Trust Anchor Management Protocol (TAMP) & CMS Content Constraints (CCC)

70th IETF
Vancouver

Carl Wallace
cwallace@cygnacom.com

Raksha Reddy
r.reddy@radium.ncsc.mil

Agenda

- Trust Anchor Management Problem Statement
  - http://www.ietf.org/internet-drafts/draft-wallace-ta-mgmt-problem-statement-02.txt
- TAMP overview
  - http://www.ietf.org/internet-drafts/draft-housley-tamp-00.txt
- CCC overview
  - http://www.ietf.org/internet-drafts/draft-housley-cms-content-constraints-extn-00.txt
- Q & A

What are trust anchors?

- Trust anchors (TAs) are trusted public keys with with associated information
  - Used for signature verification
  - Associated information varies with TA purpose
    - RFC3280 requires issuer name, public key algorithm, public key and optionally, the public key parameters associated with the public key to support certification path validation
- TAs are used for various purposes
  - Certification path validation
  - Verification of signed objects, including firmware, timestamps, OCSP responses, keys, etc.
- TAs are maintained in trust anchor stores, which are sets of one or more trust anchors

Problem statement

- There is currently no standard mechanism for managing trust anchor stores
  - Proprietary means abound
  - Remote management can be difficult (and is generally beyond the reach of PKI policy authorities)
  - Some application-specific standards are being developed (draft-ietf-dnsext-trustupdate-timers)
- No standard representation for trust anchors
  - Self-signed certificates are a de facto means of installing names and keys for use with PKI
    - However, self-signed certificates do not provide hooks for TA management
  - Uniform representation may not be necessary even if common management means are used

General Proposal

- Define a protocol for managing trust anchor stores
  - Generic trust anchor representation requirements include trust anchor name, public key information and trust anchor usage
  - Enable add/remove/query operations on trust anchor stores
- Primary aim is to reduce reliance on out-of-band trust mechanisms
  - After initial trust anchors have been installed, out-of-band means should not be necessary

TAMP Summary

- Eleven message formats
  - Five request/response pairs
  - TAMPError message
- All request messages signed; all response messages optionally signed
- Uses CMS SignedData for message integrity
- Trust anchor (TA) privileges defined and enforced using CMS Content Constraints (CCC)
- TAs represented using TrustAnchorInfo structure

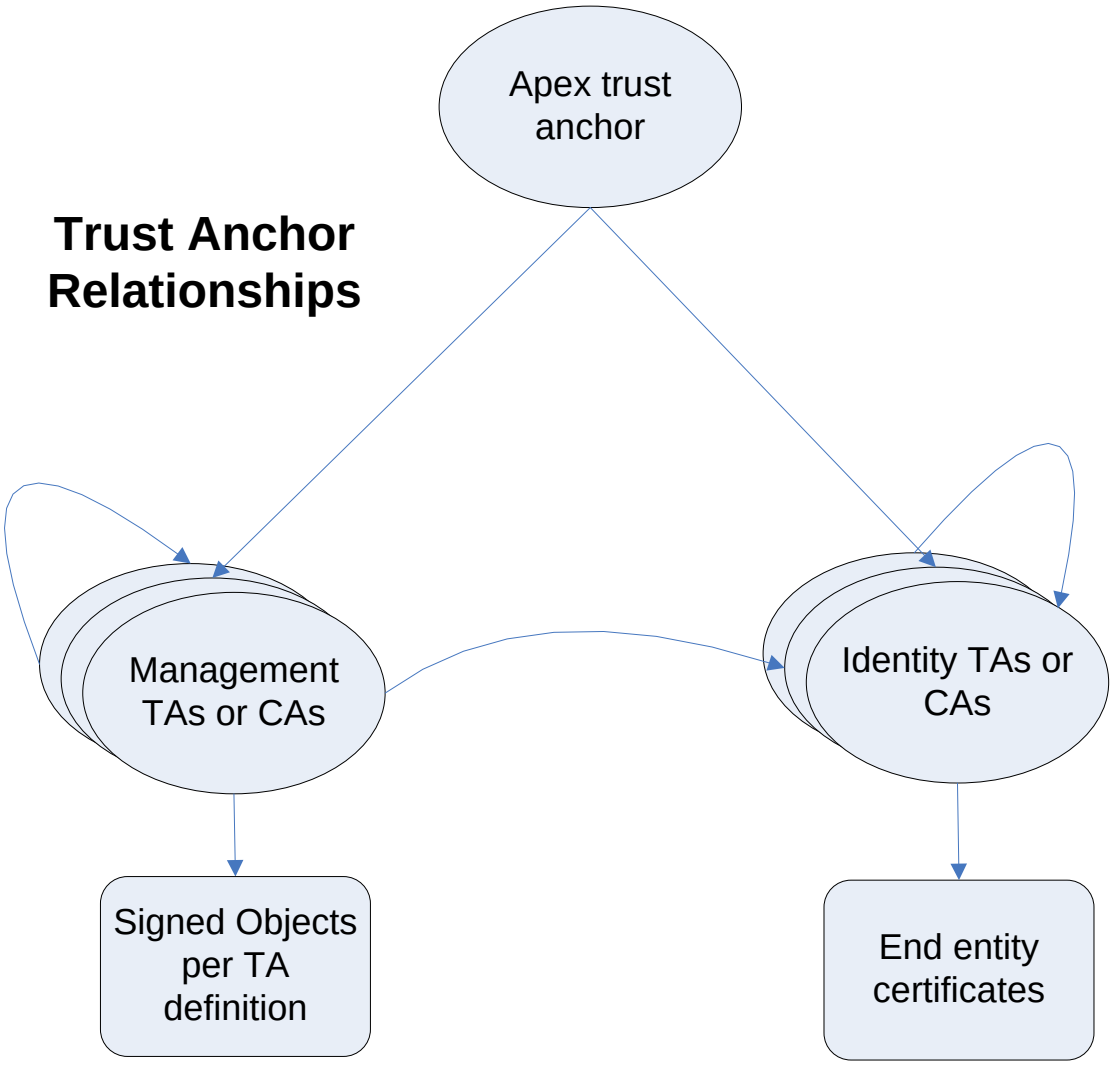
Trust anchor types

- Three types: Apex, Management, Identity
- Apex trust anchor
  - One per trust anchor store
  - Superior to all other trust anchors; Unconstrained
  - Different structure than other trust anchors. Includes two public keys: operational and contingency
    - The operational key is used in same manner as other trust anchors
    - The contingency key can only be used to update the apex trust anchor. It is distributed in encrypted form. Single use.
    - Contingency key is useful if operational key is compromised or lost
    - Contingency key may use a different algorithm than operational key

Trust anchor types (continued)

- Management trust anchors
  - Enable authorization checking for management messages
    - Where management messages are authenticated using CMS (primarily focused on RFC 4108, TAMP and draft-ietf-keyprov-symmetrickeyformat)
- Identity trust anchors
  - Used to validate certification paths
  - Generally associated with non-management applications

Trust Anchor Relationships



- One per trust anchor store
- Represented as a trust anchor only (no certificates)
- Initial Apex TA add during store initialization
- Contains two keys: operational and contingency
- Managed via Apex Trust Anchor Update messages which must be validated using operational key or contingency key

- Zero or more per trust anchor store
- May be represented as Trust Anchor or public key certificate
- Trust anchor instances are managed via Trust Anchor Update messages which must be validated using public key authorized for TAMP
- Certificate instances must validate to a trust anchor authorized to issue certificates

TrustAnchorInfo

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

- taType indicates the type of trust anchor
  - ApexTrustAnchorInfo, MgmtTrustAnchorInfo or NULL
- taTitle is human readable name for the trust anchor
- certPath provides the controls needed to initialize an X.509 certification path validation algorithm implementation
  - When absent, TA cannot be used to validate certificates
- New structure aims to help minimize size by avoiding fields in certificates that are not processed during validation

ApexTrustAnchorInfo

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

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

SeqNumber ::= INTEGER (0..9223372036854775807)

-- attribute used to convey decryption key
id-aa-TAMP-contingencyPublicKeyDecryptKey
     OBJECT IDENTIFIER ::= { id-attributes 63 }

PlaintextSymmetricKey ::= OCTET STRING
```

ApexTrustAnchorInfo (continued)

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

- ApexTrustAnchorInfo appears in the taType field of TrustAnchorInfo
  - Carries the contingency key and optional sequence number
- continPubKey is the encrypted contingency key
  - When decrypted, yields a PublicKeyInfo structure
  - Decrypted using the contingencyPublicKeyDecryptKey attribute
    - Appears as an unsigned attribute on messages that are verified using the contingency key
- seqNum can be used to set the initial sequence number value associated with the operational public key in the encapsulating TrustAnchorInfo

MgmtTrustAnchorInfo

```
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
```

MgmtTrustAnchorInfo(continued)

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

- MgmtTrustAnchorInfo appears in the taType field of TrustAnchorInfo
  - Carries the CCC privileges for the TA and optional sequence number
- taUsage identifies the types of CMS contents the TA can be used to verify
- seqNum can be used to set the initial sequence number value associated with the public key in the encapsulating TrustAnchorInfo

TAMPMsgRef

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

- TAMPMsgRef is used to target TAMP messages and to indicate sequence number
  - Target identifies the trust anchor stores or community of stores that are the target of a message
    - Can target all recipients, specific hardware types or instances or via community identifiers
  - Sequence number is a single use value that can be used to match request and response messages

Targeting trust anchor stores

- TAMP enables the generation of very targeted trust anchor management messages
  - Allows generation of messages targeting a specific trust anchor store
- Community identifiers allow trust anchor stores to be aggregated into groups
  - Groups created and managed using TAMP messages

TAMPStatusQuery and TAMPStatusResponse

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

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

- Enables list of trust anchors resident in a trust store to be requested and returned
  - Terse responses list key identifiers only
  - Verbose responses provide list of TrustAnchorInfo structures

TrustAnchorUpdate

```
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 }
```

- Includes a TrustAnchorInfo to add to the store, identifies a trust anchor to remove by public key or presents new details to replace those associated with a key already present in a trust store
- Each operation is subject to subordination checks

TrustAnchorUpdateConfirm

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

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

VerboseUpdateConfirm ::= SEQUENCE {
      status   StatusCodeList,
      taInfo   TrustAnchorInfoList }
```

- Returns status of an update operation one of two way
  - As a list of status codes (one per element in the update message)
  - As a list of status codes and TAs (represents state following update)

TAMPApexUpdate and TAMPApexUpdateConfirm

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

- Verified using either operational or contingency key
- Replacement information carried in apexTA field
- If clearTrustAnchors is TRUE, then all management and identity TAs must be deleted leaving on the newly installed apex TA
- If clearCommunities is TRUE, then all community identifiers must be deleted, leaving none
- TAMPApexUpdateConfirm (not shown) can return single status code value (terse) or a status with a list of all TAs and communities (verbose)

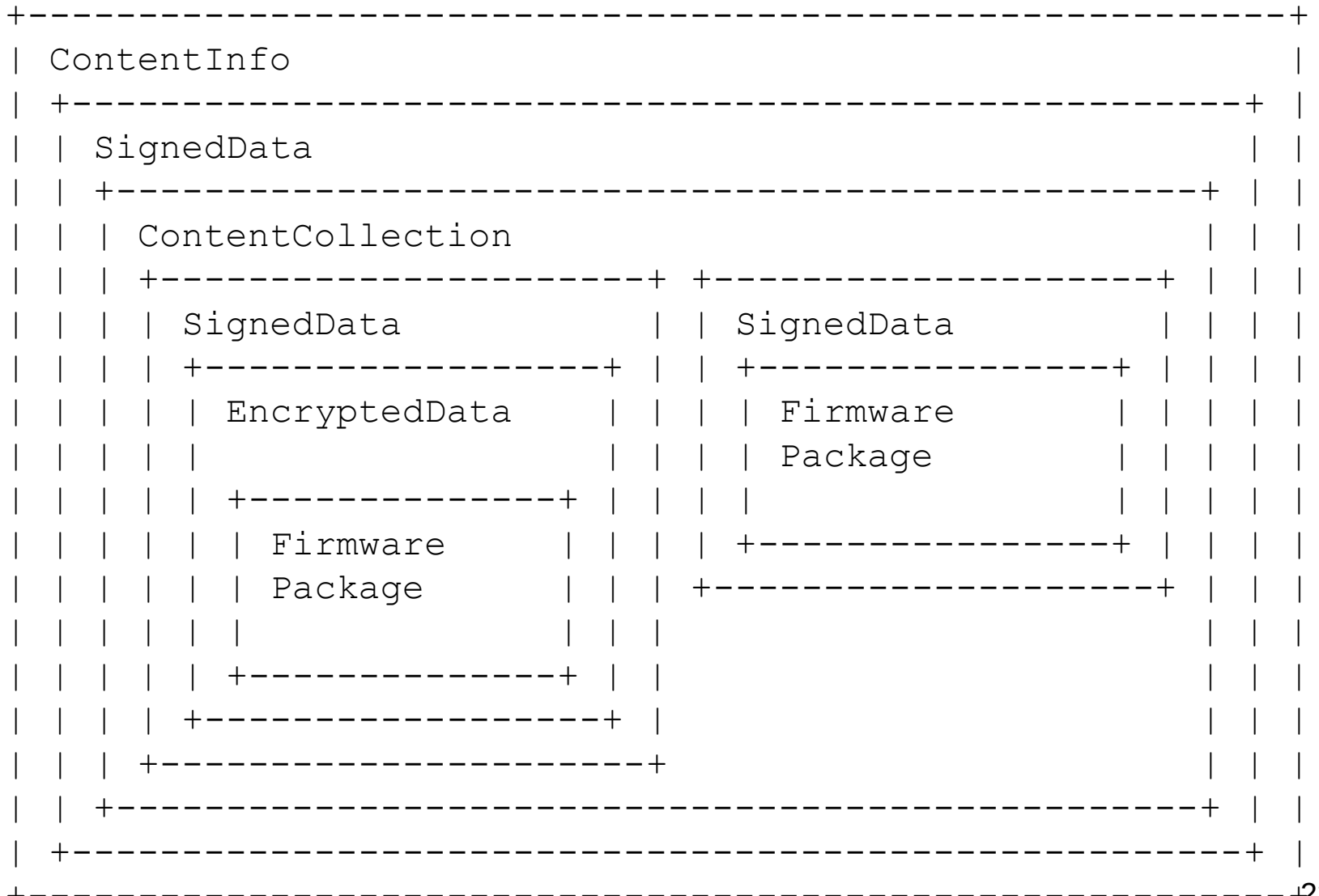
Other types

- TAMPCommunityUpdate allows community identifiers to be added or removed from the list of communities maintained by a trust anchor store (i.e., the communities to which the store belongs)
  - Terse and verbose response types
- SequenceNumberAdjust can be used to provide the most recently used sequence number to one or more stores
  - Reduces possibility of replay
  - Response simply includes a status code indicating the success or failure of the sequence number adjust message processing

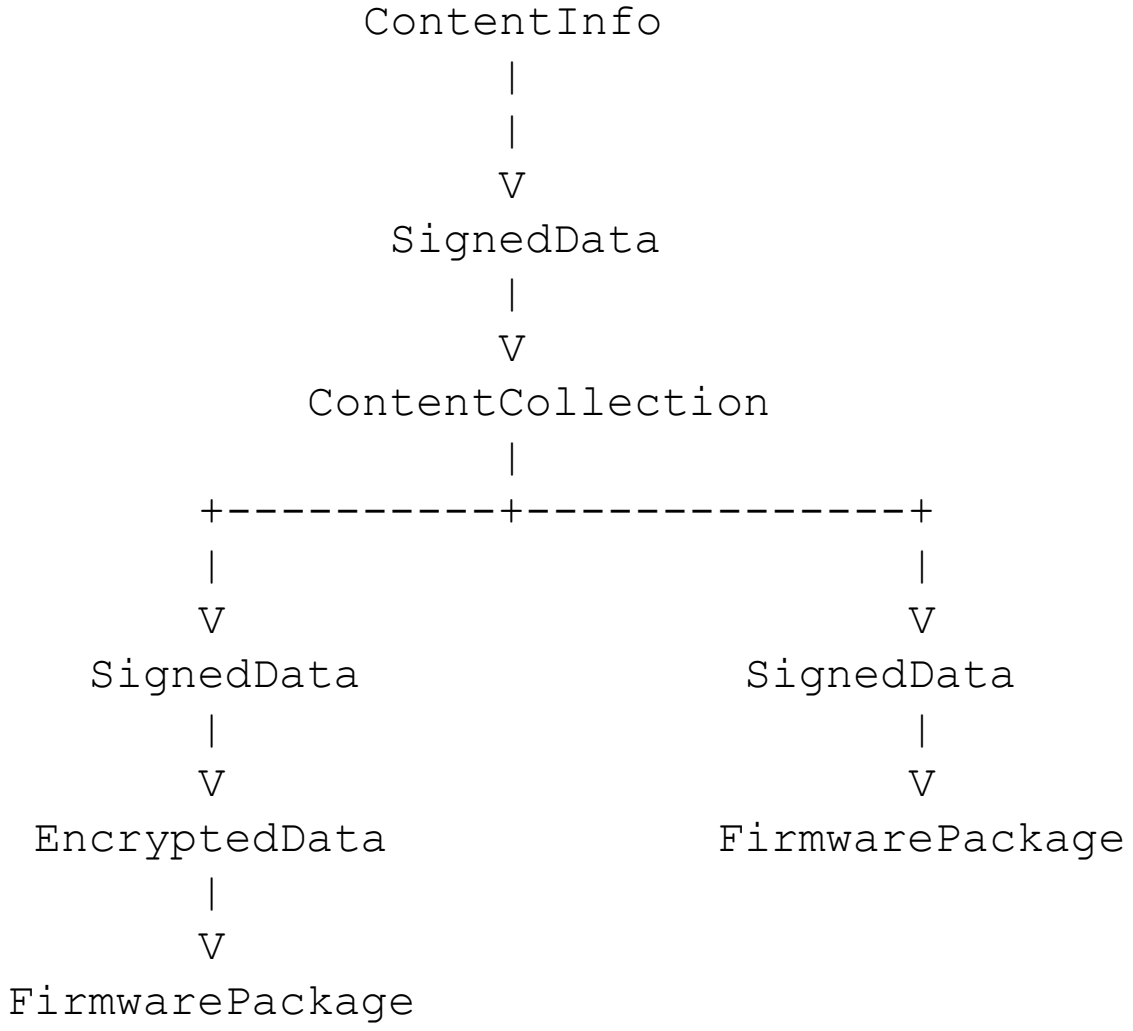
CCC Summary

- Used to restrict the types of CMS protected objects that can be verified using a particular public key
- Expressed as permitted content types and constraints on attribute values
- Privileges represented as either a TrustAnchorInfo field or as a certificate extension
  - Privileges for a particular content originator are output from certification path validation (intersection of CCC values in path)
- Object type represented by CMS content type OID
  - Object attributes collected by processing authenticated layers in a CMS message
  - Each party collaborating to produce a signed or authenticated content must be authorized for the innermost content types and attribute values

CMS Paths


ContentInfo
SignedData
ContentCollection
SignedData
EncryptedData
Firmware
Package
SignedData
Firmware
Package

CMS Paths (continued)



CMS Paths (continued)

- Two types of leaf nodes: encrypted leaf nodes and payload leaf nodes
  - Encrypted leaf nodes are one of the following types: EncryptedData, EnvelopedData or AuthEnvelopedData
  - Payload lead nodes are all other leaf node types (non-encrypted CMS types like SignedData, ContentCollection, etc. are not leaf nodes)

Subject permissions

- Identify the types of leaf nodes for which a subject can serve as originator or collaborator
  - Constrain attribute values a subject can use for particular types of leaf nodes
- Collected and evaluated during path processing
  - Content type and attributes collected from CMS path are provided as input
  - Constraints are collected from trust anchor and intersected with certificate-based constraints and evaluated during validation wrap-up
  - Default attributes are returned along with constraints for the input content type
    - Constraints may be used when processing the content

Object type and attributes

- Public keys and signed or authenticated attributes are collected from a CMS path
  - For encrypted leaf nodes, these are simply returned and may be used for further processing
  - For payload leaf nodes, a path is validated to each public key providing the object type and attributes as input
    - Each public key must be authorized for object type and each attribute value
    - Public key used to verify the signature or MAC closest to the payload leaf node must be authorized as a source for the object type

Summary

- Use TAMP to manage TAs and associated CMS-focused privileges
- Use CCC to express and enforce CMS-focused privileges
- Use RFC4108 and draft-ietf-keyprov-symmetrickeyformat to package firmware and keys with source authentication controlled by TAMP and CCC
- CCC could be useful for other CMS-protected payloads
  - Attributes give flexibility beyond extendedKeyUsage