Network Working Group	R. Housley
Internet-Draft	Vigil Security, LLC
Intended status: Standards Track	R. Reddy
Expires: April 6, 2008	National Security Agency
	C. Wallace
	Cygnacom Solutions
	October 04, 2007

TOC

Trust Anchor Management Protocol (TAMP)
draft-housley-tamp-00

Status of this Memo

Abstract

This document describes a transport independent, request-response protocol for the management of trust anchors and community identifiers stored in a device. The protocol makes use of the Cryptographic Message Syntax (CMS), and a digital signature is used to provide integrity protection and data origin authentication. Each trust anchor is associated with a list of functions within devices that make use of digital signature mechanisms. Digital signatures can be validated directly with the public key associated with the trust anchor, or they can be validated with a certified public key whose X.509 certification path terminates with the trust anchor public key.

Table of Contents

1. Introduction

TOC

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, where the trust anchor public key can be used to validate digital signatures on firmware packages or validate the X.509 certification path [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.)[X.509] (, “ITU-T Recommendation X.509 - The Directory - Authentication Framework,” 2000.) of the firmware package signer [RFC4108] (Housley, R., “Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages,” August 2005.).

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) [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.). The CMS is a data protection encapsulation syntax that makes use of ASN.1 [X.680] (, “ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One,” 1997.).

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.

1.1. Terminology

TOC

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] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

1.2. Trust Anchors

TOC

TAMP manages trust anchors, but TAMP does not dictate a particular structure for the storage of trust anchor information in cryptographic

modules. A trust anchor contains a public key that is used to validate digital signatures.
There are three types of trust anchors: apex trust anchors, management trust anchors, and identity trust anchors.
All trust anchors, regardless of their type, are named by the public key, and all trust anchors consist of the following components:

- A public key signature algorithm identifier and associated public key, which MAY include parameters
- A public key identifier
- An OPTIONAL human readable trust anchor title
- OPTIONAL X.509 certification path controls

The apex trust anchor and management trust anchors that issue TAMP messages also include a sequence number 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 the signer. 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. The trust anchor X.500 distinguished name within the OPTIONAL X.509 certification path controls is used when the trust anchor public key is used to validate an X.509 certification path. In this case, the certificate subject is the signer. Use of an X.509 certification path represents delegation, and delegation is possible only when the trust anchor configuration includes an X.500 distinguished name.
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 can be used for things other than digital signature validation; the other possible actions are constrained by the key usage certificate extension. Cryptographic modules MUST support validation of X.509 certificates that are directly signed by a trust anchor; however, support for longer certification paths is RECOMMENDED. 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.

TOC

1.2.1. Apex Trust Anchors

Within the context of a single cryptographic module, 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 cryptographic module. The ultimate authority could be the legal owner of the device in a commercial setting. 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 cryptographic module. The apex trust anchor is by definition unconstrained and therefore does not have explicit authorization information associated with it. In order to make processing of messages as uniform as possible, the apex has an implicit OID associated with it that represents the special anyContentType value. This OID will be used as input to processing algorithms to represent the apex trust anchor authorization.

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:

- The crypto period for the apex trust anchor public/private key pair has come to an end
- The apex trust anchor private key is no longer available
- The apex trust anchor public/private key pair needs to be revoked
- 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
- The authority has decided to use a different key size
- The authority has decided to transfer control to another authority

To accommodate these requirements, the apex trust anchor has a different structure than other trust anchors; it includes two public keys. Whenever the apex trust anchor is updated, both public keys are 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 which 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

TOC

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] (Housley, R., “Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages,” August 2005.) is validated to a management trust anchor.
Authorization checking is needed for management messages, and these checks are based on the content type of the management message. As a result, management trust anchors include a list of object identifiers (OIDs) that name authorized content types along with OPTIONAL constraints.

1.2.3. Identity Trust Anchors

TOC

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

TOC

TAMP does not assume any particular architecture; however, for TAMP to be useful in an architecture, it MUST include a cryptographic module, 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.

1.3.1. Cryptographic Module

TOC

The cryptographic module MUST include the following capabilities:

- Each cryptographic module within a family of cryptographic modules (which are generally produced by the same manufacturer) MUST have a unique serial number (with respect to other modules within the same family). The cryptographic module family is represented as an ASN.1 object identifier (OID), and the unique serial number is represented as a string of octets.

- Each cryptographic module MUST 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 firmware package.

- The cryptographic module MUST support the secure storage of exactly one apex trust anchor. The cryptographic module SHOULD support the secure storage of at least one additional trust anchor.

- The cryptographic module MUST 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.

- The cryptographic module MUST support at least one one-way hash function, one digital signature validation algorithm, one digital signature generation algorithm, and 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 [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.), and for validation of X.509 certification paths.

1.3.2. TAMP Protocol Processing Dependencies

TOC

TAMP processing MUST include the following capabilities:

- 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 can also be used.

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

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

- 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 access controls MUST be securely stored with each management trust anchor as described in Section 1.3.3.

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

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

- 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 [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.).

*The TAMP processing MUST have read access to the cryptographic module family identifier, serial number, and community membership list.

1.3.3. Application-Specific Protocol Processing

TOC

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

*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.

*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.

*The application-specific protocol processing MUST have read access to the content types and any associated constraints held in storage with management trust anchors to make authorization decisions for that application. The authorization decisions apply to the management trust anchor as well as any public key that is validated to the management trust anchor via an X.509 certification path.

*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.

*The application-specific protocol processing MUST have read access to the cryptographic module family identifier, serial number, and community membership list.

It is expected that application-specific protocol processing will also include constraints processing. In some applications, management trust anchors could be authorized for a subset of the functionality associated with a particular content type.

1.4. ASN.1 Encoding

TOC

The CMS uses Abstract Syntax Notation One (ASN.1) [X.680] (, "ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One," 1997.). 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] (, "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.) 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] (, "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.) 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 the 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

TOC

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, and it MUST encapsulate the CMS

SignedData content type. The CMS SignedData content type MUST encapsulate the TAMP message. A unique content type identifier identifies the particular 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     DigestAlgorithmIdentifier,
  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. Content Info

TOC

CMS requires the outer-most encapsulation to be ContentInfo [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.). The fields of ContentInfo are used as follows:

- 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.

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

2.2. SignedData Info

TOC

The SignedData content type [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) 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:

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

- 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.

*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.

*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 cryptographic module 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 [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.), 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 cryptographic module, this field SHOULD include only the cryptographic module certificate or 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] (, "PKCS #6: Extended-Certificate Syntax Standard, Version 1.5," November 1993.) and attribute certificates (either version 1 or version 2) [RFC3281] (Farrell, S. and R. Housley, "An Internet Attribute Certificate Profile for Authorization," April 2002.) 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 [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.).

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

*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:

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

*sid identifies the TAMP message originator's public key. The subjectKeyIdentifier alternative is always used with TAMP, which identifies the public key directly. When an apex trust anchor operational public key or a management trust anchor public key is used directly, this identifier is the keyId from the associated TrustAnchorInfo. When the public key is included in an X.509 certificate, this identifier is included in the subjectKeyIdentifier certificate extension.

*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.

*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] (, "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.). 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 the cryptographic module MUST ignore any unrecognized signed attributes.

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

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

*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

TOC

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

- 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.

- 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

TOC

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 [X.501] (, “ITU-T Recommendation X.501 - The Directory - Models,” 1993.). X.500 Directory provides a rich attribute syntax. A very simple subset of this syntax is used extensively in [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.), where ATTRIBUTE.&Type and ATTRIBUTE.&id are the only parts of the ATTRIBUTE class that are employed.
The attribute syntax is repeated here for convenience:

```
Attribute ::= SEQUENCE {
    type          ATTRIBUTE.&id ({SupportedAttributes}),
    values        SET SIZE (1..MAX) OF ATTRIBUTE.&Type
                       ({SupportedAttributes}{@type}) }

SupportedAttributes ATTRIBUTE ::= { ... }

ATTRIBUTE ::= CLASS {
    &derivation               ATTRIBUTE OPTIONAL,
    &Type                     OPTIONAL,
                         -- either &Type or &derivation REQUIRED
    &equality-match           MATCHING-RULE OPTIONAL,
    &ordering-match           MATCHING-RULE OPTIONAL,
    &substrings-match         MATCHING-RULE OPTIONAL,
    &single-valued            BOOLEAN DEFAULT FALSE,
    &collective               BOOLEAN DEFAULT FALSE,
    -- operational extensions
    &no-user-modification     BOOLEAN DEFAULT FALSE,
    &usage                    AttributeUsage DEFAULT
                                        userApplications,
    &id                       OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
    [ SUBTYPE OF              &derivation ]
    [ WITH SYNTAX             &Type ]
    [ EQUALITY MATCHING RULE  &equality-match ]
    [ ORDERING MATCHING RULE  &ordering-match ]
    [ SUBSTRINGS MATCHING RULE &substrings-match ]
    [ SINGLE VALUE            &single-valued ]
    [ COLLECTIVE              &collective ]
    [ NO USER MODIFICATION    &no-user-modification ]
    [ USAGE                   &usage ]
    ID                        &id }

MATCHING-RULE ::= CLASS {
    &AssertionType            OPTIONAL,
    &id                       OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
    [ SYNTAX                  &AssertionType ]
    ID                        &id }

AttributeType ::= ATTRIBUTE.&id

AttributeValue ::= ATTRIBUTE.&Type

AttributeUsage ::= ENUMERATED {
  userApplications          (0),
  directoryOperation        (1),
  distributedOperation      (2),
```

```
        dSAOperation              (3) }
```

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 signed 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:

- *content-type MUST be supported.
- *message-digest MUST be supported.
- *content-hints MAY be supported. Only present when more than one layer of encapsulation is employed.
- *binary-signing-time MAY be supported. Generally ignored by the recipient.
- *other attributes MAY be supported. Unrecognized attributes MUST be ignored by the recipient.

2.2.3.1. Content-Type Attribute

TOC

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 [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) 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

TOC

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 [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) defines the message-digest attribute.

2.2.3.3. Content-Hints Attribute

TOC

Many applications find it useful to have information that describes the innermost content when multiple layers of encapsulation have been applied. Since this version of TAMP only has one layer of encapsulation, the encapContentInfo provides the content type of the innermost content. To accommodate future versions of TAMP that might include additional layers of encapsulation, the content-hints attribute MUST be included in every instance of SignedData that does not directly encapsulate a TAMP message. Section 2.9 of [RFC2634] (Hoffman, P., “Enhanced Security Services for S/MIME,” June 1999.) defines the content-hints attribute.
The content-hints attribute contains two fields: contentDescription and contentType. The contentType field MUST be present, and the contentDescription field MAY be present. The fields of the content-hints attribute are used as follows:

*contentDescription is OPTIONAL. The TAMP message signer MAY provide a brief description of the purpose of the TAMP message. The text is intended for human consumption, not machine processing. The text is encoded in UTF-8 [RFC3629] (Yergeau, F., “UTF-8, a transformation format of ISO 10646,” November 2003.), which accommodates most of the world’s writing systems. The implementation MUST provide the capability to constrain the character set.

*contentType is mandatory. This field indicates the content type that will be discovered when CMS protection content types are removed.

TOC

2.2.3.4. 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] (Housley, R., “BinaryTime: An Alternate Format for Representing Date and Time in ASN.1,” April 2005.).
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

TOC

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 the cryptographic module MUST ignore unrecognized unsigned attributes.
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

TOC

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 Information Syntax

TOC

An implementation MAY store trust anchor information in any format; however, a common syntax is used throughout the TAMP specification for trust anchor information. This section describes the TrustAnchorInfo ASN.1 type.

```
TrustAnchorInfo ::= SEQUENCE {
   version   [0] TAMPVersion DEFAULT v2,
   pubKey    PublicKeyInfo,
   keyId     KeyIdentifier,
   taType    TrustAnchorType,
   taTitle   TrustAnchorTitle OPTIONAL,
   certPath  CertPathControls OPTIONAL }

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

PublicKeyInfo ::= SEQUENCE {
   algorithm  AlgorithmIdentifier,
   publicKey  BIT STRING }

 KeyIdentifier ::= OCTET STRING

 TrustAnchorType ::= CHOICE {
   apex   [0] ApexTrustAnchorInfo,
   mgmt   [1] MgmtTrustAnchorInfo,
   ident  [2] NULL }

 ApexTrustAnchorInfo ::= SEQUENCE {
   contInPubKey  ApexContingencyKey,
   seqNum        SeqNumber OPTIONAL }

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

 SeqNumber ::= INTEGER (0..9223372036854775807)

 MgmtTrustAnchorInfo ::= SEQUENCE {
   taUsage  TrustAnchorUsage,
   seqNum   SeqNumber OPTIONAL }

 TrustAnchorUsage ::= CMSContentConstraints

 CMSContentConstraints ::= ContentTypeConstraintList

 ContentTypeConstraintList ::= SEQUENCE SIZE (1..MAX) OF
                           ContentTypeConstraint

 ContentTypeConstraint ::= SEQUENCE {
   contentType      ContentType,
   canSource        BOOLEAN DEFAULT TRUE,
   attrConstraints  AttrConstraintList OPTIONAL }

 ContentType ::= OBJECT IDENTIFIER
```

```
AttrConstraintList ::= SEQUENCE SIZE (1..MAX) OF AttrConstraint

AttrConstraint ::= SEQUENCE {
  attrType    AttributeType,
  attrValues  SET SIZE (1..MAX) OF AttributeValue }


TrustAnchorTitle ::= UTF8String (SIZE (1..64))

CertPathControls ::= SEQUENCE {
  taName           Name,
  selfSigned       [0] Certificate OPTIONAL,
  policySet        [1] CertificatePolicies OPTIONAL,
  policyFlags      [2] CertPolicyFlags OPTIONAL,
  clearanceConstr  [3] CAClearanceConstraints OPTIONAL,
  nameConstr       [4] NameConstraints OPTIONAL }

CertificatePolicies ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

PolicyInformation ::= SEQUENCE {
  policyIdentifier  CertPolicyId,
  policyQualifiers  SEQUENCE SIZE (1..MAX) OF
                          PolicyQualifierInfo OPTIONAL }

CertPolicyId ::= OBJECT IDENTIFIER

CertPolicyFlags ::= BIT STRING {
  inhibitPolicyMapping   (0),
  requireExplicitPolicy  (1),
  inhibitAnyPolicy       (2) }

CAClearanceConstraints ::= SEQUENCE SIZE (1..MAX) OF Clearance

Clearance ::= SEQUENCE {
  policyId           [0] OBJECT IDENTIFIER,
  classList          [1] ClassList DEFAULT {unclassified},
  securityCategories [2] SET OF SecurityCategory OPTIONAL }

ClassList ::= BIT STRING {
  unmarked      (0),
  unclassified  (1),
  restricted    (2),
  confidential  (3),
  secret        (4),
  topSecret     (5) }

SecurityCategory ::= SEQUENCE {
  type   [0] SECURITY-CATEGORY.&id({SecurityCategoriesTable}),
  value  [1] EXPLICIT SECURITY-CATEGORY.&Type
```

```
                    ({SecurityCategoriesTable}{@type}) }

   SECURITY-CATEGORY ::= TYPE-IDENTIFIER

   SecurityCategoriesTable SECURITY-CATEGORY ::= {...}

   NameConstraints ::= SEQUENCE {
     permittedSubtrees  [0] GeneralSubtrees OPTIONAL,
     excludedSubtrees   [1] GeneralSubtrees OPTIONAL }

   GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

   GeneralSubtree ::= SEQUENCE {
     base     GeneralName,
     minimum  [0] BaseDistance DEFAULT 0,
     maximum  [1] BaseDistance OPTIONAL }

  BaseDistance ::= INTEGER (0..MAX)
```

The fields of TrustAnchorInfo are used as follows:

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

*pubKey identifies the public key and algorithm associated with the trust anchor using the PublicKeyInfo structure. The PublicKeyInfo structure contains the algorithm identifier followed by the public key itself. The algorithm identifier is an AlgorithmIdentifier, which contains an object identifier and OPTIONAL parameters. The object identifier names the digital signature algorithm, and it indicates the syntax of the parameters, if present, as well as the format of the public key. The public key is encoded as a BIT STRING. For the apex trust anchor, this field contains the operational public key.

*keyId contains the public key identifier of the trust anchor public key. For the apex trust anchor, this field contains the public key identifier of the operational public key.

*taType indicates the type of trust anchor, and it carries information specific to the type of trust anchor that is being represented. If an apex trust anchor is represented, then apex trust anchor information is carried using the ApexTrustAnchorInfo structure. If a management trust anchor is represented, then management trust anchor information is carried using the MgmtTrustAnchorInfo. If an identity trust anchor is represented, no additional information is carried, which is represented by NULL.

*taTitle is OPTIONAL. When it is present, it provides a human readable name for the trust anchor. The text is encoded in UTF-8 [RFC3629] (Yergeau, F., “UTF-8, a transformation format of ISO 10646,” November 2003.), which accommodates most of the world’s writing systems. The implementation MUST provide the capability to constrain the character set.

*certPath is OPTIONAL. When it is present, it provides the controls needed to initialize an X.509 certification path validation algorithm implementation (see Section 6 in [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.)). When absent, the trust anchor cannot be used to validate the signature on an X.509 certificate. For the apex trust anchor, this field contains the certification path controls associated with the operational public key.

The fields of ApexTrustAnchorInfo are used as follows:

*contInPubKey contains the encrypted apex trust anchor contingency public key using the ApexContingencyKey structure.

*seqNum is OPTIONAL. When it is present, it contains the current sequence number value stored by the cryptographic module for the apex trust anchor operational public key. When seqNum is absent, the cryptographic module is prepared to accept any sequence number value for the apex trust anchor operational public key. Section 6 provides sequence number processing details.

The fields of ApexContingencyKey are used as follows:

*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 TAMP 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.

*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 fields of MgmtTrustAnchorInfo are used as follows:

* taUsage represents the authorized uses of the management trust anchor using the TrustAnchorUsage structure.

* seqNum is OPTIONAL. When it is present, it contains the current sequence number value stored by the cryptographic module for this management trust anchor. When seqNum is absent, the cryptographic module is prepared to accept any sequence number value for this management trust anchor. Section 6 provides sequence number processing details.

The TrustAnchorUsage is defined using the CMSContentConstraints type defined in [CCC] (Housley, R. and C. Wallace, "Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension," in progress.). The CMSContentConstraints is a list of permitted content types and associated constraints. The management trust anchor can be used to validate digital signatures on the permitted content types, including TAMP message content types.
The anyContentType object identifier can be used to indicate that the trust anchor is unconstrained. The apex trust anchor has an implicit CMSContentConstraints field with a single permitted content type of anyContentType.
The fields of ContentTypeConstraint are used as follows:

* contentType indicates the encapsulated content type identifier that can be validated using the management trust anchor. For example, it contains id-ct-firmwarePackage when the management trust anchor can be used to validate digital signatures on firmware packages [RFC4108] (Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages," August 2005.). A particular content type MUST NOT appear more than once in the list. The CMS-related content types need not be included in the list of permitted content types. These content types are always authorized to facilitate the use of CMS in the protection of content, and they MUST NOT appear in the permitted list. The always authorized content types are:

  - id-signedData,

  - id-envelopedData,

  - id-digestedData,

  - id-encryptedData,

  - id-ct-authEnvData,

  - id-ct-authData,

  - id-ct-compressedData,

-id-ct-contentCollection

-id-ct-contentWithAttrs.

*canSource is a Boolean flag, and it applies to direct signatures or direct authentication for the specified content type. If the canSource flag is FALSE, then the management trust anchor cannot be used to directly sign or authenticate the specified content type. Regardless of the flag value, a management trust anchor can be used to sign or authenticate outer layers when multiple layers of CMS protected content type are present.

*attrConstraints is an OPTIONAL field that contains a sequence of content type specific constraints. If the attrConstraints field is absent, then the management trust anchor can be used to verify the specified content type without any further checking. If the attrConstraints field is present, then the management trust anchor can only be used to verify the specified content type if all of the constraints for that content type are satisfied. Content type constraints are checked by matching the attribute values in the AttrConstraintList against the attribute value in the content. The constraints checking fails if the attribute is present and the attribute value is not one of the values provided in AttrConstraintList.

The AttrConstraintList contains a sequence of attributes, which is defined in [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.) and repeated above. The fields of AttrConstraint are used as follows:

*attrType is the object identifier of the signed attribute carried in the SignerInfo of the content. For a signed content to satisfy the constraint, if the SignerInfo includes a signed attribute of the same type, then the signed attribute MUST contain one of the values supplied in the attrValues field.

*attrValues provides one or more acceptable signed attribute values. It is a set of AttributeValue. For a signed content to satisfy the constraint, if the SignerInfo includes a signed attribute of the type identified in the attrType field, then the signed attribute MUST contain one of the values in the set.

The fields of CertPathControls are used as follows:

*taName provides the X.500 distinguished name associated with the trust anchor, and this distinguished name is used to construct and validate an X.509 certification path. The name MUST NOT be an empty sequence. An identity trust anchor is of little use without a distinguished name.

*selfSigned provides an OPTIONAL self-signed X.509 certificate, which can be used in some environments to represent the trust anchor in certification path development and validation. If the self-signed certificate is present, the subject name in the certificate MUST exactly match the X.500 distinguished name provided in the taName field. The complete description of the syntax and semantics of the Certificate are provided in [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.).

*policySet is OPTIONAL. When present, it contains sequence of certificate policy identifiers to be provided as inputs to the certification path validation algorithm. When absent, the special value any-policy is provided as the input to the certification path validation algorithm. The complete description of the syntax and semantics of the CertificatePolicies are provided in [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.), including the syntax for PolicyInformation. In this context, the OPTIONAL policyQualifiers structure MUST NOT be included.

*policyFlags is OPTIONAL. When present, three Boolean values for input to the certification path validation algorithm are provided in a BIT STRING. When absent, the input to the certification path validation algorithm is { FALSE, FALSE, FALSE }, which represents the most liberal setting for these flags. The three bits are used as follows:

  -inhibitPolicyMapping indicates if policy mapping is allowed in the certification path. When set to TRUE, policy mapping is not permitted. This value represents the initial-policy-mapping-inhibit input value to the certification path validation algorithm described in section 6.1.1 of [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.).

  -requireExplicitPolicy indicates if the certification path MUST be valid for at least one of the certificate policies in the policySet. When set to TRUE, all certificates in the certification path MUST contain an acceptable policy identifier in the certificate policies extension. This value represents the initial-explicit-policy input value to the certification path validation algorithm described in section 6.1.1 of [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.).

An acceptable policy identifier is a member of the policySet or the identifier of a policy that is declared to be equivalent through policy mapping. This bit MUST be set to FALSE if policySet is absent.

-inhibitAnyPolicy indicates whether the special anyPolicy policy identifier, with the value { 2 5 29 32 0 }, is considered an explicit match for other certificate policies. When set to TRUE, the special anyPolicy policy identifier is only considered a match for itself. This value represents the initial-any-policy-inhibit input value to the certification path validation algorithm described in section 6.1.1 of [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.).

*clearanceConstr is OPTIONAL. It has the same syntax and semantics as the CA Clearance Constraints certificate extension as specified in [ClearConstr] (Turner, S., "Clearance and CA Clearance Constraints Certificate Extensions," in progress.). When it is present, constraints are provided on the CA Clearance Constraints certificate extension and Clearance certificate extension that might appear in subordinate X.509 certificates. For a subordinate certificate to be valid, it MUST conform to these constraints. When it is absent, no constraints are imposed on the CA Clearance Constraints certificate extension and Clearance certificate extension that might appear in subordinate X.509 certificates.

*nameConstr is OPTIONAL. It has the same syntax and semantics as the Name Constraints certificate extension [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.), which includes a list of permitted names and a list of excluded names. The definition of GeneralName can be found in [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.). When it is present, constraints are provided on names (including alternative names) that might appear in subordinate X.509 certificates. When applied to CA certificates, the CA can apply further constraints by including the Name Constraints certificate extension in subordinate certificates. For a subordinate certificate to be valid, it MUST conform to these constraints. When it is absent, no constraints are imposed on names that appear in subordinate X.509 certificates.

When the trust anchor is used to validate a certification path, CertPathControls provides limitations on certification paths that will

successfully validate. An application that is validating a certification path MUST NOT ignore these limitations, but the application can impose additional limitations to ensure that the validated certification path is appropriate for the intended application context. As input to the certification path validation algorithm, an application MAY:

- Provide a subset of the certification policies provided in the policySet;
- Provide a TRUE value for any of the flags in the policyFlags;
- Provide a subset of clearance values provided in the clearanceConstr;
- Provide a subset of the permitted names provided in the nameConstr;
- Provide additional excluded names to the ones that are provided in the nameConstr

---

4. Trust Anchor Management Protocol Messages

TOC

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.

- The TAMP Status Query message MUST be signed. It uses the following object identifier: { id-tamp 1 }.
- The TAMP Status Response message SHOULD be signed. It uses the following object identifier: { id-tamp 2 }.
- The Trust Anchor Update message MUST be signed. It uses the following object identifier: { id-tamp 3 }.
- The Trust Anchor Update Confirm message SHOULD be signed. It uses the following object identifier: { id-tamp 4 }.
- The Apex Trust Anchor Update message MUST be signed. It uses the following object identifier: { id-tamp 5 }.

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

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

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

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

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

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

A typical interaction between a trust anchor manager and a cryptographic module will follow the message flow shown in Figure 4-1. Figure 4-1 does not illustrate a flow where an error occurs.

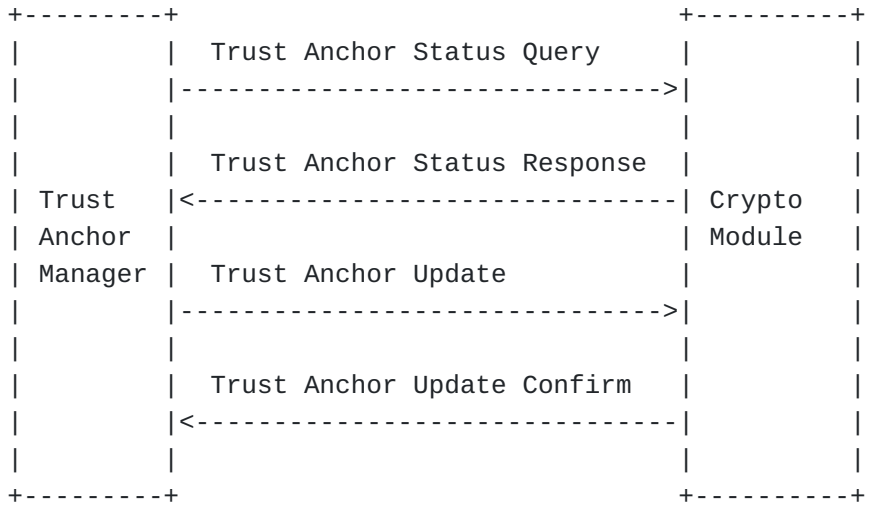

Figure 4-1: Typical TAMP Message Flow

Each TAMP query and update message include an indication of the type of response that is desired. The response can either be terse or verbose. All cryptographic modules MUST support both the terse and verbose responses.
Cryptographic modules MUST 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. Cryptographic modules MAY also process and act upon the valid payload of the Community Update message.

Cryptographic modules MUST 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 cryptographic module supports the Community Update message, then the cryptographic module MUST also support generation of the Community Update Confirm message.

4.1. TAMP Status Query

TOC

The TAMP Status Query message is used to request information about the trust anchors that are currently installed in a cryptographic module, and for the list of communities to which the cryptographic module belongs. The TAMP Status Query message MUST be signed. For the query message to be valid, the cryptographic module MUST be an intended recipient of the query, the sequence number checking described in Section 6 MUST be successful when the TAMP message source is a trust anchor, and the digital signature MUST be validated by the apex trust anchor operational public key, a management trust anchor authorized for the id-ct-TAMP-statusQuery content type, or via a valid 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 for the id-ct-TAMP-statusQuery content type, and the cryptographic module is an intended recipient of the TAMP message, then a TAMP Status Response message MUST be returned. If a TAMP Status Response message is not returned, then a TAMP Error message MUST be returned.
The TAMP Status Query content type has the following syntax:

```
PKCS7-CONTENT-TYPE ::= TYPE-IDENTIFIER

tamp-status-query PKCS7-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 }

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

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

TargetIdentifier ::= CHOICE {
  hwModules    [1] HardwareModuleIdentifierList,
  communities  [2] CommunityIdentifierList,
  allModules   [3] NULL }

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 (1..MAX) OF Community

Community ::= OBJECT IDENTIFIER
```

The fields of TAMPStatusQuery are used as follows:

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

*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.

*query contains two items: the target and the seqNum. target identifies the cryptographic module or collection of cryptographic modules that are the target 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:

*target identifies the cryptographic modules or community of cryptographic modules that are the target of the query. 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 HardwareModuleIdentifiers 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 cryptographic modules. A sequence of these object identifiers is used if more than one community is the target of the message. A cryptographic module 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.

*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] (, “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.).

4.2. TAMP Status Query Response

TOC

The TAMP Status Response message is a reply by a cryptographic module to a valid TAMP Status Query message. The TAMP Status Response message provides information about the trust anchors that are currently installed in the cryptographic module and the list of communities to which the cryptographic module belongs, if any. The TAMP Status Response message MAY be signed or unsigned. A TAMP Status Response message MUST be signed if the cryptographic module is capable of signing it.
The TAMP Status Response content type has the following syntax:

```
tamp-status-response PKCS7-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 }

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                 TrustAnchorInfoList,
  continPubKeyDecryptAlg AlgorithmIdentifier,
  communities            CommunityIdentifierList OPTIONAL }

TrustAnchorInfoList ::= SEQUENCE SIZE (1..MAX) OF TrustAnchorInfo
```

The fields of TAMPStatusResponse are used as follows:

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

* query identifies the TAMPStatusQuery to which the cryptographic module 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.

* 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.

The fields of TerseStatusResponse are used as follows:

* taKeyIds contains a sequence of key identifiers. Each trust anchor contained in the cryptographic module is represented by one key identifier. The apex trust anchor is represented by the

first key identifier in the sequence, which contains the key identifier of the operational public key.

* communities is OPTIONAL. When present, it contains a sequence of object identifiers. Each object identifier names one community to which this cryptographic module belongs. When the module belongs to no communities, this field is omitted.

The fields of VerboseStatusResponse are used as follows:

* taInfo contains a sequence of TrustAnchorInfo structures. One entry in the sequence is provided for each trust anchor contained in the cryptographic module. The apex trust anchor is the first trust anchor in the sequence.

* contInPubKeyDecryptAlg indicates the decryption algorithm needed to decrypt the currently installed apex trust anchor contingency public key.

* communities is OPTIONAL. When present, it contains a sequence of object identifiers. Each object identifier names one community to which this cryptographic module belongs. When the module belongs to no communities, this field is omitted.

The fields of TrustAnchorInfo are described in Section 3.

4.3. Trust Anchor Update

TOC

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 cryptographic module MUST be an intended receipient of the update, the sequence number checking described in Section 6 MUST be successful when the TAMP message source is a trust anchor, and the digital signature MUST be validated using the apex trust anchor operational public key, a management trust anchor authorized for the id-ct-TAMP-update content type, 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 for the id-ct-TAMP-update content type, and the cryptographic module is an intended recipient of the TAMP message, then the cryptographic module 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 MUST be returned.
The Trust Anchor Update content type has the following syntax:

```
tamp-update PKCS7-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 }

TrustAnchorUpdate ::= CHOICE {
  add     [1] EXPLICIT TrustAnchorInfo,
  remove  [2] PublicKeyInfo,
  change  [3] TrustAnchorChangeInfo }

TrustAnchorChangeInfo ::= SEQUENCE {
  pubKey      PublicKeyInfo,
  keyId       KeyIdentifier OPTIONAL,
  mgmtTAType  [0] MgmtTrustAnchorInfo OPTIONAL,
  taTitle     [1] TrustAnchorTitle OPTIONAL,
  certPath    [2] CertPathControls OPTIONAL }
```

The fields of TAMPUpdate are used as follows:

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

*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.

*msgRef contains two items: the target and the seqNum. target identifies the cryptographic module or collection of cryptographic modules that are the target 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.

*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.

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:

* add is used to insert a new management or identity trust anchor into the cryptographic module. The TrustAnchorInfo structure is used to provide the trusted public key and all of the information associated with it. However, the action MUST fail if the subordination checks described in Section 7 are not satisfied. See Section 3 for a discussion of the TrustAnchorInfo structure. The apex trust anchor cannot be introduced into a cryptographic module using this action; therefore taType MUST NOT use ApexTrustAnchorInfo. The privileges 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 TrustAnchorInfo structure, except the seqNum field, MUST be treated as a successful addition. When the seqNum field does not match the most recently stored sequence number, the larger value MUST be stored by the cryptographic module. An attempt to add a management or identity trust anchor that is already present with the same keyId and pubKey values, but with different values for any of the fields in the TrustAnchorInfo structure other than the seqNum field, MUST result in an error.

* remove is used to delete an existing management or identity trust anchor from the cryptographic module, including the deletion of the management trust anchor associated with the TAMP message signer. However, the action MUST fail if the subordination checks described in Section 7 are not satisfied. The public key contained in PublicKeyInfo 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 privileges 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 cryptographic module in an unrecoverable configuration.

* change is used to update the information associated with an existing management or identity trust anchor in the cryptographic

module. The public key contained in the PublicKeyInfo field of TrustAnchorChangeInfo names the to-be-updated trust anchor. However, the action MUST fail 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 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.

The fields of TrustAnchorChangeInfo are used as follows:

* 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 cryptographic module storage.

* keyId is OPTIONAL, and when present, it contains the public key identifier of the trust anchor public key. 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.

* mgmtTAType is OPTIONAL, and when present, it carries information specific to the management trust anchor using the MgmtTrustAnchorInfo structure. This structure can be used to convert an identity trust anchor to a management trust anchor. There is not a way to use a single Trust Anchor Update message to convert a management trust anchor to an identity trust anchor. If this structure is not present, then the previous taType is preserved. The syntax and semantics of MgmtTrustAnchorInfo is discussed in Section 3. Each of the updates MUST satisfy the subordination checks described in Section 7. Normally, the sequence number for the management trust anchor is updated by receiving a signed TAMP message, including the Sequence Number Adjust message. The seqNum field is an alternative mechanism for advancing the sequence number values stored in a cryptographic module. When this integer value is present, the provided value is stored if it is greater than the currently stored value. When this integer value is not present, the previous value is preserved.

* 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.

* 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 is discussed in Section 3.

4.4. Trust Anchor Update Confirm

TOC

The Trust Anchor Update Confirm message is a reply by a cryptographic module 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 cryptographic module is capable of signing it.
The Trust Anchor Update Confirm content type has the following syntax:

```
tamp-update-confirm PKCS7-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   TrustAnchorInfoList }
```

The fields of TAMPUpdateConfirm are used as follows:

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

*update identifies the TAMPUpdate message to which the cryptographic module 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.

*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 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:

*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.

*taInfo contains a sequence of TrustAnchorInfo structures. One entry in the sequence is provided for each trust anchor contained in the cryptographic module. These represent the state of the trust anchors after the updates have been processed. See Section 3 for a discussion of the TrustAnchorInfo structure. The apex trust anchor is the first trust anchor in the sequence.

4.5. Apex Trust Anchor Update

TOC

The Apex Trust Anchor Update message replaces both the operational and the contingency public keys associated with the apex trust anchor. Each cryptographic module has exactly one apex trust anchor. Since the apex trust anchor represents the ultimate authority over the cryptographic module, 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 cryptographic module 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 cryptographic module 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 cryptographic module 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 cryptographic module MUST be deleted. That is, the new apex trust anchor MUST be the only trust anchor remaining in the cryptographic module. If the Apex Trust Anchor Update message is valid and the clearCommunities flag is set to TRUE, then all community identifiers stored in the cryptographic module MUST be deleted.

The SignedData structure includes a 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 cryptographic module is an intended recipient of the TAMP message, then the cryptographic module 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 MUST 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 PKCS7-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,
  apexTA             TrustAnchorInfo }
```

The fields of TAMPApexUpdate are used as follows:

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

*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.

*msgRef contains two items: the target and the seqNum. target identifies the cryptographic module or collection of cryptographic modules that are the target 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.

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

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

*apexTA provides the information for the replacement apex trust anchor. The TrustAnchorInfo structure is used to provide the trusted public key and all of the information associated with it. See Section 3 for a discussion of the TrustAnchorInfo structure; the taType MUST use the apex choice. The pubKey, keyId, taTitle, and certPath fields apply to the operational public key of the apex trust anchor.

TOC

4.6. Apex Trust Anchor Update Confirm

The Apex Trust Anchor Update Confirm message is a reply by a cryptographic module 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 cryptographic module is capable of signing it.
The Apex Trust Anchor Update Confirm content type has the following syntax:

```
tamp-apex-update-confirm PKCS7-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                  TrustAnchorInfoList,
  communities             CommunityIdentifierList OPTIONAL }
```

The fields of TAMPApexUpdateConfirm are used as follows:

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

*apexReplace identifies the Apex Trust Anchor Update message to which the cryptographic module 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.

* 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:

* 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.

* taInfo contains a sequence of TrustAnchorInfo structures. One entry in the sequence is provided for each trust anchor contained in the cryptographic module. These represent the state of the trust anchors after the apex trust anchor update has been processed. See Section 3 for a description of the TrustAnchorInfo structure. The apex trust anchor is the first trust anchor in the sequence.

* communities is OPTIONAL. When present, it contains a sequence of object identifiers. Each object identifier names one community to which this cryptographic module belongs. When the module belongs to no communities, this field is omitted.

4.7. Community Update

TOC

The cryptographic module 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 cryptographic module MUST be a target of the update, the sequence number checking described in Section 6 MUST be successful when the TAMP message source is a trust anchor, and the digital signature MUST be validated by the apex trust anchor operational public key, a management trust anchor authorized for the id-ct-TAMP-communityUpdate content type, or via an X.509 certification path originating with such a trust anchor.

If the cryptographic module supports the Community Update message, the digital signature on the Community Update message is valid, sequence number checking is successful, the signer is authorized for the id-ct-TAMP-communityUpdate content type, and the cryptographic module is an intended recipient of the TAMP message, then the cryptographic module 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 MUST be returned.
The Community Update message contains a batch of updates, and all of the updates MUST be accepted for the cryptographic module to return a successful Community Update Confirm message. The remove updates, if present, MUST be processed before the add updates. This approach prevents community identifiers that are intended to be mutually exclusive from being installed by a successful addition and a failed removal.
The Community Update content type has the following syntax:

```
tamp-community-update PKCS7-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 {
  add     [1] CommunityIdentifierList OPTIONAL,
  remove  [2] CommunityIdentifierList OPTIONAL }
   -- At least one MUST be present
```

The fields of TAMPCommunityUpdate are used as follows:

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

*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.

*msgRef contains two items: the target and the seqNum. target identifies the cryptographic module or collection of cryptographicmodules that are the target 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.

*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. 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 cryptographic module community list MUST NOT be changed.
A description of the syntax associated with each of these actions follows:

*remove is used to remove one or more community identifiers from the cryptographic module.

*add is used to insert one or more new community identifiers into the cryptographic module.

4.8. Community Update Confirm

TOC

The Community Update Confirm message is a reply by a cryptographic module 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 cryptographic module is unchanged. The Community Update Confirm message MAY be signed or unsigned. A Community Update Confirm message MUST be signed if the cryptographic module is capable of signing it.
The Community Update Confirm content type has the following syntax:

```
tamp-community-update-confirm PKCS7-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:

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

*update identifies the Community Update message to which the cryptographic module 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.

*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 has been processed. 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:

*status contains a single status code, indicating the success or failure of the Community Update message has been processed. 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.

*communities contains the sequence of community identifiers present in the cryptographic module after the update is processed. When the module belongs to no communities, this field is omitted.

4.9. Sequence Number Adjust

TOC

The cryptographic module maintains the current sequence number for the apex trust anchor and each management trust anchor. Sequence number processing is discussed in Section 6. The Sequence Number Adjust message can be used provide the most recently used sequence number to one or more cryptographic modules, thereby reducing the possibility of replay. The Sequence Number Adjust message MUST be signed. For the Sequence Number Adjust message to be valid, the cryptographic module 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 a management trust anchor that is authorized for the id-ct-TAMP-seqNumAdjust content type.
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 for the id-ct-TAMP-seqNumAdjust content type, and the cryptographic module is an intended recipient of the TAMP message, then the cryptographic module 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 MUST 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 PKCS7-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:

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

*msgRef contains two items: the target and the seqNum. target identifies the cryptographic module or collection of cryptographic modules that are the target 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 source, 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 source. If sequence number checking completes successfully, then the sequence number is adjusted, otherwise it remains unchanged.

4.10. Sequence Number Adjust Confirm

TOC

The Sequence Number Adjust Confirm message is a reply by a cryptographic module 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 cryptographic module is unchanged. The Sequence Number Adjust Confirm message MAY be signed or unsigned. A Sequence Number Adjust Confirm message MUST be signed if the cryptographic module is capable of signing it.

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

```
tamp-sequence-number-adjust-confirm PKCS7-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:

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

*adjust identifies the Sequence Number Adjust message to which the cryptographic module 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.

*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

TOC

The TAMP Error message is a reply by a cryptographic module 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 cryptographic module is capable of signing it.
The object identifier names the TAMP Error message content:

```
tamp-error PKCS7-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:

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

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

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

*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

TOC

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),
 other                             (127) }
```

The various values of StatusCode are used as follows:

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

*decodeFailure is used to indicate that the cryptographic module was unable to successfully decode the provided message. The specified content type and the provided content do not match.

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

*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.

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

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

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

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

*badUnsignedAttrs is used to indicate that the unsignedAttrs within SignerInfo contains an attribute other than the contingency-public-key-decrypt-key unsigned attribute, which is the only unsigned attribute supported by this specification.

*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.

*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.

*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 Sec. 7.

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

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

*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.

*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.

*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.

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

*unsupportedTAMPMsgType indicates that the TAMP message could not be processed because the cryptographic module 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 cryptographic module 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.

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

*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.

*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.

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

*incorrectTarget indicates that the query, update, or adjust message could not be processed because the cryptographic module is not the intended recipient. The target cryptographic module is identified in one of two ways. HardwareModule identifies the cryptographic module by the module type and serial number; in which case, either one or both of these values does not match the responding cryptographic module. Alternatively, community identifies a group of cryptographic modules; in which case, the responding cryptographic module does not belong to the identified group.

*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.

*trustAnchorNotFound indicates that a change to a trust anchor was requested, but the referenced trust anchor is not represented in the cryptographic module.

*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 the cryptographic module. In addition, this status code is used if the algorithm is supported, but the parameters associated with the algorithm are not supported.

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

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

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

*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.

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

*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.

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

*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

TOC

The sequence number processing facilities in TAMP represent a balance between replay protection, operational considerations, and cryptographic module 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 signed directly by a trust anchor, and skipping these checks whenever the TAMP message originator is represented by a certificate.

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 signed directly by 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 cryptographic module 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 cryptographic module 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 signed by that trust anchor is not rejected based on sequence number checks, and the sequence number from that first TAMP message is stored. The sequence number for that trust anchor could also be updated by the OPTIONAL sequence number field of a Trust Anchor Update message that is received after the trust anchor is installed. The TAMP message recipient MUST maintain a database of the

most recent sequence number from a successfully processed TAMP message from each 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 cryptographic module for the apex trust anchor and each management trust anchor.

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

---

7. Subordination Processing

TOC

The apex trust anchor is unconstrained, which means that subordination checking is not performed on Trust Anchor Update messages signed with the apex trust anchor operational public key. 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 a management trust anchor associated with the id-ct-TAMP-update content type, either directly or via an X.509 certification path originating with the apex trust anchor operational public key or such a management trust anchor. The following subordination checks MUST also be performed as part of validation.

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 privileges of the TAMP message signer MUST be greater than or equal to the privileges of the trust anchor in the update. The privileges of the TAMP message signer and the to-be-updated trust anchor are determined based on the applicable CMS Content Constraints. Specifically, the privileges of the TAMP message signer are determined as described in section 3 of [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature

Constraints X.509 Certificate Extension," in progress.) passing the special value anyContentType and an empty set of attributes as input; the privileges of the to-be-updated trust anchor are determined as described below. If the privileges of a trust anchor in an update exceed the privileges of the signer, that update MUST be rejected. Each update is considered and accepted or rejected individually without regard to other updates in the TAMP message. The privileges of the to-be-updated trust anchors are determined as follows:

* If the to-be-updated trust anchor is the subject of an add operation, the privileges are read from the taType.mgmt.taUsage field of the corresponding TrustAnchorInfo in the update.

* If the to-be-updated trust anchor is the subject of a remove operation, the trust anchor is located in the message recipient's trust anchor store using the public key included in the update. The privileges are read from the taType.mgmt.taUsage (or equivalent) field in the to-be-updated trust anchor.

* If the to-be-updated trust anchor is the subject of a change operation, the trust anchor has two distinct sets of privileges that MUST be checked. The trust anchor's pre-change privileges are determined by locating the trust anchor in the message recipient's trust anchor store using the public key included in the update and reading the privileges from the taType.mgmt.taUsage (or equivalent) field in the trust anchor. The trust anchor's post-change privileges are read from the taType.mgmt.taUsage field of the corresponding TrustAnchorChangeInfo in the update. If the taType.mgmt.taUsage field is not present, then the trust anchor's post-change privileges are equivalent to the trust anchor's pre-change privileges.

The following steps can be used to determine if a Trust Anchor Update message signer is authorized to manage each to-be-updated trust anchor contained in a Trust Anchor Update message.

* The TAMP message signer's CMS Content Constraints privileges are determined as described in section 3 of [CCC] (Housley, R. and C. Wallace, "Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension," in progress.) passing the special value anyContentType and an empty set of attributes as input. Note that it is possible for the TAMP message signer to have more than one possible certification path that will authorize it to sign Trust Anchor Update messages, with each certification path resulting in different CMS Content Constraints privileges. The update is authorized if the processing below succeeds for any one certification path of the TAMP message signer. The resulting cms_permitted_content_types variable is

used to check each to-be-updated trust anchor contained in the update message. The message signer MUST be authorized for the Trust Anchor Update message. This can be confirmed using the steps described in section 4 of [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.).

*The privileges of each to-be-updated trust anchor in the TAMP message MUST be checked against the message signer's privileges (represented in the message signer’s cms_permitted_content_types computed above) using the following steps. For change operations, the following steps MUST be performed for the trust anchor's pre-change privileges and the trust anchor's post-change privileges.

-Operations on identity trust anchors are permitted provided the message signer is authorized for the Trust Anchor Update message.

-If the to-be-updated trust anchor is unconstrained, the message signer MUST also be unconstrained, i.e., the message signer’s cms_permitted_content_types MUST be set to the special value anyContentType. If the to-be-updated trust anchor is unconstrained and the message signer is not, then the message signer is not authorized to manage the trust anchor and the update MUST be rejected.

-The message signer's authorization for each permitted content type MUST be checked using the state variables and procedures similar to those described in sections 3.2 and 3.3 of [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.). For each permitted content type in the to-be-updated trust anchor's privileges,

oSet cms_effective_attributes equal to the value of the attrConstraints field from the permitted content type.

oIf the content type does not match an entry in the message signer’s cms_permitted_content_types, the message signer is not authorized to manage the trust anchor and the update MUST be rejected. Note, the special value anyContentType produces a match for all content types with the resulting matching entry containing the content type, canSource set to TRUE and attrConstraints absent.

oIf the content type matches an entry in the message signer’s cms_permitted_content_types, the canSource field of the entry is FALSE and the canSource field in the to-be-updated trust anchor's privilege is TRUE, the message

signer is not authorized to manage the trust anchor and the update MUST be rejected.

- If the content type matches an entry in the message signer's cms_permitted_content_types and the entry's attrConstraints field is present, then constraints MUST be checked. For each attrType in the entry's attrConstraints, a corresponding attribute MUST be present in cms_effective_attributes containing values from the entry's attrConstraints. If values appear in the corresponding attribute that are not in the entry's attrConstraints or if there is no corresponding attribute, the message signer is not authorized to manage the trust anchor and the update MUST be rejected.

Once these steps are completed, if the update has not been rejected, then the message signer is authorized to manage the to-be-updated trust anchor.

Note that a management trust anchor that has only the id-ct-TAMP-update permitted content type is useful only for managing identity trust anchors. It can sign a Trust Anchor Update message, but it cannot impact a management trust anchor that is associated with any other content type.

---

8. Implementation Considerations

TOC

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 anchor 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.

---

9. Security Considerations

TOC

The majority of this specification is devoted to the syntax and semantics of TAMP messages. It relies on other specifications, especially [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) and [RFC3280] (Housley, R., Polk, W., Ford, W., and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” April 2002.), for the syntax and semantics of CMS protecting content types and X.509 certificates, respectively. Since TAMP messages that change the trust anchor state of a cryptographic module 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 cryptographic module 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 a CMS content constraints certificate extension [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.) 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 cryptographic modules, 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 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] (, “FIPS Pub 186: Digital

Signature Standard," May 1994.). 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 cryptographic modules 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 cryptographic modules 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 into the cryptographic module.
Compromise of the Apex Trust Anchor operational private key permits unauthorized parties to generate signed messages that will be acceptable to all cryptographic modules 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 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 associated with the trust anchor. If the CA is subordinate to a management trust anchor, the scope of potential damage caused by a private key compromise is also limited by the CMS content constraints certificate extension [CCC] (Housley, R. and C. Wallace, "Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension," in progress.) in the CA certificate, the CMS content constraints on any superior CA certificates, and the CMS content constraints on the parent management 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 associated with the trust anchor. If the certified public key is subordinate to a management trust anchor, the scope of potential damage caused by a private key compromise is also limited by the CMS content constraints certificate extension [CCC] (Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.) in the end entity certificate, the CMS content constraints on any superior CA certificates, and the CMS content constraints on the parent management trust anchor.

Compromise of a cryptographic module’s digital signature private key permits unauthorized parties to generate signed TAMP response messages, masquerading as the cryptographic module.

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.

To implement TAMP, a cryptographic module needs 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, signed attributes, and certificates.

---

10. IANA Considerations

TOC

There are no IANA considerations. Please delete this section prior to RFC publication.

---

11. References

TOC

11.1. Normative References

TOC

[CCC]	Housley, R. and C. Wallace, “Cryptographic Message Syntax (CMS) Content Signature Constraints X.509 Certificate Extension,” in progress.
[ClearConstr]	Turner, S., “Clearance and CA Clearance Constraints Certificate Extensions,” in progress.
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

[RFC2634]	Hoffman, P., "Enhanced Security Services for S/MIME," RFC 2634, June 1999 (TXT).
[RFC3280]	Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, April 2002 (TXT).
[RFC3629]	Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).
[RFC3852]	Housley, R., "Cryptographic Message Syntax (CMS)," RFC 3852, July 2004 (TXT).
[RFC4049]	Housley, R., "BinaryTime: An Alternate Format for Representing Date and Time in ASN.1," RFC 4049, April 2005 (TXT).
[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.

11.2. Informative References

TOC

[DSS]	"FIPS Pub 186: Digital Signature Standard," May 1994.
[PKCS#6]	"PKCS #6: Extended-Certificate Syntax Standard, Version 1.5," November 1993.
[RFC3281]	Farrell, S. and R. Housley, "An Internet Attribute Certificate Profile for Authorization," RFC 3281, April 2002 (TXT).
[RFC4108]	Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages," RFC 4108, August 2005 (TXT).
[X.208]	"ITU-T Recommendation X.208 - Specification of Abstract Syntax Notation One (ASN.1)," 1988.
[X.501]	"ITU-T Recommendation X.501 - The Directory - Models," 1993.
[X.509]	"ITU-T Recommendation X.509 - The Directory - Authentication Framework," 2000.

Appendix A. ASN.1 Modules

TOC

Appendix A.1 provides the normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680] (, "ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One," 1997.). Appendix A.2 provides a module using ASN.1 as defined in [X.208] (, "ITU-T Recommendation X.208 -

Specification of Abstract Syntax Notation One (ASN.1),” 1988.). The module in 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. A.2 is included as a courtesy to developers using ASN.1 compilers that do not support current ASN.1.

TOC

A.1. ASN.1 Module Using 1993 Syntax

```
TrustAnchorManagementProtocolVersion2
    { joint-iso-ccitt(2) country(16) us(840) organization(1)
      gov(101) dod(2) infosec(1) modules(0) TBD }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
    ATTRIBUTE, Attribute
      FROM InformationFramework -- from [X.501]
        { joint-iso-itu-t(2) ds(5) module(1)
          informationFramework(1) 4 }
    ContentType
      FROM CryptographicMessageSyntax2004 -- [RFC3852]
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }
    AlgorithmIdentifier, Certificate, Name
      FROM PKIX1Explicit88 -- from [RFC3280]
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-pkix1-explicit(18) }
    CertificatePolicies, KeyIdentifier, NameConstraints
      FROM PKIX1Implicit88 -- from <xref target="RFC3280"/>
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-pkix1-implicit(19) }
    CMSContentConstraints
      FROM CMSContentConstraintsCertExtn-93 -- [CCC]
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          cmsContentConstraints-93(42) }
    CAClearanceConstraints
       FROM Clearance-CAClearanceConstraints93 -- from [ClearConstr]
        { joint-iso-ccitt(2) country(16) us(840) organization(1)
           gov(101) dod(2) infosec(1) modules(0) 9997 } ;
                 -- Placeholder for TBD

-- Trust Anchor Information

TrustAnchorInfo ::= SEQUENCE {
  version   [0] TAMPVersion DEFAULT v2,
  pubKey    PublicKeyInfo,
  keyId     KeyIdentifier,
  taType    TrustAnchorType,
  taTitle   TrustAnchorTitle OPTIONAL,
  certPath  CertPathControls OPTIONAL }
```

```
PublicKeyInfo ::= SEQUENCE {
  algorithm  AlgorithmIdentifier,
  publicKey  BIT STRING }

KeyIdentifier ::= OCTET STRING

TrustAnchorType ::= CHOICE {
  apex   [0] ApexTrustAnchorInfo,
  mgmt   [1] MgmtTrustAnchorInfo,
  ident  [2] NULL }

ApexTrustAnchorInfo ::= SEQUENCE {
  continPubKey  ApexContingencyKey,
  seqNum        SeqNumber OPTIONAL }

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

SeqNumber ::= INTEGER (0..9223372036854775807)

MgmtTrustAnchorInfo ::= SEQUENCE {
  taUsage  TrustAnchorUsage,
  seqNum   SeqNumber OPTIONAL }

TrustAnchorUsage ::= CMSContentConstraints

CMSContentConstraints ::= ContentTypeConstraintList

ContentTypeConstraintList ::= SEQUENCE SIZE (1..MAX) OF
                          ContentTypeConstraint

ContentTypeConstraint ::= SEQUENCE {
  contentType      ContentType,
  canSource        BOOLEAN DEFAULT TRUE,
  attrConstraints  AttrConstraintList OPTIONAL }

AttrConstraintList ::= SEQUENCE SIZE (1..MAX) OF AttrConstraint

AttrConstraint ::= SEQUENCE {
  attrType    AttributeType,
  attrValues  SET SIZE (1..MAX) OF AttributeValue }

ContentType ::= OBJECT IDENTIFIER

TrustAnchorTitle ::= UTF8String (SIZE (1..64))

CertPathControls ::= SEQUENCE {
  taName          Name,
  selfSigned      [0] Certificate OPTIONAL,
```

```
  policySet       [1] CertificatePolicies OPTIONAL,
  policyFlags     [2] CertPolicyFlags OPTIONAL,
  clearanceConstr [3] CAClearanceConstraints OPTIONAL,
  nameConstr      [4] NameConstraints OPTIONAL }

CertPolicyFlags ::= BIT STRING {
  inhibitPolicyMapping    (0),
  requireExplicitPolicy   (1),
  inhibitAnyPolicy        (2) }


-- 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

PKCS7-CONTENT-TYPE ::= TYPE-IDENTIFIER

TAMPContentTypes PKCS7-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 PKCS7-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 }

TargetIdentifier ::= CHOICE {
  hwModules       [1] HardwareModuleIdentifierList,
  communities     [2] CommunityIdentifierList,
  allModules      [3] NULL }

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 (1..MAX) OF Community

Community ::= OBJECT IDENTIFIER


-- TAMP Status Response Message

tamp-status-response PKCS7-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 }

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                 TrustAnchorInfoList,
  continPubKeyDecryptAlg AlgorithmIdentifier,
  communities            CommunityIdentifierList OPTIONAL }

TrustAnchorInfoList ::= SEQUENCE SIZE (1..MAX) OF TrustAnchorInfo


-- Trust Anchor Update Message

tamp-update PKCS7-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 }

TrustAnchorUpdate ::= CHOICE {
  add             [1] EXPLICIT TrustAnchorInfo,
  remove          [2] PublicKeyInfo,
  change          [3] TrustAnchorChangeInfo }

TrustAnchorChangeInfo ::= SEQUENCE {
  pubKey          PublicKeyInfo,
  keyId           KeyIdentifier OPTIONAL,
  taType          [0] TrustAnchorChangeType OPTIONAL,
  taTitle         [1] TrustAnchorTitle OPTIONAL,
  certPath        [2] CertPathControls OPTIONAL }

TrustAnchorChangeType ::= CHOICE {
  mgmt            [1] MgmtTrustAnchorInfo,
  ident           [2] NULL }


-- Trust Anchor Update Confirm Message

tamp-update-confirm PKCS7-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                TrustAnchorInfoList }


-- Apex Trust Anchor Update Message

tamp-apex-update PKCS7-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,
  apexTA              TrustAnchorInfo }


-- Apex Trust Anchor Update Confirm Message

tamp-apex-update-confirm PKCS7-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                TrustAnchorInfoList,
  communities           CommunityIdentifierList OPTIONAL }


-- Community Update Message

tamp-community-update PKCS7-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 {
  add             [1] CommunityIdentifierList OPTIONAL,
  remove          [2] CommunityIdentifierList OPTIONAL }
  -- At least one must be present


-- Community Update Confirm Message

tamp-community-update-confirm PKCS7-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 PKCS7-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 Message

tamp-sequence-number-adjust-confirm PKCS7-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 PKCS7-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),
  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

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


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

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


END
```

TOC

A.2. ASN.1 Module Using 1988 Syntax

```
TrustAnchorManagementProtocolVersion2_88
   { joint-iso-ccitt(2) country(16) us(840) organization(1)
     gov(101) dod(2) infosec(1) modules(0) 997 }
                 -- Placeholder for TBD

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
  ContentType
    FROM CryptographicMessageSyntax2004 -- [RFC3852]
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }
  AlgorithmIdentifier, Certificate, Name, Attribute
    FROM PKIX1Explicit88 -- [RFC3280]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-explicit(18) }
  CertificatePolicies, KeyIdentifier, NameConstraints
    FROM PKIX1Implicit88 -- [RFC3280]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-implicit(19) }
  CMSContentConstraints
    FROM CMSContentConstraintsCertExtn-88 -- [CCC]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        cmsContentConstr-88(41) }
  CAClearanceConstraints
    FROM Clearance-CAClearanceConstraints88 -- [ClearConstr]
      { joint-iso-ccitt(2) country(16) us(840) organization(1)
         gov(101) dod(2) infosec(1) modules(0) 9998 } ;
               -- Placeholder for TBD

-- Trust Anchor Information

TrustAnchorInfo ::= SEQUENCE {
  version         [0] TAMPVersion DEFAULT v2,
  pubKey          PublicKeyInfo,
  keyId           KeyIdentifier,
  taType          TrustAnchorType,
  taTitle         TrustAnchorTitle OPTIONAL,
  certPath        CertPathControls OPTIONAL }

PublicKeyInfo ::= SEQUENCE {
  algorithm       AlgorithmIdentifier,
  publicKey       BIT STRING }
```

```
TrustAnchorType ::= CHOICE {
  apex            [0] ApexTrustAnchorInfo,
  mgmt            [1] MgmtTrustAnchorInfo,
  ident           [2] NULL }

ApexTrustAnchorInfo ::= SEQUENCE {
  contInPubKey    ApexContingencyKey,
  seqNum          SeqNumber OPTIONAL }

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

SeqNumber ::= INTEGER (0.. 9223372036854775807)

MgmtTrustAnchorInfo ::= SEQUENCE {
  taUsage         TrustAnchorUsage,
  seqNum          SeqNumber OPTIONAL }

TrustAnchorUsage ::= CMSContentConstraints

TrustAnchorTitle ::= UTF8String (SIZE (1..64))

CertPathControls ::= SEQUENCE {
  taName          Name,
  selfSigned      [0] Certificate OPTIONAL,
  policySet       [1] CertificatePolicies OPTIONAL,
  policyFlags     [2] CertPolicyFlags OPTIONAL,
  clearanceConstr [3] CAClearanceConstraints OPTIONAL,
  nameConstr      [4] NameConstraints OPTIONAL }

CertPolicyFlags ::= BIT STRING {
  inhibitPolicyMapping    (0),
  requireExplicitPolicy   (1),
  inhibitAnyPolicy        (2) }


-- 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) }

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

TargetIdentifier ::= CHOICE {
  hwModules       [1] HardwareModuleIdentifierList,
  communities     [2] CommunityIdentifierList,
  allModules      [3] NULL }

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 (1..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 }

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                 TrustAnchorInfoList,
  contInPubKeyDecryptAlg AlgorithmIdentifier,
  communities            CommunityIdentifierList OPTIONAL }

TrustAnchorInfoList ::= SEQUENCE SIZE (1..MAX) OF TrustAnchorInfo


-- 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 }

TrustAnchorUpdate ::= CHOICE {
  add             [1] EXPLICIT TrustAnchorInfo,
  remove          [2] PublicKeyInfo,
  change          [3] TrustAnchorChangeInfo }

TrustAnchorChangeInfo ::= SEQUENCE {
  pubKey          PublicKeyInfo,
  keyId           KeyIdentifier OPTIONAL,
  mgmtTAType      [0] MgmtTrustAnchorInfo OPTIONAL,
  taTitle         [1] TrustAnchorTitle OPTIONAL,
  certPath        [2] CertPathControls 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                TrustAnchorInfoList }


-- 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,
  apexTA              TrustAnchorInfo }


-- 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                 TrustAnchorInfoList,
  communities            CommunityIdentifierList 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 }
-- Placeholder for TBD

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


-- Sequence Number Adjust Message

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

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),
  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

END
```

Authors' Addresses

TOC

	Russ Housley
	Vigil Security, LLC
	918 Spring Knoll Drive
	Herndon, VA 20170
Email:	housley@vigilsec.com
	Raksha Reddy
	National Security Agency
	Suite 6751
	9800 Savage Road
	Fort Meade, MD 20755
Email:	r.reddy@radium.ncsc.mil
	Carl Wallace
	Cygnacom Solutions
	Suite 5200
	7925 Jones Branch Drive
	McLean, VA 22102
Email:	cwallace@cygnacom.com

Full Copyright Statement

TOC

THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.