] TAMPVersion DEFAULT v2,
     msgType             OBJECT IDENTIFIER,
     status              StatusCode,
     msgRef              TAMPMsgRef OPTIONAL }

   -- Status Codes

   StatusCode ::= ENUMERATED {
     success                          (0),
     decodeFailure                    (1),
     badContentInfo                   (2),
     badSignedData                    (3),
     badEncapContent                  (4),
     badCertificate                   (5),
     badSignerInfo                    (6),
     badSignedAttrs                   (7),
     badUnsignedAttrs                 (8),
     missingContent                   (9),
     noTrustAnchor                   (10),
     notAuthorized                   (11),
     badDigestAlgorithm              (12),
     badSignatureAlgorithm           (13),
     unsupportedKeySize              (14),
     unsupportedParameters           (15),
     signatureFailure                (16),
     insufficientMemory              (17),
     unsupportedTAMPMsgType          (18),
     apexTAMPAnchor                  (19),
     improperTAAddition              (20),
     seqNumFailure                   (21),
     contingencyPublicKeyDecrypt     (22),
     incorrectTarget                 (23),
     communityUpdateFailed           (24),
     trustAnchorNotFound             (25),
     unsupportedTAAlgorithm          (26),
     unsupportedTAKeySize            (27),
     unsupportedContinPubKeyDecryptAlg (28),
     missingSignature                (29),
     resourcesBusy                   (30),
     versionNumberMismatch           (31),
     missingPolicySet                (32),
     revokedCertificate              (33),
     unsupportedTrustAnchorFormat    (34),
```

```
      improperTAChange                  (35),
      malformed                         (36),
      cmsError                          (37),
      unsupportedTargetIdentifier       (38),
      other                            (127) }
```

   -- Object Identifier Arc for Attributes

   id-attributes OBJECT IDENTIFIER ::= { joint-iso-ccitt(2) country(16)
       us(840) organization(1) gov(101) dod(2) infosec(1) 5 }

   -- id-aa-TAMP-contingencyPublicKeyDecryptKey uses
   -- PlaintextSymmetricKey syntax
   id-aa-TAMP-contingencyPublicKeyDecryptKey OBJECT IDENTIFIER ::= {
     id-attributes 63 }

   PlaintextSymmetricKey ::= OCTET STRING

   -- id-pe-wrappedApexcontinKey extension

   id-pe-wrappedApexContinKey OBJECT IDENTIFIER ::=
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) pe(1) 20 }

   ApexContingencyKey ::= SEQUENCE {
     wrapAlgorithm   AlgorithmIdentifier,
     wrappedContinPubKey   OCTET STRING }

   END

Appendix B. Media Type Registrations

   Eleven media type registrations are provided in this appendix, one
   for each content type defined in this specification. As noted in
   Section 2, in all cases TAMP messages are encapsulated within
   ContentInfo structures. Signed messages are additionally
   encapsulated within a SignedData structure.

B.1. application/tamp-status-query

   Media type name: application

   Subtype name: tamp-status-query

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries a signed request for status
   information. Integrity protection is discussed in Section 4.1.
   Replay detection is discussed in Section 6.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests for status information.

   Additional information:

      Magic number(s): None

      File extension(s): .tsq

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.2. application/tamp-status-response

   Media type name: application

   Subtype name: tamp-status-response

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries optionally signed status
   information. Integrity protection is discussed in Section 4.2.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests for status information.

   Additional information:

      Magic number(s): None

      File extension(s): .tsr

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.3. application/tamp-update

   Media type name: application

   Subtype name: tamp-update

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries a signed trust anchor update
   message. Integrity protection is discussed in Section 4.3. Replay
   detection is discussed in Section 6.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update trust anchor information.

   Additional information:

      Magic number(s): None

      File extension(s): .tur

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.4. application/tamp-update-confirm

   Media type name: application

   Subtype name: tamp-update-confirm

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries an optionally signed TAMP update
   response. Integrity protection is discussed in Section 4.4.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update trust anchor information.

   Additional information:

      Magic number(s): None

      File extension(s): .tuc

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.5. application/tamp-apex-update

   Media type name: application

   Subtype name: tamp-apex-update

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries a signed request to update an apex
   trust anchor information. Integrity protection is discussed in
   Section 4.5. Replay detection is discussed in Section 6.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update an apex trust anchor.

Additional information:

    Magic number(s): None

    File extension(s): .tau

    Macintosh File Type Code(s):

Person & email address to contact for further information:

Sam Ashmore - srashmo@radium.ncsc.mil

Intended usage: LIMITED USE

Restrictions on usage: None

Author: Sam Ashmore - srashmo@radium.ncsc.mil

Change controller: IESG

B.6. application/tamp-apex-update-confirm

Media type name: application

Subtype name: tamp-apex-update-confirm

Required parameters: None

Optional parameters: None

Encoding considerations: binary

Security considerations: Carries an optionally signed response to an
apex update request. Integrity protection is discussed in
Section 4.6.

Interoperability considerations: None

Published specification: RFC 5934

Applications that use this media type: TAMP clients responding to
requests to update an apex trust anchor.

   Additional information:

      Magic number(s): None

      File extension(s): .auc

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.7. application/tamp-community-update

   Media type name: application

   Subtype name: tamp-community-update

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries a signed request to update community
   membership information. Integrity protection is discussed in
   Section 4.7. Replay detection is discussed in Section 6.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update community membership.

   Additional information:

      Magic number(s): None

      File extension(s): .tcu

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.8. application/tamp-community-update-confirm

   Media type name: application

   Subtype name: tamp-community-update-confirm

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries an optionally signed response to a
   community update request. Integrity protection is discussed in
   Section 4.8.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update community membership.

   Additional information:

      Magic number(s): None

      File extension(s): .cuc

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

B.9. application/tamp-sequence-adjust

   Media type name: application

   Subtype name: tamp-sequence-adjust

   Required parameters: None

   Optional parameters: None

   Encoding considerations: binary

   Security considerations: Carries a signed request to update sequence
   number information. Integrity protection is discussed in
   Section 4.9. Replay detection is discussed in Section 6.

   Interoperability considerations: None

   Published specification: RFC 5934

   Applications that use this media type: TAMP clients responding to
   requests to update sequence number information.

Additional information:

    Magic number(s): None

    File extension(s): .tsa

    Macintosh File Type Code(s):

Person & email address to contact for further information:

Sam Ashmore - srashmo@radium.ncsc.mil

Intended usage: LIMITED USE

Restrictions on usage: None

Author: Sam Ashmore - srashmo@radium.ncsc.mil

Change controller: IESG

B.10. application/tamp-sequence-adjust-confirm

Media type name: application

Subtype name: tamp-sequence-adjust-confirm

Required parameters: None

Optional parameters: None

Encoding considerations: binary

Security considerations: Carries an optionally signed sequence number
adjust confirmation message. Integrity protection is discussed in
Section 4.10.

Interoperability considerations: None

Published specification: RFC 5934

Applications that use this media type: TAMP clients responding to
requests to update sequence number information.

Additional information:

    Magic number(s): None

    File extension(s): .sac

    Macintosh File Type Code(s):

Person & email address to contact for further information:

Sam Ashmore - srashmo@radium.ncsc.mil

Intended usage: LIMITED USE

Restrictions on usage: None

Author: Sam Ashmore - srashmo@radium.ncsc.mil

Change controller: IESG

B.11. application/tamp-error

Media type name: application

Subtype name: tamp-error

Required parameters: None

Optional parameters: None

Encoding considerations: binary

Security considerations: Carries optionally signed error information
collecting during TAMP processing. Integrity protection is discussed
in Section 4.11.

Interoperability considerations: None

Published specification: RFC 5934

Applications that use this media type: TAMP clients processing TAMP
messages.

   Additional information:

      Magic number(s): None

      File extension(s): .ter

      Macintosh File Type Code(s):

   Person & email address to contact for further information:

   Sam Ashmore - srashmo@radium.ncsc.mil

   Intended usage: LIMITED USE

   Restrictions on usage: None

   Author: Sam Ashmore - srashmo@radium.ncsc.mil

   Change controller: IESG

Appendix C. TAMP over HTTP

   This appendix describes the formatting and transportation conventions
   for the TAMP messages when carried by HTTP [RFC2616]. Each TAMP
   message type is covered by a subsection below. Each TAMP request
   message sent via HTTP is responded to either with an HTTP response
   containing a TAMP response or error or, if failure occurs prior to
   invoking TAMP, an HTTP error. TAMP response, confirmation, and error
   messages are not suitable for caching. In order for TAMP clients and
   servers using HTTP to interoperate, the following rules apply.

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

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

   o Clients MAY attempt to send HTTPS requests using Transport Layer
      Security (TLS) 1.0 or later, although servers are not required to
      support TLS.

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

   o Clients and servers are expected to follow the other rules and
      restrictions in [RFC2616]. Note that some of those rules are for
      HTTP methods other than POST; clearly, only the rules that apply
      to POST are relevant for this specification.

C.1. TAMP Status Query Message

   A TAMP Status Query Message using the POST method is constructed as
   follows: The Content-Type header MUST have the value "application/
   tamp-status-query".

   The body of the message is the binary value of the DER encoding of
   the TAMPStatusQuery, wrapped in a CMS body as described in Section 2.

C.2. TAMP Status Response Message

   An HTTP-based TAMP Status Response message is composed of the
   appropriate HTTP headers, followed by the binary value of the DER
   encoding of the TAMPStatusResponse, wrapped in a CMS body as
   described in Section 2.

   The Content-Type header MUST have the value "application/
   tamp-status-response."

C.3. Trust Anchor Update Message

   A Trust Anchor Update Message using the POST method is constructed as
   follows: The Content-Type header MUST have the value "application/
   tamp-update".

   The body of the message is the binary value of the DER encoding of
   the TAMPUpdate, wrapped in a CMS body as described in Section 2.

C.4. Trust Anchor Update Confirm Message

   An HTTP-based Trust Anchor Update Confirm message is composed of the
   appropriate HTTP headers, followed by the binary value of the DER
   encoding of the TAMPUpdateConfirm, wrapped in a CMS body as described
   in Section 2.

   The Content-Type header MUST have the value "application/
   tamp-update-confirm".

C.5. Apex Trust Anchor Update Message

   An Apex Trust Anchor Update Message using the POST method is
   constructed as follows: The Content-Type header MUST have the value
   "application/tamp-apex-update".

   The body of the message is the binary value of the DER encoding of
   the TAMPApexUpdate, wrapped in a CMS body as described in Section 2.

C.6. Apex Trust Anchor Update Confirm Message

   An HTTP-based Apex Trust Anchor Update Confirm message is composed of
   the appropriate HTTP headers, followed by the binary value of the DER
   encoding of the TAMPApexUpdateConfirm, wrapped in a CMS body as
   described in Section 2.

   The Content-Type header MUST have the value "application/
   tamp-apex-update-confirm".

C.7. Community Update Message

   A Community Update Message using the POST method is constructed as
   follows: The Content-Type header MUST have the value "application/
   tamp-community-update".

   The body of the message is the binary value of the DER encoding of
   the TAMPCommunityUpdate, wrapped in a CMS body as described in
   Section 2.

C.8. Community Update Confirm Message

   An HTTP-based Community Update Confirm message is composed of the
   appropriate HTTP headers, followed by the binary value of the DER
   encoding of the TAMPCommunityUpdateConfirm, wrapped in a CMS body as
   described in Section 2.

   The Content-Type header MUST have the value "application/
   tamp-community-update-confirm".

C.9. Sequence Number Adjust Message

   A Sequence Number Adjust Message using the POST method is constructed
   as follows: The Content-Type header MUST have the value "application/
   tamp-sequence-adjust".

   The body of the message is the binary value of the DER encoding of
   the SequenceNumberAdjust, wrapped in a CMS body as described in
   Section 2.

C.10. Sequence Number Adjust Confirm Message

   An HTTP-based Sequence Number Adjust Confirm message is composed of
   the appropriate HTTP headers, followed by the binary value of the DER
   encoding of the SequenceNumberAdjustConfirm, wrapped in a CMS body as
   described in Section 2.

   The Content-Type header MUST have the value "application/
   tamp-sequence-adjust-confirm".

C.11. TAMP Error Message

   An HTTP-based TAMP Error message is composed of the appropriate HTTP
   headers, followed by the binary value of the DER encoding of the
   TAMPError, wrapped in a CMS body as described in Section 2.

   The Content-Type header MUST have the value "application/tamp-error".

Authors' Addresses

   Russ Housley
   Vigil Security, LLC
   918 Spring Knoll Drive
   Herndon, VA 20170
   USA

   EMail: housley@vigilsec.com

   Sam Ashmore
   National Security Agency
   Suite 6751
   9800 Savage Road
   Fort Meade, MD 20755
   USA

   EMail: srashmo@radium.ncsc.mil

   Carl Wallace
   Cygnacom Solutions
   Suite 5400
   7925 Jones Branch Drive
   McLean, VA 22102
   USA

   EMail: cwallace@cygnacom.com