Network Working Group
Request for Comments: 4108
Category: Standards Track

R. Housley
Vigil Security

August 2005

Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages

Status of This Memo

Copyright Notice

Abstract

This document describes the use of the Cryptographic Message Syntax (CMS) to protect firmware packages, which provide object code for one or more hardware module components. CMS is specified in RFC 3852. A digital signature is used to protect the firmware package from undetected modification and to provide data origin authentication. Encryption is optionally used to protect the firmware package from disclosure, and compression is optionally used to reduce the size of the protected firmware package. A firmware package loading receipt can optionally be generated to acknowledge the successful loading of a firmware package. Similarly, a firmware package load error report can optionally be generated to convey the failure to load a firmware package.

Table of Contents

1. Introduction

This document describes the use of the Cryptographic Message Syntax (CMS) [CMS] to protect firmware packages. This document also describes the use of CMS for receipts and error reports for firmware package loading. The CMS is a data protection encapsulation syntax that makes use of ASN.1 [X.208-88, X.209-88]. The protected firmware package can be associated with any particular hardware module; however, this specification was written with the requirements of cryptographic hardware modules in mind, as these modules have strong security requirements.

The firmware package contains object code for one or more programmable components that make up the hardware module. The firmware package, which is treated as an opaque binary object, is digitally signed. Optional encryption and compression are also supported. When all three are used, the firmware package is compressed, then encrypted, and then signed. Compression simply

reduces the size of the firmware package, allowing more efficient processing and transmission. Encryption protects the firmware package from disclosure, which allows transmission of sensitive firmware packages over insecure links. The encryption algorithm and mode employed may also provide integrity, protecting the firmware package from undetected modification. The encryption protects proprietary algorithms, classified algorithms, trade secrets, and implementation techniques. The digital signature protects the firmware package from undetected modification and provides data origin authentication. The digital signature allows the hardware module to confirm that the firmware package comes from an acceptable source.

If encryption is used, the firmware-decryption key must be made available to the hardware module via a secure path. The key might be delivered via physical media or via an independent electronic path. One optional mechanism for distributing the firmware-decryption key is specified in Section 2.3.1, but any secure key distribution mechanism is acceptable.

The signature verification public key must be made available to the hardware module in a manner that preserves its integrity and confirms its source. CMS supports the transfer of certificates, and this facility can be used to transfer a certificate that contains the signature verification public key (a firmware-signing certificate). However, use of this facility introduces a level of indirection. Ultimately, a trust anchor public key must be made available to the hardware module. Section 1.2 establishes a requirement that the hardware module store one or more trust anchors.

Hardware modules may not be capable of accessing certificate repositories or delegated path discovery (DPD) servers [DPD&DPV] to acquire certificates needed to complete a certification path. Thus, it is the responsibility of the firmware package signer to include sufficient certificates to enable each module to validate the firmware-signer certificate (see Section 2.1.2). Similarly, hardware modules may not be capable of accessing a certificate revocation list (CRL) repository, an OCSP responder [OCSP], or a delegated path validation (DPV) server [DPD&DPV] to acquire revocation status information. Thus, if the firmware package signature cannot be validated solely with the trust anchor public key and the hardware module is not capable of performing full certification path validation, then it is the responsibility of the entity loading a package into a hardware module to validate the firmware-signer certification path prior to loading the package into a hardware module. The means by which this external certificate revocation status checking is performed is beyond the scope of this specification.

Hardware modules will only accept firmware packages with a valid digital signature. The signature is either validated directly using the trust anchor public key or using a firmware-signer certification path that is validated to the trust anchor public key. Thus, the trust anchors define the set of entities that can create firmware packages for the hardware module.

The disposition of a previously loaded firmware package after the successful validation of another firmware package is beyond the scope of this specification. The amount of memory available to the hardware module will determine the range of alternatives.

In some cases, hardware modules can generate receipts to acknowledge the loading of a particular firmware package. Such receipts can be used to determine which hardware modules need to receive an updated firmware package whenever a flaw in an earlier firmware package is discovered. Hardware modules can also generate error reports to indicate the unsuccessful firmware package loading. To implement either receipt or error report generation, the hardware module is required to have a unique permanent serial number. Receipts and error reports can be either signed or unsigned. To generate digitally signed receipts or error reports, a hardware module MUST be issued its own private signature key and a certificate that contains the corresponding signature validation public key. In order to save memory with the hardware module, the hardware module might store a certificate designator instead of the certificate itself. The private signature key requires secure storage.

1.1. Terminology

In this document, the key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are to be interpreted as described in [STDWORDS].

1.2. Architectural Elements

The architecture includes the hardware module, the firmware package, and a bootstrap loader. The bootstrap loader MUST have access to one or more trusted public keys, called trust anchors, to validate the signature on the firmware package. If a signed firmware package load receipt or error report is created on behalf of the hardware module, then the bootstrap loader MUST have access to a private signature key to generate the signature and the signer identifier for the corresponding signature validation certificate or its designator. A signature validation certificate MAY be included to aid signature validation. To implement this optional capability, the hardware module MUST have a unique serial number and a private signature key; the hardware module MAY also include a certificate that contains the

corresponding signature validation public key. These items MUST be installed in the hardware module before it is deployed. The private key and certificate can be generated and installed as part of the hardware module manufacture process. Figure 1 illustrates these architectural elements.

ASN.1 object identifiers are the preferred means of naming the architectural elements.

Details of managing the trust anchors are beyond the scope of this specification. However, one or more trust anchors MUST be installed in the hardware module using a secure process before it is deployed. These trust anchors provide a means of controlling the acceptable sources of firmware packages. The hardware module vendor can include provisions for secure, remote management of trust anchors. One approach is to include trust anchors in the firmware packages themselves. This approach is analogous to the optional capability described later for updating the bootstrap loader.

In a cryptographic hardware module, the firmware package might implement many different cryptographic algorithms.

When the firmware package is encrypted, the firmware-decryption key and the firmware package MUST both be provided to the hardware module. The firmware-decryption key is necessary to use the associated firmware package. Generally, separate distribution mechanisms will be employed for the firmware-decryption key and the firmware package. An optional mechanism for securely distributing the firmware-decryption key with the firmware package is specified in Section 2.3.1.

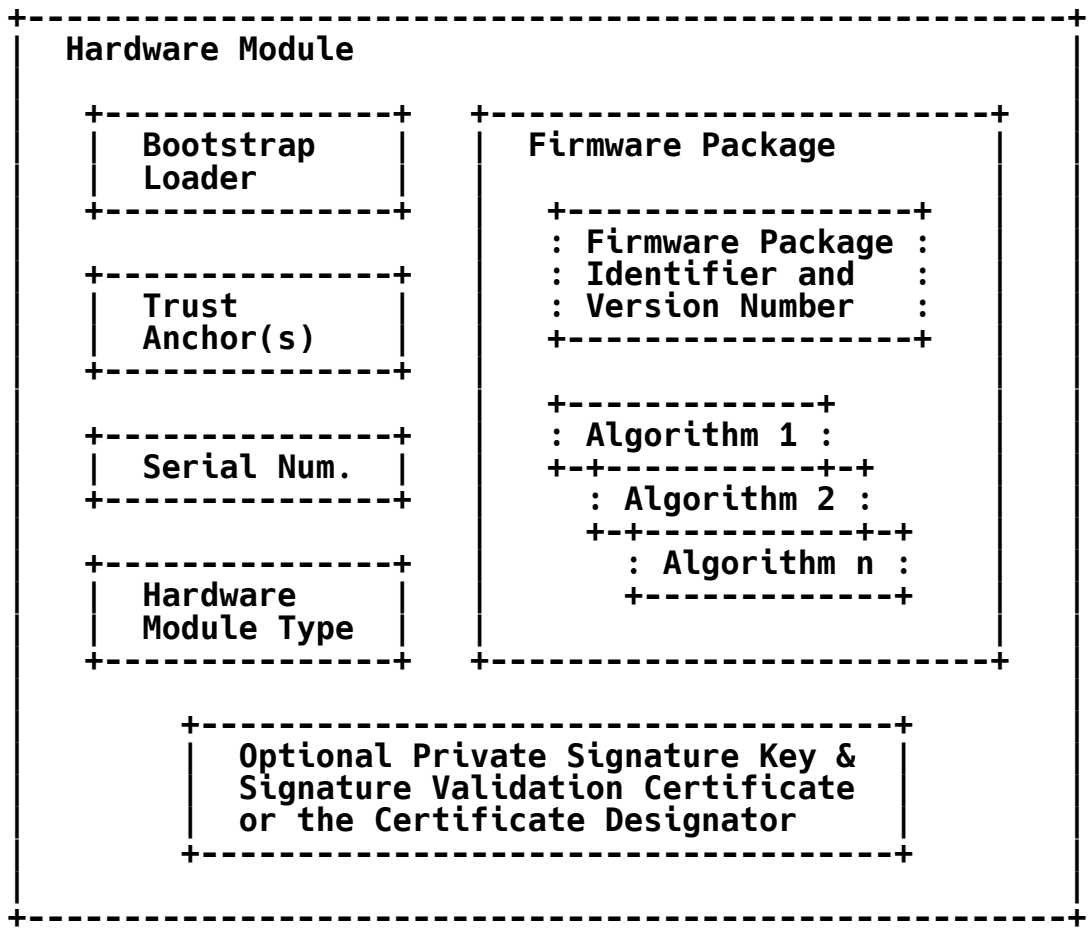

Figure 1. Architectural Elements

1.2.1. Hardware Module Requirements

Many different vendors develop hardware modules, and each vendor typically identifies its modules by product type (family) and revision level. A unique object identifier MUST name each hardware module type and revision.

Each hardware module within a hardware module family SHOULD have a unique permanent serial number. However, if the optional receipt or error report generation capability is implemented, then the hardware module MUST have a unique permanent serial number. If the optional receipt or error report signature capability is implemented, then the hardware module MUST have a private signature key and a certificate containing the corresponding public signature validation key or its designator. If a serial number is present, the bootstrap loader uses

it for authorization decisions (see Section 2.2.8), receipt generation (see Section 3), and error report generation (see Section 4).

When the hardware module includes more than one firmware-programmable component, the bootstrap loader distributes components of the package to the appropriate components within the hardware module after the firmware package is validated. The bootstrap loader is discussed further in Section 1.2.3.

1.2.2. Firmware Package Requirements

Two approaches to naming firmware packages are supported: legacy and preferred. Firmware package names are placed in a CMS signed attribute, not in the firmware package itself.

Legacy firmware package names are simply octet strings, and no structure is assumed. This firmware package name form is supported in order to facilitate existing configuration management systems. We assume that the firmware signer and the bootstrap loader will understand any internal structure to the octet string. In particular, given two legacy firmware package names, we assume that the firmware signer and the bootstrap loader will be able to determine which one represents the newer version of the firmware package. This capability is necessary to implement the stale version feature. If a firmware package with a disastrous flaw is released, subsequent firmware package versions MAY designate a stale legacy firmware package name in order to prevent subsequent rollback to the stale version or versions earlier than the stale version.

Preferred firmware package names are a combination of the firmware package object identifier and a version number. A unique object identifier MUST identify the collection of features that characterize the firmware package. For example, firmware packages for a cable modem and a wireless LAN network interface card warrant distinct object identifiers. Similarly, firmware packages that implement distinct suites of cryptographic algorithms and modes of operation, or that emulate different (non-programmable) cryptographic devices warrant distinct object identifiers. The version number MUST identify a particular build or release of the firmware package. The version number MUST be a monotonically increasing non-negative integer. Generally, an earlier version is replaced with a later one. If a firmware package with a disastrous flaw is released, subsequent firmware package versions MAY designate a stale version number to prevent subsequent rollback to the stale version or versions earlier than the stale version.

Firmware packages are developed to run on one or more hardware module type. The firmware package digital signature MUST bind the list of supported hardware module object identifiers to the firmware package.

In many cases, the firmware package signature will be validated directly with the trust anchor public key, avoiding the need to construct certification paths. Alternatively, the trust anchor can delegate firmware package signing to another public key through a certification path. In the latter case, the firmware package SHOULD contain the certificates needed to construct the certification path that begins with a certificate issued by the trust anchors and ends with a certificate issued to the firmware package signer.

The firmware package MAY contain a list of community identifiers. These identifiers name the hardware modules that are authorized to load the firmware package. If the firmware package contains a list of community identifiers, then the bootstrap loader MUST reject the firmware package if the hardware module is not a member of one of the identified communities.

When a hardware module includes multiple programmable components, the firmware package SHOULD contain executable code for all of the components. Internal tagging within the firmware package MUST tell the bootstrap loader which portion of the overall firmware package is intended for each component; however, this tagging is expected to be specific to each hardware module. Because this specification treats the firmware package as an opaque binary object, the format of the firmware package is beyond the scope of this specification.

1.2.3. Bootstrap Loader Requirements

The bootstrap loader MUST have access to a physical interface and any related driver or protocol software necessary to obtain a firmware package. The same interface SHOULD be used to deliver receipts and error reports. Details of the physical interface as well as the driver or protocol software are beyond the scope of this specification.

The bootstrap loader can be a permanent part of the hardware module, or it can be replaced by loading a firmware package. In Figure 1, the bootstrap loader is implemented as separate logic within the hardware module. Not all hardware modules will include the ability to replace or update the bootstrap loader, and this specification does not mandate such support.

If the bootstrap loader can be loaded by a firmware package, an initial bootstrap loader MUST be installed in non-volatile memory prior to deployment. All bootstrap loaders, including an initial

bootstrap loader if one is employed, MUST meet the requirements in this section. However, the firmware package containing the bootstrap loader MAY also contain other routines.

The bootstrap loader requires access to cryptographic routines. These routines can be implemented specifically for the bootstrap loader, or they can be shared with other hardware module features. The bootstrap loader MUST have access to a one-way hash function and digital signature verification routines to validate the digital signature on the firmware package and to validate the certification path for the firmware-signing certificate.

If firmware packages are encrypted, the bootstrap loader MUST have access to a decryption routine. Access to a corresponding encryption function is not required, since hardware modules need not be capable of generating firmware packages. Because some symmetric encryption algorithm implementations (such as AES [AES]) employ separate logic for encryption and decryption, some hardware module savings might result.

If firmware packages are compressed, the bootstrap loader MUST also have access to a decompression function. This function can be implemented specifically for the bootstrap loader, or it can be shared with other hardware module features. Access to a corresponding compression function is not required, since hardware modules need not be capable of generating firmware packages.

If the optional receipt generation or error report capability is supported, the bootstrap loader MUST have access to the hardware module serial number and the object identifier for the hardware module type. If the optional signed receipt generation or signed error report capability is supported, the bootstrap loader MUST also have access to a one-way hash function and digital signature routines, the hardware module private signing key, and the corresponding signature validation certificate or its designator.

The bootstrap loader requires access to one or more trusted public keys, called trust anchors, to validate the firmware package digital signature. One or more trust anchors MUST be installed in non-volatile memory prior to deployment. The bootstrap loader MUST reject a firmware package if it cannot validate the signature, which MAY require the construction of a valid certification path from the firmware-signing certificate to one of the trust anchors [PROFILE]. However, in many cases, the firmware package signature will be validated directly with the trust anchor public key, avoiding the need to construct certification paths.

The bootstrap loader MUST reject a firmware package if the list of supported hardware module type identifiers within the firmware package does not include the object identifier of the hardware module.

The bootstrap loader MUST reject a firmware package if the firmware package includes a list of community identifiers and the hardware module is not a member of one of the listed communities. The means of determining community membership is beyond the scope of this specification.

The bootstrap loader MUST reject a firmware package if it cannot successfully decrypt the firmware package using the firmware-decryption key available to the hardware module. The firmware package contains an identifier of the firmware-decryption key needed for decryption.

When an earlier version of a firmware package is replacing a later one, the bootstrap loader SHOULD generate a warning. The manner in which a warning is generated is highly dependent on the hardware module and the environment in which it is being used. If a firmware package with a disastrous flaw is released and subsequent firmware package versions designate a stale version, the bootstrap loader SHOULD prevent loading of the stale version and versions earlier than the stale version.

1.2.3.1. Legacy Stale Version Processing

In case a firmware package with a disastrous flaw is released, subsequent firmware package versions that employ the legacy firmware package name form MAY include a stale legacy firmware package name to prevent subsequent rollback to the stale version or versions earlier than the stale version. As described in the Security Considerations section of this document, the inclusion of a stale legacy firmware package name in a firmware package cannot completely prevent subsequent use of the stale firmware package. However, many hardware modules are expected to have very few firmware packages written for them, allowing the stale firmware package version feature to provide important protections.

Non-volatile storage for stale version numbers is needed. The number of stale legacy firmware package names that can be stored depends on the amount of storage that is available. When a firmware package is loaded and it contains a stale legacy firmware package name, then it SHOULD be added to a list kept in non-volatile storage. When subsequent firmware packages are loaded, the legacy firmware package

name of the new package is compared to the list in non-volatile storage. If the legacy firmware package name represents the same version or an older version of a member of the list, then the new firmware packages SHOULD be rejected.

The amount of non-volatile storage that needs to be dedicated to saving legacy firmware package names and stale legacy firmware packages names depends on the number of firmware packages that are likely to be developed for the hardware module.

1.2.3.2. Preferred Stale Version Processing

If a firmware package with a disastrous flaw is released, subsequent firmware package versions that employ preferred firmware package name form MAY include a stale version number to prevent subsequent rollback to the stale version or versions earlier than the stale version. As described in the Security Considerations section of this document, the inclusion of a stale version number in a firmware package cannot completely prevent subsequent use of the stale firmware package. However, many hardware modules are expected to have very few firmware packages written for them, allowing the stale firmware package version feature to provide important protections.

Non-volatile storage for stale version numbers is needed. The number of stale version numbers that can be stored depends on the amount of storage that is available. When a firmware package is loaded and it contains a stale version number, then the object identifier of the firmware package and the stale version number SHOULD be added to a list that is kept in non-volatile storage. When subsequent firmware packages are loaded, the object identifier and version number of the new package are compared to the list in non-volatile storage. If the object identifier matches and the version number is less than or equal to the stale version number, then the new firmware packages SHOULD be rejected.

The amount of non-volatile storage that needs to be dedicated to saving firmware package identifiers and stale version numbers depends on the number of firmware packages that are likely to be developed for the hardware module.

1.2.4. Trust Anchors

A trust anchor MUST consist of a public key signature algorithm and an associated public key, which MAY optionally include parameters. A trust anchor MUST also include a public key identifier. A trust anchor MAY also include an X.500 distinguished name.

The trust anchor public key is used in conjunction with the signature validation algorithm in two different ways. First, the trust anchor public key is used directly to validate the firmware package signature. Second, 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 firmware package signature.

The public key names the trust anchor, and each public key has a public key identifier. The public key identifier identifies the trust anchor as the signer when it is used directly to validate firmware package signatures. This key identifier can be stored with the trust anchor, or it can be computed from the public key whenever needed.

The optional trusted X.500 distinguished name MUST be present in order for the trust anchor public key to be used to validate an X.509 certification path. Without an X.500 distinguished name, certification path construction cannot use the trust anchor.

1.2.5. Cryptographic and Compression Algorithm Requirements

A firmware package for a cryptographic hardware module includes cryptographic algorithm implementations. In addition, a firmware package for a non-cryptographic hardware module will likely include cryptographic algorithm implementations to support the bootstrap loader in the validation of firmware packages.

A unique algorithm object identifier MUST be assigned for each cryptographic algorithm and mode implemented by a firmware package. A unique algorithm object identifier MUST also be assigned for each compression algorithm implemented by a firmware package. The algorithm object identifiers can be used to determine whether a particular firmware package satisfies the needs of a particular application. To facilitate the development of algorithm-agile applications, the cryptographic module interface SHOULD allow applications to query the cryptographic module for the object identifiers associated with each cryptographic algorithm contained in the currently loaded firmware package. Applications SHOULD also be able to query the cryptographic module to determine attributes associated with each algorithm. Such attributes might include the algorithm type (symmetric encryption, asymmetric encryption, key agreement, one-way hash function, digital signature, and so on), the algorithm block size or modulus size, and parameters for asymmetric algorithms. This specification does not establish the conventions for the retrieval of algorithm identifiers or algorithm attributes.

1.3. Hardware Module Security Architecture

The bootstrap loader MAY be permanently stored in read-only memory or separately loaded into non-volatile memory as discussed above.

In most hardware module designs, the firmware package execution environment offers a single address space. If it does, the firmware package SHOULD contain a complete firmware package load for the hardware module. In this situation, the firmware package does not contain a partial or incremental set of functions. A complete firmware package load will minimize complexity and avoid potential security problems. From a complexity perspective, the incremental loading of packages makes it necessary for each package to identify any other packages that are required (its dependencies), and the bootstrap loader needs to verify that all of the dependencies are satisfied before attempting to execute the firmware package. When a hardware module is based on a general purpose processor or a digital signal processor, it is dangerous to allow arbitrary packages to be loaded simultaneously unless there is a reference monitor to ensure that independent portions of the code cannot interfere with one another. Also, it is difficult to evaluate arbitrary combinations of software modules [SECREQMTS]. For these reasons, a complete firmware package load is RECOMMENDED; however, this specification allows the firmware signer to identify dependencies between firmware packages in order to handle all situations.

The firmware packages MAY have dependencies on routines provided by other firmware packages. To minimize the security evaluation complexity of a hardware module employing such a design, the firmware package MUST identify the package identifiers (and the minimum version numbers when the preferred firmware package name form is used) of the packages upon which it depends. The bootstrap loader MUST reject a firmware package load if it contains a dependency on a firmware package that is not available.

Loading a firmware package can impact the satisfactory resolution of dependencies of other firmware packages that are already part of the hardware module configuration. For this reason, the bootstrap loader MUST reject the loading of a firmware package if the dependencies of any firmware package in the resulting configurations will be unsatisfied.

1.4. ASN.1 Encoding

The CMS uses Abstract Syntax Notation One (ASN.1) [X.208-88, X.209-88]. 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) 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.509-88] 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.

In this specification, digitally signed structures MUST be encoded with DER. Other structures do not require DER, but the use of definite length encoding is strongly RECOMMENDED. By always using definite length encoding, the bootstrap loader will have fewer options to implement. In situations where there is very high confidence that only definite length encoding will be used, support for indefinite length decoding MAY be omitted.

1.5. Protected Firmware Package Loading

This document does not attempt to specify a physical interface, any related driver software, or a protocol necessary for loading firmware packages. Many different delivery mechanisms are envisioned, including portable memory devices, file transfer, and web pages. Section 2 of this specification defines the format that MUST be presented to the hardware module regardless of the interface that is used. This specification also specifies the format of the response that MAY be generated by the hardware module. Section 3 of this specification defines the format that MAY be returned by the hardware module when a firmware package loads successfully. Section 4 of this specification defines the format that MAY be returned by the hardware module when a firmware package load is unsuccessful. The firmware package load receipts and firmware package load error reports can be either signed or unsigned.

2. Firmware Package Protection

The Cryptographic Message Syntax (CMS) is used to protect a firmware package, which is treated as an opaque binary object. A digital signature is used to protect the firmware package from undetected modification and to provide data origin authentication. Encryption is optionally used to protect the firmware package from disclosure, and compression is optionally used to reduce the size of the protected firmware package. The CMS ContentInfo content type MUST always be present, and it MUST encapsulate the CMS SignedData content type. If the firmware package is encrypted, then the CMS SignedData content type MUST encapsulate the CMS EncryptedData content type. If the firmware package is compressed, then either the CMS SignedData

content type (when encryption is not used) or the CMS EncryptedData content type (when encryption is used) MUST encapsulate the CMS CompressedData content type. Finally, (1) the CMS SignedData content type (when neither encryption nor compression is used), (2) the CMS EncryptedData content type (when encryption is used, but compression is not), or (3) the CMS CompressedData content type (when compression is used) MUST encapsulate the simple firmware package using the FirmwarePkgData content type defined in this specification (see Section 2.1.5).

The firmware package protection is summarized as follows (see [CMS] for the full syntax):

```
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, -- Signer cert. path
  crls                 CertificateRevocationLists, -- Optional
  signerInfos          SET OF SignerInfo -- Only one
}

SignerInfo {
  version              CMSVersion, -- always set to 3
  sid                  SignerIdentifier,
  digestAlgorithm      DigestAlgorithmIdentifier,
  signedAttrs          SignedAttributes, -- Required
  signatureAlgorithm   SignatureAlgorithmIdentifier,
  signature            SignatureValue,
  unsignedAttrs        UnsignedAttributes -- Optional
}
```

```
EncapsulatedContentInfo {
  eContentType         id-encryptedData, -- (1.2.840.113549.1.7.6)
                       -- OR --
                       id-ct-compressedData,
                                 -- (1.2.840.113549.1.9.16.1.9)
                       -- OR --
                       id-ct-firmwarePackage,
                                 -- (1.2.840.113549.1.9.16.1.16)
  eContent             OCTET STRING
}                            -- Contains EncryptedData OR
                             -- CompressedData OR
                             -- FirmwarePkgData

EncryptedData {
  version              CMSVersion, -- Always set to 0
  encryptedContentInfo EncryptedContentInfo,
  unprotectedAttrs     UnprotectedAttributes -- Omit
}

EncryptedContentInfo {
  contentType          id-ct-compressedData,
                                 -- (1.2.840.113549.1.9.16.1.9)
                       -- OR --
                       id-ct-firmwarePackage,
                                 -- (1.2.840.113549.1.9.16.1.16)
  contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
  encryptedContent OCTET STRING
}                                -- Contains CompressedData OR
                                 -- FirmwarePkgData

CompressedData {
  version              CMSVersion, -- Always set to 0
  compressionAlgorithm CompressionAlgorithmIdentifier,
  encapContentInfo     EncapsulatedContentInfo
}

EncapsulatedContentInfo {
  eContentType         id-ct-firmwarePackage,
                                   -- (1.2.840.113549.1.9.16.1.16)
  eContent             OCTET STRING -- Contains FirmwarePkgData
}

FirmwarePkgData         OCTET STRING -- Contains firmware package
```

2.1. Firmware Package Protection CMS Content Type Profile

This section specifies the conventions for using the CMS ContentInfo, SignedData, EncryptedData, and CompressedData content types. It also defines the FirmwarePkgData content type.

2.1.1. ContentInfo

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

- contentType indicates the type of the associated content, and in this case, the encapsulated type is always SignedData. The id-signedData (1.2.840.113549.1.7.2) object identifier MUST be present in this field.

- content holds the associated content, and in this case, the content field MUST contain SignedData.

2.1.2. SignedData

The SignedData content type [CMS] contains the signed firmware package (which might be compressed, encrypted, or compressed and then encrypted prior to signature), the certificates needed to validate the signature, and one digital signature value. The fields of SignedData are used as follows:

version is the syntax version number, and in this case, it MUST be set to 3.

digestAlgorithms is a collection of message digest algorithm identifiers, and in this case, it MUST contain a single message digest algorithm identifier. The message digest algorithm employed by the firmware package signer MUST be present.

encapContentInfo contains the signed content, consisting of a content type identifier and the content itself. The use of the EncapsulatedContentInfo type is discussed further in Section 2.1.2.2.

certificates is an optional collection of certificates. If the trust anchor signed the firmware package directly, then certificates SHOULD be omitted. If it did not, then certificates SHOULD include the X.509 certificate of the firmware package signer. The set of certificates SHOULD be sufficient for the bootstrap loader to construct a certification path from the trust anchor to the firmware-signer's certificate. PKCS#6 extended certificates

[PKCS#6] and attribute certificates (either version 1 or version 2) [X.509-97, X.509-00, ACPROFILE] MUST NOT be included in the set of certificates.

crls is an optional collection of certificate revocation lists (CRLs), and in this case, CRLs SHOULD NOT be included by the firmware package signer. It is anticipated that firmware packages may be generated, signed, and made available in repositories for downloading into hardware modules. In such contexts, it would be difficult for the firmware package signer to include timely CRLs in the firmware package. However, because the CRLs are not covered by the signature, timely CRLs MAY be inserted by some other party before the firmware package is delivered to the hardware module.

signerInfos is a collection of per-signer information, and in this case, the collection MUST contain exactly one SignerInfo. The use of the SignerInfo type is discussed further in Section 2.1.2.1.

2.1.2.1. SignerInfo

The firmware package signer is represented in the SignerInfo type. The fields of SignerInfo are used as follows:

version is the syntax version number, and it MUST be 3.

sid identifies the signer's public key. CMS supports two alternatives: issuerAndSerialNumber and subjectKeyIdentifier. However, the bootstrap loader MUST support the subjectKeyIdentifier alternative, which identifies the signer's public key directly. When this public key is contained in a certificate, this identifier SHOULD appear in the X.509 subjectKeyIdentifier extension.

digestAlgorithm identifies the message digest algorithm, and any associated parameters, used by the firmware package signer. It MUST contain the message digest algorithms employed by the firmware package signer. (Note that this message digest algorithm identifier MUST be the same as the one carried in the digestAlgorithms value in SignedData.)

signedAttrs is an optional collection of attributes that are signed along with the content. The signedAttrs are optional in the CMS, but in this specification, signedAttrs are REQUIRED for the firmware package; however, implementations MUST ignore unrecognized signed attributes. The SET OF attributes MUST be DER

encoded [X.509-88]. Section 2.2 of this document lists the attributes that MUST be included in the collection; other attributes MAY be included as well.

signatureAlgorithm identifies the signature algorithm, and any associated parameters, used by the firmware package signer to generate the digital signature.

signature is the digital signature value.

unsignedAttrs is an optional SET of attributes that are not signed. As described in Section 2.3, this set can only contain a single instance of the wrapped-firmware-decryption-key attribute and no others.

2.1.2.2. EncapsulatedContentInfo

The EncapsulatedContentInfo content type encapsulates the firmware package, which might be compressed, encrypted, or compressed and then encrypted prior to signature. The firmware package, in any of these formats, is carried within the EncapsulatedContentInfo type. The fields of EncapsulatedContentInfo are used as follows:

eContentType is an object identifier that uniquely specifies the content type, and in this case, the value MUST be id-encryptedData (1.2.840.113549.1.7.6), id-ct-compressedData (1.2.840.113549.1.9.16.1.9), or id-ct-firmwarePackage (1.2.840.113549.1.9.16.1.16). When eContentType contains id-encryptedData, the firmware package was encrypted prior to signing, and may also have been compressed prior to encryption. When it contains id-ct-compressedData, the firmware package was compressed prior to signing, but was not encrypted. When it contains id-ct-firmwarePackage, the firmware package was not compressed or encrypted prior to signing.

eContent contains the signed firmware package, which might also be encrypted, compressed, or compressed and then encrypted, prior to signing. The content is encoded as an octet string. The eContent octet string need not be DER encoded.

2.1.3. EncryptedData

The EncryptedData content type [CMS] contains the encrypted firmware package (which might be compressed prior to encryption). However, if the firmware package was not encrypted, the EncryptedData content type is not present. The fields of EncryptedData are used as follows:

version is the syntax version number, and in this case, version MUST be 0.

encryptedContentInfo is the encrypted content information. The use of the EncryptedContentInfo type is discussed further in Section 2.1.3.1.

unprotectedAttrs is an optional collection of unencrypted attributes, and in this case, unprotectedAttrs MUST NOT be present.

2.1.3.1. EncryptedContentInfo

The encrypted firmware package, which might be compressed prior to encryption, is encapsulated in the EncryptedContentInfo type. The fields of EncryptedContentInfo are used as follows:

contentType indicates the type of content, and in this case, it MUST contain either id-ct-compressedData (1.2.840.113549.1.9.16.1.9) or id-ct-firmwarePackage (1.2.840.113549.1.9.16.1.16). When it contains id-ct-compressedData, then the firmware package was compressed prior to encryption. When it contains id-ct-firmwarePackage, then the firmware package was not compressed prior to encryption.

contentEncryptionAlgorithm identifies the firmware-encryption algorithm, and any associated parameters, used to encrypt the firmware package.

encryptedContent is the result of encrypting the firmware package. The field is optional; however, in this case, it MUST be present.

2.1.4. CompressedData

The CompressedData content type [COMPRESS] contains the compressed firmware package. If the firmware package was not compressed, then the CompressedData content type is not present. The fields of CompressedData are used as follows:

version is the syntax version number; in this case, it MUST be 0.

compressionAlgorithm identifies the compression algorithm, and any associated parameters, used to compress the firmware package.

encapContentInfo is the compressed content, consisting of a content type identifier and the content itself. The use of the EncapsulatedContentInfo type is discussed further in Section 2.1.4.1.

2.1.4.1. EncapsulatedContentInfo

The CompressedData content type encapsulates the compressed firmware package, and it is carried within the EncapsulatedContentInfo type. The fields of EncapsulatedContentInfo are used as follows:

eContentType is an object identifier that uniquely specifies the content type, and in this case, it MUST be the value of id-ct-firmwarePackage (1.2.840.113549.1.9.16.1.16).

eContent is the compressed firmware package, encoded as an octet string. The eContent octet string need not be DER encoded.

2.1.5. FirmwarePkgData

The FirmwarePkgData content type contains the firmware package. It is a straightforward encapsulation in an octet string, and it need not be DER encoded.

The FirmwarePkgData content type is identified by the id-ct-firmwarePackage object identifier:

```
id-ct-firmwarePackage OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) ct(1) 16 }
```

The FirmwarePkgData content type is a simple octet string:

```
FirmwarePkgData ::= OCTET STRING
```

2.2. Signed Attributes

The firmware package signer MUST digitally sign a collection of attributes along with the firmware package. Each attribute in the collection MUST be DER encoded [X.509-88]. The syntax for attributes is defined in [CMS], but it is repeated here for convenience:

```
Attribute ::= SEQUENCE {
  attrType OBJECT IDENTIFIER,
  attrValues SET OF AttributeValue }

AttributeValue ::= ANY
```

Each of the attributes used with this 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.

The firmware package signer MUST include the following four attributes: content-type, message-digest, firmware-package-identifier, and target-hardware-module-identifiers.

If the firmware package is encrypted, then the firmware package signer MUST also include the decrypt-key-identifier attribute.

If the firmware package implements cryptographic algorithms, then the firmware package signer MAY also include the implemented-crypto-algorithms attribute. Similarly, if the firmware package implements compression algorithms, then the firmware package signer MAY also include the implemented-compress-algorithms attribute.

If the firmware package is intended for use only by specific communities, then the firmware package signer MUST also include the community-identifiers attribute.

If the firmware package depends on the presence of one or more other firmware packages to operate properly, then the firmware package signer SHOULD also include the firmware-package-info attribute. For example, the firmware-package-info attribute dependencies field might indicate that the firmware package contains a dependency on a particular bootstrap loader or separation kernel.

The firmware package signer SHOULD also include the three following attributes: firmware-package-message-digest, signing-time, and content-hints. Additionally, if the firmware package signer has a certificate (meaning that the firmware package signer is not always configured as a trust anchor), then the firmware package signer SHOULD also include the signing-certificate attribute.

The firmware package signer MAY include any other attribute that it deems appropriate.

2.2.1. Content Type

The firmware package signer MUST include a content-type attribute with the value of id-encryptedData (1.2.840.113549.1.7.6), id-ct-compressedData (1.2.840.113549.1.9.16.1.9), or id-ct-firmwarePackage (1.2.840.113549.1.9.16.1.16). When it contains id-encryptedData, the firmware package was encrypted prior to signing. When it contains id-ct-compressedData, the firmware package was compressed prior to signing, but was not encrypted. When it contains

id-ct-firmwarePackage, the firmware package was not compressed or encrypted prior to signing. Section 11.1 of [CMS] defines the content-type attribute.

2.2.2. Message Digest

The firmware package signer MUST include a message-digest attribute, having as its value the message digest computed on the encapContentInfo eContent octet string, as defined in Section 2.1.2.2. This octet string contains the firmware package, and it MAY be compressed, encrypted, or both compressed and encrypted. Section 11.2 of [CMS] defines the message-digest attribute.

2.2.3. Firmware Package Identifier

The firmware-package-identifier attribute names the protected firmware package. Two approaches to naming firmware packages are supported: legacy and preferred. The firmware package signer MUST include a firmware-package-identifier attribute using one of these name forms.

A legacy firmware package name is an octet string, and no structure within the octet string is assumed.

A preferred firmware package name is a combination of an object identifier and a version number. The object identifier names a collection of functions implemented by the firmware package, and the version number is a non-negative integer that identifies a particular build or release of the firmware package.

If a firmware package with a disastrous flaw is released, the firmware package that repairs the previously distributed flaw MAY designate a stale firmware package version to prevent the reloading of the flawed version. The hardware module bootstrap loader SHOULD prevent subsequent rollback to the stale version or versions earlier than the stale version. When the legacy firmware package name form is used, the stale version is indicated by a stale legacy firmware package name, which is an octet string. We assume that the firmware package signer and the bootstrap loader can determine whether a given legacy firmware package name represents a version that is more recent than the stale one. When the preferred firmware package name form is used, the stale version is indicated by a stale version number, which is an integer.

The following object identifier identifies the firmware-package-identifier attribute:

```
id-aa-firmwarePackageID OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 35 }
```

The firmware-package-identifier attribute values have ASN.1 type FirmwarePackageIdentifier:

```
FirmwarePackageIdentifier ::= SEQUENCE {
  name PreferredOrLegacyPackageIdentifier,
  stale PreferredOrLegacyStalePackageIdentifier OPTIONAL }

PreferredOrLegacyPackageIdentifier ::= CHOICE {
  preferred PreferredPackageIdentifier,
  legacy OCTET STRING }

PreferredPackageIdentifier ::= SEQUENCE {
  fwPkgID OBJECT IDENTIFIER,
  verNum INTEGER (0..MAX) }

PreferredOrLegacyStalePackageIdentifier ::= CHOICE {
  preferredStaleVerNum INTEGER (0..MAX),
  legacyStaleVersion OCTET STRING }
```

2.2.4. Target Hardware Module Identifiers

The target-hardware-module-identifiers attribute names the types of hardware modules that the firmware package supports. A unique object identifier names each supported hardware model type and revision.

The bootstrap loader MUST reject the firmware package if its own hardware module type identifier is not listed in the target-hardware-module-identifiers attribute.

The following object identifier identifies the target-hardware-module-identifiers attribute:

```
id-aa-targetHardwareIDs OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 36 }
```

The target-hardware-module-identifiers attribute values have ASN.1 type TargetHardwareIdentifiers:

```
TargetHardwareIdentifiers ::= SEQUENCE OF OBJECT IDENTIFIER
```

2.2.5. Decrypt Key Identifier

The decrypt-key-identifier attribute names the symmetric key needed to decrypt the encapsulated firmware package. The CMS EncryptedData content type is used when the firmware package is encrypted. The decrypt-key-identifier signed attribute is carried in the SignedData content type that encapsulates EncryptedData content type, naming the symmetric key needed to decrypt the firmware package. No particular structure is imposed on the key identifier. The means by which the firmware-decryption key is securely distributed to all modules that are authorized to use the associated firmware package is beyond the scope of this specification; however, an optional mechanism for securely distributing the firmware-decryption key with the firmware package is specified in Section 2.3.1.

The following object identifier identifies the decrypt-key-identifier attribute:

```
id-aa-decryptKeyID OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 37 }
```

The decrypt-key-identifier attribute values have ASN.1 type DecryptKeyIdentifier:

```
DecryptKeyIdentifier ::= OCTET STRING
```

2.2.6. Implemented Crypto Algorithms

The implemented-crypto-algorithms attribute MAY be present in the SignedAttributes, and it names the cryptographic algorithms that are implemented by the firmware package and available to applications. Only those algorithms that are made available at the interface of the cryptographic module are listed. Any cryptographic algorithm that is used internally and is not accessible via the cryptographic module interface MUST NOT be listed. For example, if the firmware package implements the decryption algorithm for future firmware package installations and this algorithm is not made available for other uses, then the firmware-decryption algorithm would not be listed.

The object identifier portion of AlgorithmIdentifier identifies an algorithm and its mode of use. No algorithm parameters are included. Cryptographic algorithms include traffic-encryption algorithms, key-encryption algorithms, key transport algorithms, key agreement algorithms, one-way hash algorithms, and digital signature algorithms. Cryptographic algorithms do not include compression algorithms.

The following object identifier identifies the implemented-crypto-algorithms attribute:

```
id-aa-implCryptoAlgs OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 38 }
```

The implemented-crypto-algorithms attribute values have ASN.1 type ImplementedCryptoAlgorithms:

```
ImplementedCryptoAlgorithms ::= SEQUENCE OF OBJECT IDENTIFIER
```

2.2.7. Implemented Compression Algorithms

The implemented-compress-algorithms attribute MAY be present in the SignedAttributes, and it names the compression algorithms that are implemented by the firmware package and available to applications. Only those algorithms that are made available at the interface of the hardware module are listed. Any compression algorithm that is used internally and is not accessible via the hardware module interface MUST NOT be listed. For example, if the firmware package implements a decompression algorithm for future firmware package installations and this algorithm is not made available for other uses, then the firmware-decompression algorithm would not be listed.

The object identifier portion of AlgorithmIdentifier identifies a compression algorithm. No algorithm parameters are included.

The following object identifier identifies the implemented-compress-algorithms attribute:

```
id-aa-implCompressAlgs OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 43 }
```

The implemented-compress-algorithms attribute values have ASN.1 type ImplementedCompressAlgorithms:

```
ImplementedCompressAlgorithms ::= SEQUENCE OF OBJECT IDENTIFIER
```

2.2.8. Community Identifiers

If present in the SignedAttributes, the community-identifiers attribute names the communities that are permitted to execute the firmware package. The bootstrap loader MUST reject the firmware package if the hardware module is not a member of one of the identified communities. The means of assigning community membership is beyond the scope of this specification.

The community-identifiers attributes names the authorized communities by a list of community object identifiers, by a list of specific hardware modules, or by a combination of the two lists. A specific hardware module is specified by the combination of the hardware module identifier (as defined in Section 2.2.4) and a serial number. To facilitate compact representation of serial numbers, a contiguous block can be specified by the lowest authorized serial number and the highest authorized serial number. Alternatively, all of the serial numbers associated with a hardware module family identifier can be specified with the NULL value.

If the bootstrap loader does not have a mechanism for obtaining a list of object identifiers that identify the communities to which the hardware module is a member, then the bootstrap loader MUST behave as though the list is empty. Similarly, if the bootstrap loader does not have access to the hardware module serial number, then the bootstrap loader MUST behave as though the hardware module is not included on the list of authorized hardware modules.

The following object identifier identifies the community-identifiers attribute:

```
id-aa-communityIdentifiers OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 40 }
```

The community-identifiers attribute values have ASN.1 type CommunityIdentifiers:

```
CommunityIdentifiers ::= SEQUENCE OF CommunityIdentifier

CommunityIdentifier ::= CHOICE {
  communityOID OBJECT IDENTIFIER,
  hwModuleList HardwareModules }

HardwareModules ::= SEQUENCE {
  hwType OBJECT IDENTIFIER,
  hwSerialEntries SEQUENCE OF HardwareSerialEntry }

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

2.2.9. Firmware Package Information

If a hardware module supports more than one type of firmware package, then the firmware package signer SHOULD include the firmware-package-info attribute with a populated fwPkgType field to identify the firmware package type. This value can aid the bootstrap loader in the correct placement of the firmware package within the hardware module. The firmware package type is an INTEGER, and the meaning of the integer value is specific to each hardware module. For example, a hardware module could assign different integer values for a bootstrap loader, a separation kernel, and an application.

Some hardware module architectures permit one firmware package to use routines provided by another. If the firmware package contains a dependency on another, then the firmware package signer SHOULD also include the firmware-package-info attribute with a populated dependencies field. If the firmware package does not depend on any other firmware packages, then the firmware package signer MUST NOT include the firmware-package-info attribute with a populated dependencies field.

Firmware package dependencies are identified by the firmware package identifier or by information contained in the firmware package itself, and in either case the bootstrap loader ensures that the dependencies are met. The bootstrap loader MUST reject a firmware package load if it identifies a dependency on a firmware package that is not already loaded. Also, the bootstrap loader MUST reject a firmware package load if the action will result in a configuration where the dependencies of an already loaded firmware package will no longer be satisfied. As described in Section 2.2.3, two approaches to naming firmware packages are supported: legacy and preferred. When the legacy firmware package name form is used, the dependency is indicated by a legacy firmware package name. We assume that the firmware package signer and the bootstrap loader can determine whether a given legacy firmware package name represents the named version of an acceptable newer version. When the preferred firmware package name form is used, an object identifier and an integer are provided. The object identifier MUST exactly match the object identifier portion of a preferred firmware package name associated with a firmware package that is already loaded, and the integer MUST be less than or equal to the integer portion of the preferred firmware package name associated with the same firmware package. That is, the dependency specifies the minimum value of the version that is acceptable.

The following object identifier identifies the firmware-package-info attribute:

```
id-aa-firmwarePackageInfo OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 42 }
```

The firmware-package-info attribute values have ASN.1 type FirmwarePackageInfo:

```
FirmwarePackageInfo ::= SEQUENCE {
  fwPkgType INTEGER OPTIONAL,
  dependencies SEQUENCE OF
    PreferredOrLegacyPackageIdentifier OPTIONAL }
```

2.2.10. Firmware Package Message Digest

The firmware package signer SHOULD include a firmware-package-message-digest attribute, which provides the message digest algorithm and the message digest value computed on the firmware package. The message digest is computed on the firmware package prior to any compression, encryption, or signature processing. The bootstrap loader MAY use this message digest to confirm that the intended firmware package has been recovered after all of the layers of encapsulation are removed.

The following object identifier identifies the firmware-package-message-digest attribute:

```
id-aa-fwPkgMessageDigest OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 41 }
```

The firmware-package-message-digest attribute values have ASN.1 type FirmwarePackageMessageDigest:

```
FirmwarePackageMessageDigest ::= SEQUENCE {
  algorithm AlgorithmIdentifier,
  msgDigest OCTET STRING }
```

2.2.11. Signing Time

The firmware package signer SHOULD include a signing-time attribute, specifying the time at which the signature was applied to the firmware package. Section 11.3 of [CMS] defines the signing-time attribute.

2.2.12. Content Hints

The firmware package signer SHOULD include a content-hints attribute, including a brief text description of the firmware package. The text is encoded in UTF-8, which supports most of the world's writing systems [UTF-8]. Section 2.9 of [ESS] defines the content-hints attribute.

When multiple layers of encapsulation are employed, the content-hints attribute is included in the outermost SignedData to provide information about the innermost content. In this case, the content-hints attribute provides a brief text description of the firmware package, which can help a person select the correct firmware package when more than one is available.

When the preferred firmware package name forms are used, the content-hints attribute can provide a linkage to a legacy firmware package name. This is especially helpful when an existing configuration management system is in use, but the features associated with the preferred firmware package name are deemed useful. A firmware package name associated with such a configuration management system might look something like "R1234.C0(AJ11).D62.A02.11(b)." Including these firmware package names in the text description may be helpful to developers by providing a clear linkage between the two name forms.

The content-hints attribute contains two fields, and in this case, both fields MUST be present. The fields of ContentHints are used as follows:

contentDescription provides a brief text description of the firmware package.

contentType provides the content type of the inner most content type, and in this case, it MUST be id-ct-firmwarePackage (1.2.840.113549.1.9.16.1.16).

2.2.13. Signing Certificate

When the firmware-signer's public key is contained in a certificate, the firmware package signer SHOULD include a signing-certificate attribute to identify the certificate that was employed. However, if the firmware package signature does not have a certificate (meaning that the signature will only be validated with the trust anchor public key), then the firmware package signer is unable to include a signing-certificate attribute. Section 5.4 of [ESS] defines this attribute.

The signing-certificate attribute contains two fields: certs and policies. The certs field MUST be present, and the policies field MAY be present. The fields of SigningCertificate are used as follows:

certs contains a sequence of certificate identifiers. In this case, sequence of certificate identifiers contains a single entry. The certs field MUST contain only the certificate identifier of the certificate that contains the public key used to verify the firmware package signature. The certs field uses the ESSCertID syntax specified in Section 5.4 of [ESS], and it is comprised of the SHA-1 hash [SHA1] of the entire ASN.1 DER encoded certificate and, optionally, the certificate issuer and the certificate serial number. The SHA-1 hash value MUST be present. The certificate issuer and the certificate serial number SHOULD be present.

policies is optional; when it is present, it contains a sequence of policy information. The policies field, when present, MUST contain only one entry, and that entry MUST match one of the certificate policies in the certificate policies extension of the certificate that contains the public key used to verify the firmware package signature. The policies field uses the PolicyInformation syntax specified in Section 4.2.1.5 of [PROFILE], and it is comprised of the certificate policy object identifier and, optionally, certificate policy qualifiers. The certificate policy object identifier MUST be present. The certificate policy qualifiers SHOULD NOT be present.

2.3. Unsigned Attributes

CMS allows a SET of unsigned attributes to be included; however, in this specification, the set MUST be absent or include a single instance of the wrapped-firmware-decryption-key attribute. Because the digital signature does not cover this attribute, it can be altered at any point in the delivery path from the firmware package signer to the hardware module. This property can be employed to distribute the firmware-decryption key along with an encrypted and signed firmware package, allowing the firmware-decryption key to be wrapped with a different key-encryption key for each link in the distribution chain.

The syntax for attributes is defined in [CMS], and it is repeated at the beginning of Section 2.2 of this document for convenience. Each of the attributes used with this 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 UnsignedAttributes syntax within signerInfo is defined as a SET OF Attribute. The UnsignedAttributes MUST include only one instance of any particular attribute.

2.3.1. Wrapped Firmware Decryption Key

The firmware package signer, or any other party in the distribution chain, MAY include a wrapped-firmware-decryption-key attribute.

The following object identifier identifies the wrapped-firmware-decryption-key attribute:

```
id-aa-wrappedFirmwareKey OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 39 }
```

The wrapped-firmware-decryption-key attribute values have ASN.1 type of EnvelopedData. Section 6 of [CMS] defines the EnvelopedData content type, which is used to construct the value of the attribute. EnvelopedData permits the firmware-decryption key to be protected using symmetric or asymmetric techniques. The EnvelopedData does not include an encrypted content; rather, the EnvelopedData feature of having the encrypted content in another location is employed. The encrypted content is found in the eContent field of the EncryptedData structure. The firmware-decryption key is contained in the recipientInfos field. Section 6 of [CMS] refers to this key as the content-encryption key.

The EnvelopedData syntax supports many different key management algorithms. Four general techniques are supported: key transport, key agreement, symmetric key-encryption keys, and passwords.

The EnvelopedData content type is profiled for the wrapped-firmware-decryption-key attribute. The EnvelopedData fields are described fully in Section 6 of [CMS]. Additional rules apply when EnvelopedData is used as a wrapped-firmware-decryption-key attribute.

Within the EnvelopedData structure, the following apply:

- The set of certificates included in OriginatorInfo MUST NOT include certificates with a type of extendedCertificate, v1AttrCert, or v2AttrCert [X.509-97, X.509-00, ACPROFILE]. The optional crls field MAY be present.

- The optional unprotectedAttrs field MUST NOT be present.

Within the EncryptedContentInfo structure, the following apply:

- contentType MUST match the content type object identifier carried in the contentType field within the EncryptedContentInfo structure of EncryptedData as described in Section 2.1.3.1.

- contentEncryptionAlgorithm identifies the firmware-encryption algorithm, and any associated parameters, used to encrypt the firmware package carried in the encryptedContent field of the EncryptedContentInfo structure of EncryptedData. Therefore, it MUST exactly match the value of the EncryptedContentInfo structure of EncryptedData as described in Section 2.1.3.1.

- encryptedContent is optional, and in this case, it MUST NOT be present.

3. Firmware Package Load Receipt

The Cryptographic Message Syntax (CMS) is used to indicate that a firmware package loaded successfully. Support for firmware package load receipts is OPTIONAL. However, those hardware modules that choose to generate such receipts MUST follow the conventions specified in this section. Because not all hardware modules will have private signature keys, the firmware package load receipt can be either signed or unsigned. Use of the signed firmware package load receipt is RECOMMENDED.

Hardware modules that support receipt generation MUST have a unique serial number. Hardware modules that support signed receipt generation MUST have a private signature key to sign the receipt and the corresponding signature validation certificate or its designator. The designator is the certificate issuer name and the certificate serial number, or it is the public key identifier. Memory-constrained hardware modules will generally store the public key identifier since it requires less storage.

The unsigned firmware package load receipt is encapsulated by ContentInfo. Alternatively, the signed firmware package load receipt is encapsulated by SignedData, which is in turn encapsulated by ContentInfo.

The firmware package load receipt is summarized as follows (see [CMS] for the full syntax):

```
ContentInfo {
  contentType          id-signedData, -- (1.2.840.113549.1.7.2)
                       -- OR --
                       id-ct-firmwareLoadReceipt,
                            -- (1.2.840.113549.1.9.16.1.17)
  content              SignedData
                       -- OR --
                       FirmwarePackageLoadReceipt
}

SignedData {
  version              CMSVersion, -- always set to 3
  digestAlgorithms     DigestAlgorithmIdentifiers, -- Only one
  encapContentInfo     EncapsulatedContentInfo,
  certificates         CertificateSet, -- Optional Module certificate
  crls                 CertificateRevocationLists, -- Optional
  signerInfos          SET OF SignerInfo -- Only one
}

SignerInfo {
  version              CMSVersion, -- either set to 1 or 3
  sid                  SignerIdentifier,
  digestAlgorithm      DigestAlgorithmIdentifier,
  signedAttrs          SignedAttributes, -- Required
  signatureAlgorithm   SignatureAlgorithmIdentifier,
  signature            SignatureValue,
  unsignedAttrs        UnsignedAttributes -- Omit
}

EncapsulatedContentInfo {
  eContentType         id-ct-firmwareLoadReceipt,
                            -- (1.2.840.113549.1.9.16.1.17)
  eContent             OCTET STRING -- Contains receipt
}

FirmwarePackageLoadReceipt {
  version              INTEGER, -- The DEFAULT is always used
  hwType               OBJECT IDENTIFIER, -- Hardware module type
  hwSerialNum          OCTET STRING, -- H/W module serial number
  fwPkgName            PreferredOrLegacyPackageIdentifier,
  trustAnchorKeyID     OCTET STRING, -- Optional
  decryptKeyID         OCTET STRING -- Optional
}
```

3.1. Firmware Package Load Receipt CMS Content Type Profile

This section specifies the conventions for using the CMS ContentInfo and SignedData content types for firmware package load receipts. It also defines the firmware package load receipt content type.

3.1.1. ContentInfo

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

contentType indicates the type of the associated content. If the firmware package load receipt is signed, then the encapsulated type MUST be SignedData, and the id-signedData (1.2.840.113549.1.7.2) object identifier MUST be present in this field. If the receipt is not signed, then the encapsulated type MUST be FirmwarePackageLoadReceipt, and the id-ct-firmwareLoadReceipt (1.2.840.113549.1.9.16.1.17) object identifier MUST be present in this field.

content holds the associated content. If the firmware package load receipt is signed, then this field MUST contain the SignedData. If the receipt is not signed, then this field MUST contain the FirmwarePackageLoadReceipt.

3.1.2. SignedData

The SignedData content type contains the firmware package load receipt and one digital signature. If the hardware module locally stores its certificate, then the certificate can be included as well. The fields of SignedData are used as follows:

version is the syntax version number, and in this case, it MUST be set to 3.

digestAlgorithms is a collection of message digest algorithm identifiers, and in this case, it MUST contain a single message digest algorithm identifier. The message digest algorithms employed by the hardware module 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 3.1.2.2.

certificates is an optional collection of certificates. If the hardware module locally stores its certificate, then the X.509 certificate of the hardware module SHOULD be included. If the

hardware module does not, then the certificates field is omitted. PKCS#6 extended certificates [PKCS#6] and attribute certificates (either version 1 or version 2) [X.509-97, X.509-00, ACPROFILE] MUST NOT be included in the set of certificates.

crls is an optional collection of certificate revocation lists (CRLs). CRLs MAY be included, but they will normally be omitted since hardware modules will not generally have access to the most recent CRL. Signed receipt recipients SHOULD be able to handle the presence of the optional crls field.

signerInfos is a collection of per-signer information, and in this case, the collection MUST contain exactly one SignerInfo. The use of the SignerInfo type is discussed further in Section 3.1.2.1.

3.1.2.1. SignerInfo

The hardware module is represented in the SignerInfo type. The fields of SignerInfo are used as follows:

version is the syntax version number, and it MUST be either 1 or 3, depending on the method used to identify the hardware module's public key. The use of the subjectKeyIdentifier is RECOMMENDED, which results in the use of version 3.

sid specifies the hardware module's certificate (and thereby the hardware module's public key). CMS supports two alternatives: issuerAndSerialNumber and subjectKeyIdentifier. The hardware module MUST support one or both of the alternatives for receipt generation; however, the support of subjectKeyIdentifier is RECOMMENDED. The issuerAndSerialNumber alternative identifies the hardware module's certificate by the issuer's distinguished name and the certificate serial number. The identified certificate, in turn, contains the hardware module's public key. The subjectKeyIdentifier alternative identifies the hardware module's public key directly. When this public key is contained in a certificate, this identifier SHOULD appear in the X.509 subjectKeyIdentifier extension.

digestAlgorithm identifies the message digest algorithm, and any associated parameters, used by the hardware module. It MUST contain the message digest algorithms employed to sign the receipt. (Note that this message digest algorithm identifier MUST be the same as the one carried in the digestAlgorithms value in SignedData.)

signedAttrs is an optional collection of attributes that are signed along with the content. The signedAttrs are optional in the CMS, but in this specification, signedAttrs are REQUIRED for use with the firmware package load receipt content. The SET OF attributes MUST be DER encoded [X.509-88]. Section 3.2 of this document lists the attributes that MUST be included in the collection. Other attributes MAY be included, but the recipient will ignore any unrecognized signed attributes.

signatureAlgorithm identifies the signature algorithm, and any associated parameters, used to sign the receipt.

signature is the digital signature.

unsignedAttrs is an optional collection of attributes that are not signed, and in this case, there MUST NOT be any unsigned attributes present.

3.1.2.2. EncapsulatedContentInfo

The FirmwarePackageLoadReceipt is encapsulated in an OCTET STRING, and it is carried within the EncapsulatedContentInfo type. The fields of EncapsulatedContentInfo are used as follows:

eContentType is an object identifier that uniquely specifies the content type, and in this case, it MUST be the value of id-ct-firmwareLoadReceipt (1.2.840.113549.1.9.16.1.17).

eContent is the firmware package load receipt, encapsulated in an OCTET STRING. The eContent octet string need not be DER encoded.

3.1.3. FirmwarePackageLoadReceipt

The following object identifier identifies the firmware package load receipt content type:

```
id-ct-firmwareLoadReceipt OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) ct(1) 17 }
```

The firmware package load receipt content type has the ASN.1 type FirmwarePackageLoadReceipt:

```
FirmwarePackageLoadReceipt ::= SEQUENCE {
  version FWReceiptVersion DEFAULT v1,
  hwType OBJECT IDENTIFIER,
  hwSerialNum OCTET STRING,
  fwPkgName PreferredOrLegacyPackageIdentifier,
  trustAnchorKeyID OCTET STRING OPTIONAL,
  decryptKeyID [1] OCTET STRING OPTIONAL }

FWReceiptVersion ::= INTEGER { v1(1) }
```

The fields of the FirmwarePackageLoadReceipt type have the following meanings:

version is an integer that provides the syntax version number for compatibility with future revisions of this specification. Implementations that conform to this specification MUST set the version to the default value, which is v1.

hwType is an object identifier that identifies the type of hardware module on which the firmware package was loaded.

hwSerialNum is the serial number of the hardware module on which the firmware package was loaded. No particular structure is imposed on the serial number; it need not be an integer. However, the combination of the hwType and hwSerialNum uniquely identifies the hardware module.

fwPkgName identifies the firmware package that was loaded. As described in Section 2.2.3, two approaches to naming firmware packages are supported: legacy and preferred. A legacy firmware package name is an octet string. A preferred firmware package name is a combination of the firmware package object identifier and an integer version number.

trustAnchorKeyID is optional, and when it is present, it identifies the trust anchor that was used to validate the firmware package signature.

decryptKeyID is optional, and when it is present, it identifies the firmware-decryption key that was used to decrypt the firmware package.

The firmware package load receipt MUST include the version, hwType, hwSerialNum, and fwPkgName fields, and it SHOULD include the trustAnchorKeyID field. The firmware package load receipt MUST NOT

include the decryptKeyID, unless the firmware package associated with the receipt is encrypted, the firmware-decryption key is available to the hardware module, and the firmware package was successfully decrypted.

3.2. Signed Attributes

The hardware module MUST digitally sign a collection of attributes along with the firmware package load receipt. Each attribute in the collection MUST be DER encoded [X.509-88]. The syntax for attributes is defined in [CMS], and it was repeated in Section 2.2 for convenience.

Each of the attributes used with this 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 Attributes. The SignedAttributes MUST include only one instance of any particular attribute.

The hardware module MUST include the content-type and message-digest attributes. If the hardware module includes a real-time clock, then the hardware module SHOULD also include the signing-time attribute. The hardware module MAY include any other attribute that it deems appropriate.

3.2.1. Content Type

The hardware module MUST include a content-type attribute with the value of id-ct-firmwareLoadReceipt (1.2.840.113549.1.9.16.1.17). Section 11.1 of [CMS] defines the content-type attribute.

3.2.2. Message Digest

The hardware module MUST include a message-digest attribute, having as its value the message digest of the FirmwarePackageLoadReceipt content. Section 11.2 of [CMS] defines the message-digest attribute.

3.2.3. Signing Time

If the hardware module includes a real-time clock, then the hardware module SHOULD include a signing-time attribute, specifying the time at which the receipt was generated. Section 11.3 of [CMS] defines the signing-time attribute.

4. Firmware Package Load Error

The Cryptographic Message Syntax (CMS) is used to indicate that an error has occurred while attempting to load a protected firmware package. Support for firmware package load error reports is OPTIONAL. However, those hardware modules that choose to generate such error reports MUST follow the conventions specified in this section. Not all hardware modules have private signature keys; therefore the firmware package load error report can be either signed or unsigned. Use of the signed firmware package error report is RECOMMENDED.

Hardware modules that support error report generation MUST have a unique serial number. Hardware modules that support signed error report generation MUST also have a private signature key to sign the error report and the corresponding signature validation certificate or its designator. The designator is the certificate issuer name and the certificate serial number, or it is the public key identifier. Memory-constrained hardware modules will generally store the public key identifier since it requires less storage.

The unsigned firmware package load error report is encapsulated by ContentInfo. Alternatively, the signed firmware package load error report is encapsulated by SignedData, which is in turn encapsulated by ContentInfo.

The firmware package load error report is summarized as follows (see [CMS] for the full syntax):

```
ContentInfo {
  contentType          id-signedData, -- (1.2.840.113549.1.7.2)
                       -- OR --
                       id-ct-firmwareLoadError,
                            -- (1.2.840.113549.1.9.16.1.18)
  content              SignedData
                       -- OR --
                       FirmwarePackageLoadError
}

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

```
SignerInfo {
  version               CMSVersion, -- either set to 1 or 3
  sid                   SignerIdentifier,
  digestAlgorithm       DigestAlgorithmIdentifier,
  signedAttrs           SignedAttributes, -- Required
  signatureAlgorithm    SignatureAlgorithmIdentifier,
  signature             SignatureValue,
  unsignedAttrs         UnsignedAttributes -- Omit
}

EncapsulatedContentInfo {
  eContentType          id-ct-firmwareLoadError,
                             -- (1.2.840.113549.1.9.16.1.18)
  eContent              OCTET STRING -- Contains error report
}

FirmwarePackageLoadError {
  version             INTEGER, -- The DEFAULT is always used
  hwType              OBJECT IDENTIFIER, -- Hardware module type
  hwSerialNum         OCTET STRING, -- H/W module serial number
  errorCode           FirmwarePackageLoadErrorCode -- Error identifier
  vendorErrorCode     VendorErrorCode, -- Optional
  fwPkgName           PreferredOrLegacyPackageIdentifier, -- Optional
  config              SEQUENCE OF CurrentFWConfig, -- Optional
}

CurrentFWConfig {       -- Repeated for each package in configuration
  fwPkgType             INTEGER, -- Firmware package type; Optional
  fwPkgName             PreferredOrLegacyPackageIdentifier
}
```

4.1. Firmware Package Load Error CMS Content Type Profile

This section specifies the conventions for using the CMS ContentInfo and SignedData content types for firmware package load error reports. It also defines the firmware package load error content type.

4.1.1. ContentInfo

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

contentType indicates the type of the associated content. If the firmware package load error report is signed, then the encapsulated type MUST be SignedData, and the id-signedData (1.2.840.113549.1.7.2) object identifier MUST be present in this field. If the report is not signed, then the encapsulated type

MUST be FirmwarePackageLoadError, and the id-ct-firmwareLoadError (1.2.840.113549.1.9.16.1.18) object identifier MUST be present in this field.

content holds the associated content. If the firmware package load error report is signed, then this field MUST contain the SignedData. If the report is not signed, then this field MUST contain the FirmwarePackageLoadError.

4.1.2. SignedData

The SignedData content type contains the firmware package load error report and one digital signature. If the hardware module locally stores its certificate, then the certificate can be included as well. The fields of SignedData are used exactly as described in Section 3.1.2.

4.1.2.1. SignerInfo

The hardware module is represented in the SignerInfo type. The fields of SignerInfo are used exactly as described in Section 3.1.2.1.

4.1.2.2. EncapsulatedContentInfo

The FirmwarePackageLoadError is encapsulated in an OCTET STRING, and it is carried within the EncapsulatedContentInfo type. The fields of EncapsulatedContentInfo are used as follows:

eContentType is an object identifier that uniquely specifies the content type, and in this case, it MUST be the value of id-ct-firmwareLoadError (1.2.840.113549.1.9.16.1.18).

eContent is the firmware package load error report, encapsulated in an OCTET STRING. The eContent octet string need not be DER encoded.

4.1.3. FirmwarePackageLoadError

The following object identifier identifies the firmware package load error report content type:

```
id-ct-firmwareLoadError OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) ct(1) 18 }
```

The firmware package load error report content type has the ASN.1 type FirmwarePackageLoadError:

```
FirmwarePackageLoadError ::= SEQUENCE {
  version FWErrorVersion DEFAULT v1,
  hwType OBJECT IDENTIFIER,
  hwSerialNum OCTET STRING,
  errorCode FirmwarePackageLoadErrorCode,
  vendorErrorCode VendorLoadErrorCode OPTIONAL,
  fwPkgName PreferredOrLegacyPackageIdentifier OPTIONAL,
  config [1] SEQUENCE OF CurrentFWConfig OPTIONAL }

FWErrorVersion ::= INTEGER { v1(1) }

CurrentFWConfig ::= SEQUENCE {
  fwPkgType INTEGER OPTIONAL,
  fwPkgName PreferredOrLegacyPackageIdentifier }

FirmwarePackageLoadErrorCode ::= ENUMERATED {
  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),
  signatureFailure            (15),
  contentTypeMismatch         (16),
  badEncryptedData            (17),
  unprotectedAttrsPresent     (18),
  badEncryptContent           (19),
  badEncryptAlgorithm         (20),
  missingCiphertext           (21),
  noDecryptKey                (22),
  decryptFailure              (23),
  badCompressAlgorithm        (24),
  missingCompressedContent    (25),
  decompressFailure           (26),
  wrongHardware               (27),
  stalePackage                (28),
  notInCommunity              (29),
```

```
      unsupportedPackageType      (30),
      missingDependency           (31),
      wrongDependencyVersion      (32),
      insufficientMemory          (33),
      badFirmware                 (34),
      unsupportedParameters       (35),
      breaksDependency            (36),
      otherError                  (99) }

   VendorLoadErrorCode ::= INTEGER
```

The fields of the FirmwarePackageLoadError type have the following meanings:

version is an integer, and it provides the syntax version number for compatibility with future revisions of this specification. Implementations that conform to this specification MUST set the version to the default value, which is v1.

hwType is an object identifier that identifies the type of hardware module on which the firmware package load was attempted.

hwSerialNum is the serial number of the hardware module on which the firmware package load was attempted. No particular structure is imposed on the serial number; it need not be an integer. However, the combination of the hwType and hwSerialNum uniquely identifies the hardware module.

errorCode identifies the error that occurred.

vendorErrorCode is optional; however, it MUST be present if the errorCode contains a value of otherError. When errorCode contains a value other than otherError, the vendorErrorCode can provide vendor-specific supplemental information.

fwPkgName is optional. When it is present, it identifies the firmware package that was being loaded when the error occurred. As described in Section 2.2.3, two approaches to naming firmware packages are supported: legacy and preferred. A legacy firmware package name is an octet string. A preferred firmware package name is a combination of the firmware package object identifier and an integer version number.

config identifies the current firmware configuration. The field is OPTIONAL, but support for this field is RECOMMENDED for hardware modules that permit the loading of more than one firmware package. One instance of CurrentFWConfig is used to provide information about each firmware package in hardware module.

The fields of the CurrentFWConfig type have the following meanings:

fwPkgType identifies the firmware package type. The firmware package type is an INTEGER, and the meaning of the integer value is specific to each hardware module.

fwPkgName identifies the firmware package. As described in Section 2.2.3, two approaches to naming firmware packages are supported: legacy and preferred. A legacy firmware package name is an octet string. A preferred firmware package name is a combination of the firmware package object identifier and an integer version number.

The errorCode values have the following meanings:

decodeFailure: The ASN.1 decode of the firmware package load failed. The provided input did not conform to BER, or it was not ASN.1 at all.

badContentInfo: Invalid ContentInfo syntax, or the contentType carried within the ContentInfo is unknown or unsupported.

badSignedData: Invalid SignedData syntax, the version is unknown or unsupported, or more than one entry is present in digestAlgorithms.

badEncapContent: Invalid EncapsulatedContentInfo syntax, or the contentType carried within the eContentType is unknown or unsupported. This error can be generated due to problems located in SignedData or CompressedData.

badCertificate: Invalid syntax for one or more certificates in CertificateSet.

badSignerInfo: Invalid SignerInfo syntax, or the version is unknown or unsupported.

badSignedAttrs: Invalid signedAttrs syntax within SignerInfo.

badUnsignedAttrs: The unsignedAttrs within SignerInfo contains an attribute other than the wrapped-firmware-decryption-key attribute, which is the only unsigned attribute supported by this specification.

missingContent: The optional eContent is missing in EncapsulatedContentInfo, which is required in this specification. This error can be generated due to problems located in SignedData or CompressedData.

noTrustAnchor: Two situations can lead to this error. 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 does not identify the public key of a trust anchor or a certification path that terminates with an installed trust anchor.

notAuthorized: The sid within SignerInfo leads to an installed trust anchor, but that trust anchor is not an authorized firmware package signer.

badDigestAlgorithm: The digestAlgorithm in either SignerInfo or SignedData is unknown or unsupported.

badSignatureAlgorithm: The signatureAlgorithm in SignerInfo is unknown or unsupported.

unsupportedKeySize: The signatureAlgorithm in SignerInfo is known and supported, but the firmware package signature could not be validated because an unsupported key size was employed by the signer.

signatureFailure: The signatureAlgorithm in SignerInfo is known and supported, but the signature in signature in SignerInfo could not be validated.

contentTypeMismatch: The contentType carried within the eContentType does not match the content type carried in the signed attribute.

badEncryptedData: Invalid EncryptedData syntax; the version is unknown or unsupported.

unprotectedAttrsPresent: EncryptedData contains unprotectedAttrs, which are not permitted in this specification.

badEncryptContent: Invalid EncryptedContentInfo syntax, or the contentType carried within the contentType is unknown or unsupported.

badEncryptAlgorithm: The firmware-encryption algorithm identified by contentEncryptionAlgorithm in EncryptedContentInfo is unknown or unsupported.

missingCiphertext: The optional encryptedContent is missing in EncryptedContentInfo, which is required in this specification.

noDecryptKey: The hardware module does not have the firmware-decryption key named in the decrypt key identifier signed attribute.

decryptFailure: The firmware package did not decrypt properly.

badCompressAlgorithm: The compression algorithm identified by compressionAlgorithm in CompressedData is unknown or unsupported.

missingCompressedContent: The optional eContent is missing in EncapsulatedContentInfo, which is required in this specification.

decompressFailure: The firmware package did not decompress properly.

wrongHardware: The processing hardware module is not listed in the target hardware module identifiers signed attribute.

stalePackage: The firmware package is rejected because it is stale.

notInCommunity: The hardware module is not a member of the community described in the community identifiers signed attribute.

unsupportedPackageType: The firmware package type identified in the firmware package information signed attribute is not supported by the combination of the hardware module and the bootstrap loader.

missingDependency: The firmware package being loaded depends on routines that are part of another firmware package, but that firmware package is not available.

wrongDependencyVersion: The firmware package being loaded depends on routines that are part of the another firmware package, and the available version of that package has an older version number than is required. The available firmware package does not fulfill the dependencies.

insufficientMemory: The firmware package could not be loaded because the hardware module did not have sufficient memory.

badFirmware: The signature on the firmware package was validated, but the firmware package itself was not in an acceptable format. The details will be specific to each hardware module. For example, a hardware module that is composed of multiple firmware-programmable components could not find the internal tagging within the firmware package to distribute executable code to each of the components.

unsupportedParameters: The signature on the firmware package could not be validated because the signer used signature algorithm parameters that are not supported by the hardware module signature verification routines.

breaksDependency: Another firmware package has a dependency that can no longer be satisfied if the firmware package being loaded is accepted.

otherError: An error occurred that does not fit any of the previous error codes.

4.2. Signed Attributes

The hardware module MUST digitally sign a collection of attributes along with the firmware package load error report. Each attribute in the collection MUST be DER encoded [X.509-88]. The syntax for attributes is defined in [CMS], and it was repeated in Section 2.2 for convenience.

Each of the attributes used with this 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 Attributes. The SignedAttributes MUST include only one instance of any particular attribute.

The hardware module MUST include the content-type and message-digest attributes. If the hardware module includes a real-time clock, then the hardware module SHOULD also include the signing-time attribute. The hardware module MAY include any other attribute that it deems appropriate.

4.2.1. Content Type

The hardware module MUST include a content-type attribute with the value of id-ct-firmwareLoadError (1.2.840.113549.1.9.16.1.18). Section 11.1 of [CMS] defines the content-type attribute.

4.2.2. Message Digest

The hardware module MUST include a message-digest attribute, having as its value the message digest of the FirmwarePackageLoadError content. Section 11.2 of [CMS] defines the message-digest attribute.

4.2.3. Signing Time

If the hardware module includes a real-time clock, then hardware module SHOULD include a signing-time attribute, specifying the time at which the firmware package load error report was generated. Section 11.3 of [CMS] defines the signing-time attribute.

5. Hardware Module Name

Support for firmware package load receipts, as discussed in Section 3, is OPTIONAL, and support for the firmware package load error reports, as discussed in Section 4, is OPTIONAL. Hardware modules that support receipt or error report generation MUST have unique serial numbers. Further, hardware modules that support signed receipt or error report generation MUST have private signature keys and corresponding signature validation certificates [PROFILE] or their designators. The conventions for hardware module naming in the signature validation certificates are specified in this section.

The hardware module vendor or a trusted third party MUST issue the signature validation certificate prior to deployment of the hardware module. The certificate is likely to be issued at the time of manufacture. The subject alternative name in this certificate identifies the hardware module. The subject distinguished name is empty, but a critical subject alternative name extension contains the hardware module name, using the otherName choice within the GeneralName structure.

The hardware module name form is identified by the id-on-hardwareModuleName object identifier:

```
id-on-hardwareModuleName OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) dod(6) internet(1) security(5)
  mechanisms(5) pkix(7) on(8) 4 }
```

A HardwareModuleName is composed of an object identifier and an octet string:

```
HardwareModuleName ::= SEQUENCE {
  hwType OBJECT IDENTIFIER,
  hwSerialNum OCTET STRING }
```

The fields of the HardwareModuleName type have the following meanings:

hwType is an object identifier that identifies the type of hardware module. A unique object identifier names a hardware model and revision.

hwSerialNum is the serial number of the hardware module. No particular structure is imposed on the serial number; it need not be an integer. However, the combination of the hwType and hwSerialNum uniquely identifies the hardware module.

6. Security Considerations

This document describes the use of the Cryptographic Message Syntax (CMS) to protect firmware packages; therefore, the security considerations discussed in [CMS] apply to this specification as well.

The conventions specified in this document raise a few security considerations of their own.

6.1. Cryptographic Keys and Algorithms

Private signature keys must be protected. Compromise of the private key used to sign firmware packages permits unauthorized parties to generate firmware packages that are acceptable to hardware modules. Compromise of the hardware module private key allows unauthorized parties to generate signed firmware package load receipts and error reports.

The firmware-decryption key must be protected. Compromise of the key may result in the disclosure of the firmware package to unauthorized parties.

Cryptographic algorithms become weaker with time. As new cryptanalysis techniques are developed and computing performance improves, the work factor to break a particular cryptographic algorithm will be reduced. The ability to change the firmware package provides an opportunity to update or replace cryptographic algorithms. Although this capability is desirable, cryptographic algorithm replacement can lead to interoperability failures. Therefore, the rollout of new cryptographic algorithms must be managed. Generally, the previous generation of cryptographic algorithms and their replacements need to be supported at the same time in order to facilitate an orderly transition.

6.2. Random Number Generation

When firmware packages are encrypted, the source of the firmware package must randomly generate firmware-encryption keys. Also, the generation of public/private signature key pairs relies on a random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic keys can result in little or no security. An attacker may find it much easier to reproduce the PRNG

environment that produced the keys, searching the resulting small set of possibilities, rather than brute-force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RANDOM] offers important guidance in this area.

6.3. Stale Firmware Package Version Number

The firmware signer determines whether a stale version number is included. The policy of the firmware signer needs to consider many factors. Consider the flaw found by Ian Goldberg and David Wagner in the random number generator of the Netscape browser in 1996 [DDJ]. This flaw completely undermines confidentiality protection. A firmware signer might use the stale version number to ensure that upgraded hardware modules do not resume use of the flawed firmware. However, another firmware signer may not consider this an appropriate situation to employ the stale version number, preferring to delegate this decision to someone closer to the operation of the hardware module. Such a person is likely to be in a better position to evaluate whether other bugs introduced in the newer firmware package impose worse operational concerns than the confidentiality concern caused by the flawed random number generator. For example, a user who never uses the encryption feature of the flawed Netscape browser will determine the most appropriate version to use without considering the random number flaw or its fix.

The stale version number is especially useful when the security interests of the person choosing which firmware package version to load into a particular hardware module do not align with the security interests of the firmware package signer. For example, stale version numbers may be useful in hardware modules that provide digital rights management (DRM). Also, stale version numbers will be useful when the deployment organization (as opposed to the firmware package vendor) is the firmware signer. Further, stale version numbers will be useful for firmware packages that need to be trusted to implement organizational (as opposed to the deployment organization) security policy, regardless of whether the firmware signer is the deployment organization or the vendor. For example, hardware devices employed by the military will probably make use of stale version numbers.

The use of a stale version number in a firmware package that employs the preferred firmware package name form cannot completely prevent subsequent use of the stale firmware package. Despite this shortcoming, the feature is included since it is useful in some important situations. By loading different types of firmware packages, each with its own stale firmware package version number until the internal storage for the stale version number is exceeded, the user can circumvent the mechanism. Consider a hardware module

that has storage for two stale version numbers. Suppose that FWPKG-A version 3 is loaded, indicating that FWPKG-A version 2 is stale. The user can sequentially load the following:

- FWPKG-B version 8, indicating that FWPKG-B version 4 is stale.
  (Note: The internal storage indicates that FWPKG-A version 2 and FWPKG-B version 4 are stale.)

- FWPKG-C version 5, indicating that FWPKG-C version 3 is stale.
  (Note: The internal storage indicates that FWPKG-B version 4 and FWPKG-C version 3 are stale.)

- FWPKG-A version 2.

Because many hardware modules are expected to have very few firmware packages written for them, the stale firmware package version feature provides important protections. The amount of non-volatile storage that needs to be dedicated to saving firmware package identifiers and version numbers depends on the number of firmware packages that are likely to be developed for the hardware module.

The use of legacy firmware package name form does not improve this situation. In fact, the legacy firmware package names are usually larger than an object identifier. Thus, comparable stale version protection requires more memory.

A firmware signer can ensure that stale version numbers are honored by limiting the number of different types of firmware packages that are signed. If all of the hardware modules are able to store a stale version number for each of the different types of firmware package, then the hardware module will be able to provide the desired protection. This requires the firmware signer to have a deep understanding of all of the hardware modules that might accept the firmware package.

6.4. Community Identifiers

When a firmware package includes a community identifier, the confidence that the package is only used by the intended community depends on the mechanism used to configure community membership. This document does not specify a mechanism for the assignment of community membership to hardware modules, and the various alternatives have different security properties. Also, the authority that makes community identifier assignments to hardware modules might be different than the authority that generates firmware packages.

7. References

7.1. Normative References

[COMPRESS] Gutmann, P., "Compressed Data Content Type for Cryptographic Message Syntax (CMS)", RFC 3274, June 2002.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[ESS] Hoffman, P., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[PROFILE] 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.

[SHA1] National Institute of Standards and Technology. FIPS Pub 180-1: Secure Hash Standard. 17 April 1995.

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

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[X.208-88] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

[X.209-88] CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). 1988.

[X.509-88] CCITT. Recommendation X.509: The Directory - Authentication Framework. 1988.

7.2. Informative References

[ACPROFILE] Farrell, S. and R. Housley, "An Internet Attribute Certificate Profile for Authorization", RFC 3281, April 2002.

[AES] National Institute of Standards and Technology. FIPS Pub 197: Advanced Encryption Standard (AES). 26 November 2001.

[DDJ] Goldberg, I. and D. Wagner. "Randomness and the Netscape Browser." Dr. Dobb's Journal, January 1996.

[DPD&DPV] Pinkas, D. and R. Housley, "Delegated Path Validation and Delegated Path Discovery Protocol Requirements", RFC 3379, September 2002.

[OCSP] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

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

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

[SECREQMTS] National Institute of Standards and Technology. FIPS Pub 140-2: Security Requirements for Cryptographic Modules. 25 May 2001.

[X.509-97] ITU-T. Recommendation X.509: The Directory - Authentication Framework. 1997.

[X.509-00] ITU-T. Recommendation X.509: The Directory - Authentication Framework. 2000.

Appendix A: ASN.1 Module

The ASN.1 module contained in this appendix defines the structures that are needed to implement the CMS-based firmware package wrapper. It is expected to be used in conjunction with the ASN.1 modules in [CMS], [COMPRESS], and [PROFILE].

```
CMSFirmwareWrapper
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs-9(9) smime(16) modules(0) cms-firmware-wrap(22) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

IMPORTS
    EnvelopedData
    FROM CryptographicMessageSyntax -- [CMS]
         { iso(1) member-body(2) us(840) rsadsi(113549)
           pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) };


-- Firmware Package Content Type and Object Identifier

id-ct-firmwarePackage OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) ct(1) 16 }

FirmwarePkgData ::= OCTET STRING


-- Firmware Package Signed Attributes and Object Identifiers

id-aa-firmwarePackageID OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) aa(2) 35 }

FirmwarePackageIdentifier ::= SEQUENCE {
  name PreferredOrLegacyPackageIdentifier,
  stale PreferredOrLegacyStalePackageIdentifier OPTIONAL }

PreferredOrLegacyPackageIdentifier ::= CHOICE {
  preferred PreferredPackageIdentifier,
  legacy OCTET STRING }

PreferredPackageIdentifier ::= SEQUENCE {
  fwPkgID OBJECT IDENTIFIER,
  verNum INTEGER (0..MAX) }
```

```
   PreferredOrLegacyStalePackageIdentifier ::= CHOICE {
     preferredStaleVerNum INTEGER (0..MAX),
     legacyStaleVersion OCTET STRING }


   id-aa-targetHardwareIDs OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 36 }

   TargetHardwareIdentifiers ::= SEQUENCE OF OBJECT IDENTIFIER


   id-aa-decryptKeyID OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 37 }

   DecryptKeyIdentifier ::= OCTET STRING


   id-aa-implCryptoAlgs OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 38 }

   ImplementedCryptoAlgorithms ::= SEQUENCE OF OBJECT IDENTIFIER

   id-aa-implCompressAlgs OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 43 }

   ImplementedCompressAlgorithms ::= SEQUENCE OF OBJECT IDENTIFIER


   id-aa-communityIdentifiers OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 40 }

   CommunityIdentifiers ::= SEQUENCE OF CommunityIdentifier

   CommunityIdentifier ::= CHOICE {
     communityOID OBJECT IDENTIFIER,
     hwModuleList HardwareModules }

   HardwareModules ::= SEQUENCE {
     hwType OBJECT IDENTIFIER,
     hwSerialEntries SEQUENCE OF HardwareSerialEntry }
```

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


   id-aa-firmwarePackageInfo OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 42 }

   FirmwarePackageInfo ::= SEQUENCE {
     fwPkgType INTEGER OPTIONAL,
     dependencies SEQUENCE OF
       PreferredOrLegacyPackageIdentifier OPTIONAL }


   -- Firmware Package Unsigned Attributes and Object Identifiers

   id-aa-wrappedFirmwareKey OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) aa(2) 39 }

   WrappedFirmwareKey ::= EnvelopedData


   -- Firmware Package Load Receipt Content Type and Object Identifier

   id-ct-firmwareLoadReceipt OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) ct(1) 17 }

   FirmwarePackageLoadReceipt ::= SEQUENCE {
     version FWReceiptVersion DEFAULT v1,
     hwType OBJECT IDENTIFIER,
     hwSerialNum OCTET STRING,
     fwPkgName PreferredOrLegacyPackageIdentifier,
     trustAnchorKeyID OCTET STRING OPTIONAL,
     decryptKeyID [1] OCTET STRING OPTIONAL }

   FWReceiptVersion ::= INTEGER { v1(1) }
```

```
   -- Firmware Package Load Error Report Content Type
   -- and Object Identifier

   id-ct-firmwareLoadError OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) ct(1) 18 }

   FirmwarePackageLoadError ::= SEQUENCE {
     version FWErrorVersion DEFAULT v1,
     hwType OBJECT IDENTIFIER,
     hwSerialNum OCTET STRING,
     errorCode FirmwarePackageLoadErrorCode,
     vendorErrorCode VendorLoadErrorCode OPTIONAL,
     fwPkgName PreferredOrLegacyPackageIdentifier OPTIONAL,
     config [1] SEQUENCE OF CurrentFWConfig OPTIONAL }

   FWErrorVersion ::= INTEGER { v1(1) }

   CurrentFWConfig ::= SEQUENCE {
     fwPkgType INTEGER OPTIONAL,
     fwPkgName PreferredOrLegacyPackageIdentifier }

   FirmwarePackageLoadErrorCode ::= ENUMERATED {
     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),
     signatureFailure            (15),
     contentTypeMismatch         (16),
     badEncryptedData            (17),
     unprotectedAttrsPresent     (18),
     badEncryptContent           (19),
     badEncryptAlgorithm         (20),
     missingCiphertext           (21),
     noDecryptKey                (22),
     decryptFailure              (23),
     badCompressAlgorithm        (24),
     missingCompressedContent    (25),
```

```
      decompressFailure           (26),
      wrongHardware               (27),
      stalePackage                (28),
      notInCommunity              (29),
      unsupportedPackageType      (30),
      missingDependency           (31),
      wrongDependencyVersion      (32),
      insufficientMemory          (33),
      badFirmware                 (34),
      unsupportedParameters       (35),
      breaksDependency            (36),
      otherError                  (99) }

   VendorLoadErrorCode ::= INTEGER


   -- Other Name syntax for Hardware Module Name

   id-on-hardwareModuleName OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) dod(6) internet(1) security(5)
     mechanisms(5) pkix(7) on(8) 4 }

   HardwareModuleName ::= SEQUENCE {
     hwType OBJECT IDENTIFIER,
     hwSerialNum OCTET STRING }


   END
```

Author's Address

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

EMail: housley@vigilsec.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.