Internet Engineering Task Force (IETF)
Request for Comments: 5752
Category: Standards Track
ISSN: 2070-1721

S. Turner
IECA
J. Schaad
Soaring Hawk

January 2010

Multiple Signatures in Cryptographic Message Syntax (CMS)

Abstract

Cryptographic Message Syntax (CMS) SignedData includes the SignerInfo structure to convey per-signer information. SignedData supports multiple signers and multiple signature algorithms per signer with multiple SignerInfo structures. If a signer attaches more than one SignerInfo, there are concerns that an attacker could perform a downgrade attack by removing the SignerInfo(s) with the 'strong' algorithm(s). This document defines the multiple-signatures attribute, its generation rules, and its processing rules to allow signers to convey multiple SignerInfo objects while protecting against downgrade attacks. Additionally, this attribute may assist during periods of algorithm migration.

Status of This Memo

This is an Internet Standards Track document.

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

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5752.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

The Cryptographic Message Syntax (CMS; see [CMS]) defined SignerInfo to provide data necessary for relying parties to verify the signer's digital signature, which is also included in the SignerInfo structure. Signers include more than one SignerInfo in a SignedData if they use different digest or signature algorithms. Each SignerInfo exists independently and new SignerInfo structures can be added or existing ones removed without perturbing the remaining signatures.

The concern is that if an attacker successfully attacked a hash or signature algorithm, the attacker could remove all SignerInfo structures except the SignerInfo with the successfully attacked hash or signature algorithm. The relying party is then left with the attacked SignerInfo even though the relying party supported more than just the attacked hash or signature algorithm.

A solution is to have signers include a pointer to all the signer's SignerInfo structures. If an attacker removes any SignerInfo, then relying parties will be aware that an attacker has removed one or more SignerInfo objects.

Note that this attribute ought not be confused with the countersignature attribute (see Section 11.4 of [CMS]) as this is not intended to sign over an existing signature. Rather, it is to provide a pointer to additional signatures by the signer that are all at the same level. That is, countersignature provides a serial signature while the attribute defined herein provides pointers to parallel signatures by the same signer.

1.1. Conventions Used in This Document

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

2. Rationale

The rationale for this specification is to protect against downgrade attacks that remove the 'strong' signature to leave the 'weak' signature, which has presumably been successfully attacked. If a CMS SignedData object has multiple SignerInfo objects, then the attacker, whether it be Alice, Bob, or Mallory, can remove a SignerInfo object without the relying party being aware that more than one was generated.

Removal of a SignerInfo does not render the signature invalid nor does it constitute an error. In the following scenario, a signer generates a SignedData with two SignerInfo objects, one with a 'weak' algorithm and one with a 'strong' algorithm; there are three types of relying parties:

1) Those that support only a 'weak' algorithm. If both SignerInfo objects are present, the relying party processes the algorithm it supports. If both SignerInfo objects are not present, the relying party can easily determine that another SignerInfo has been removed, but not changed. In both cases, if the 'weak' signature verifies, the relying party MAY consider the signature valid.

2) Those that support only a 'strong' algorithm. If both SignerInfo objects are present, the relying party processes the algorithm it supports. If both SignerInfo objects are not present, the relying party can easily determine that another SignerInfo has been removed, but the relying party doesn't care. In both cases, if the 'strong' signature verifies, the relying party MAY consider the signature valid.

3) Those that support both a 'weak' algorithm and a 'strong' algorithm. If both SignerInfo objects are present, the relying party processes both algorithms. If both SignerInfo objects are not present, the relying party can easily determine that another SignerInfo has been removed. In both cases, if the 'strong' and/or 'weak' signatures verify, the relying party MAY consider the signature valid. (Policy may dictate that both signatures are required to validate if present.)

Local policy MAY dictate that the removal of the 'strong' algorithm results in an invalid signature. See Section 5 for further processing.

2.1. Attribute Design Requirements

The attribute will have the following characteristics:

1) Use CMS attribute structure;

2) Be computable before any signatures are applied;

3) Contain enough information to identify individual signatures (i.e., a particular SignerInfo); and

4) Contain enough information to resist collision, preimage, and second preimage attacks.

3. Multiple Signature Indication

The multiple-signatures attribute type specifies a pointer to a signer's other multiple-signatures attribute(s). For example, if a signer applies three signatures, there must be two attribute values for multiple-signatures in each SignerInfo. The 1st SignerInfo object points to the 2nd and 3rd SignerInfo objects. The 2nd SignerInfo object points to the 1st and 3rd SignerInfo objects. The 3rd SignerInfo object points to the 1st and 2nd SignerInfo objects.

The multiple-signatures attribute MUST be a signed attribute. The number of attribute values included in a SignerInfo is the number of signatures applied by a signer less one. This attribute is multi-valued, and there MAY be more than one AttributeValue present. The following object identifier identifies the multiple-signatures attribute:

```
id-aa-multipleSignatures OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  id-aa(16) 51 }
```

multiple-signatures attribute values have the ASN.1 type MultipleSignatures:

```
MultipleSignatures ::= SEQUENCE {
  bodyHashAlg     DigestAlgorithmIdentifier,
  signAlg         SignatureAlgorithmIdentifier,
  signAttrsHash   SignAttrsHash,
  cert            ESSCertIDv2 OPTIONAL}

SignAttrsHash ::= SEQUENCE {
  algID            DigestAlgorithmIdentifier,
  hash             OCTET STRING }
```

The fields in MultipleSignatures have the following meaning:

- bodyHashAlg includes the digest algorithmIdentifier for the referenced multiple-signatures attribute.

- signAlg includes the signature algorithmIdentifier for the referenced multiple-signatures attribute.

- signAttrsHash has two fields:

  -- algId MUST match the digest algorithm for the SignerInfo in which this multiple-signatures attribute value is placed.

  -- hash is the hash value of the signedAttrs (see Section 4.3).

- cert is optional. It identities the certificate used to sign the SignerInfo that contains the other multiple-signatures attribute(s). It MUST be present if the fields in the other multiple-signatures attribute(s) are the same.

The following is an example:

```
   SignedData
     DigestAlg=sha1,sha256
     SignerInfo1                SignerInfo2
       digestAlg=sha1             digestAlg=sha256
       signatureAlg=dsawithsha1   signatureAlg=ecdsawithsha256
       signedAttrs=               signedAttrs=
         signingTime1               signingTime1
         messageDigest1             messageDigest2
         multiSig1=                 multiSig2=
           bodyHash=sha256           bodyHash=sha1
           signAlg=ecdsawithsha256   signAlg=dsawithsha1
             signAttrsHash=          signAttrsHash=
             algID=sha1              algID=sha256
             hash=value1             hash=value2
```

4. Message Generation and Processing

The following are the additional procedures for message generation when using the multiple-signatures attribute. These paragraphs track with Sections 5.1-5.6 in [CMS].

4.1. SignedData Type

The following steps MUST be followed by a signer when generating SignedData:

- The signer MUST indicate the CMS version.

- The signer SHOULD include the digest algorithm used in SignedData.digestAlgorithms, if the digest algorithm's identifier is not already present.

- The signer MUST include the encapContentInfo. Note that the encapContentInfo is the same for all signers in this SignedData.

- The signer SHOULD add certificates sufficient to contain certificate paths from a recognized "root" or "top-level certification authority" to the signer, if the signer's certificates are not already present.

- The signer MAY include the Certificate Revocation Lists (CRLs) necessary to validate the digital signature, if the CRLs are not already present.

- The signer MUST:

  -- Retain the existing signerInfo objects.

  -- Include their signerInfo object(s).

4.2. EncapsulatedContentInfo Type

The procedures for generating EncapsulatedContentInfo are as specified in Section 5.2 of [CMS].

4.3. SignerInfo Type

The procedures for generating SignerInfo are as specified in Section 4.4.1 of [CMS] with the following addition:

The signer MUST include the multiple-signatures attribute in signedAttrs.

4.4. Message Digest Calculation Process

4.4.1. multiple-signatures Signed Attribute Generation

The procedure for generating the multiple-signatures signed attribute is as follows:

1) All other signed attributes are placed in the respective SignerInfo structures, but the signatures are not yet computed for the SignerInfo.

2) The multiple-signatures attributes are added to each of the SignerInfo structures with the SignAttrsHash.hash field containing a zero-length octet string.

3) The correct SignAttrsHash.hash value is computed for each of the SignerInfo structures.

4) After all hash values have been computed, the correct hash values are placed into their respective SignAttrsHash.hash fields.

4.4.2. Message Digest Calculation Process

The remaining procedures for generating the message-digest attribute are as specified in Section 5.4 of [CMS].

4.5. Signature Generation Process

The procedures for signature generation are as specified in Section 5.5 of [CMS].

4.6. Signature Verification Process

The procedures for signature verification are as specified in Section 5.6 of [CMS] with the following addition:

If the SignedData signerInfo includes the multiple-signatures attribute, the attribute's values must be calculated as described in Section 4.4.1.

For every SignerInfo to be considered present for a given signer, the number of MultipleSignatures AttributeValue(s) present in a given SignerInfo MUST equal the number of SignerInfo objects for that signer less one and the hash value present in each of the MultipleSignatures AttributeValue(s) MUST match the output of the message digest calculation from Section 4.4.1 for each SignerInfo.

The hash corresponding to the n-th SignerInfo must match the value in the multiple-signatures attribute that points to the n-th SignerInfo present in all other SignerInfo objects.

5. Signature Evaluation Processing

This section describes recommended processing of signatures when there are more than one SignerInfo present in a message. This may be due to either multiple SignerInfo objects being present in a single SignedData object or multiple SignerData objects embedded in each other.

The text in this section is non-normative. The processing described is highly recommended, but is not forced. Changes in the processing that have the same results with somewhat different orders of processing is sufficient.

Order of operations:

1) Evaluate each SignerInfo object independently.

2) Combine the results of all SignerInfo objects at the same level (i.e., attached to the same SignerData object).

3) Combine the results of the nested SignerData objects. Note that this should ignore the presence of other CMS objects between the SignedData objects.

5.1. Evaluation of a SignerInfo Object

When evaluating a SignerInfo object, there are three different pieces that need to be examined.

The first piece is the mathematics of the signature itself (i.e., can one actually successfully do the computations and get the correct answer?). This result is one of three results. The mathematics succeeds, the mathematics fails, or the mathematics cannot be evaluated. The type of things that lead to the last state are non-implementation of an algorithm or required inputs, such as the public key, are missing.

The second piece is the validation of the source of the public key. For CMS, this is generally determined by extracting the public key from a certificate. The certificate needs to be evaluated. This is done by the procedures outlined in [PROFILE]. In addition to the processing described in that document, there may be additional requirements on certification path processing that are required by the application in question. One such set of additional processing is described in [SMIME-CERT]. One piece of information that is part of this additional certificate path processing is local and application policy. The output of this processing can actually be one of four different states: Success, Failure, Indeterminate, and Warning. The first three states are described in [PROFILE]; Warning would be generated when it is possible that some information is currently acceptable, but may not be acceptable either in the near future or under some circumstances.

The third piece of the validation is local and application policy as applied to the contents of the SignerInfo object. This would cover such issues as the requirements on mandatory signed attributes or requirements on signature algorithms.

5.2. Evaluation of a SignerInfo Set

Combining the results of the individual SignerInfo objects into a result for a SignedData object requires knowledge of the results for the individual SignerInfo objects, the required application policy, and any local policies. The default processing if no other rules are applied should be:

1) Group the SignerInfo objects by the signer.

2) Take the best result from each signer.

3) Take the worst result from all of the different signers; this is the result for the SignedData object.

Application and local policy can affect each of the steps outlined above.

In Step 1:

- If the subject name or subject alternative name(s) cannot be used to determine if two SignerInfo objects were created by the same identity, then applications need to specify how such matching is to be done. As an example, the S/MIME message specification [SMIME-MSG] could say that as long as the same rfc822Name exists in either the subject name or the subject alt name they are the same identity. This would be true even if other information that did not match existed in these fields.

- Some applications may specify that this step should be skipped; this has the effect of making each SignerInfo object independent of all other SignerInfo objects even if the signing identity is the same. Applications that specify this need to be aware that algorithm rollover will not work correctly in this case.

In Step 2:

- The major policy implication at this step is the treatment of and order for the indeterminate states. In most cases, this state would be placed between the failure and warning states. Part of the issue is the question of having a multi-state or a binary answer as to success or failure of an evaluation. Not every application can deal with the statement "try again later". It may also be dependent on what the reason for the indeterminate state is. It makes more sense to try again later if the problem is that a CRL cannot be found than if you are not able to evaluate the algorithm for the signature.

In Step 3:

- The same policy implications from Step 2 apply here.

5.3. Evaluation of a SignedData Set

Simple applications will generally use the worst single outcome (success, unknown, failure) as the outcome for a set of SignedData objects (i.e., one failure means the entire item fails). However, not all applications will want to have this behavior.

A work flow application could work as follows:

The second signer will modify the original content, keep the original signature, and then sign the message. This means that only the outermost signature is of significance during evaluation. The second signer is asserting that they successfully validated the inner signature as part of its processing.

A Signed Mail application could work as follows:

If signatures are added for the support of [ESS] features, then the fact that an outer layer signature cannot be validated can be treated as a non-significant failure. The only thing that matters is that the originator signature is valid. This means that all outer layer signatures that fail can be stripped from the message prior to display leaving only the inner-most valid signature to be displayed.

6. Security Considerations

Security considerations from the hash and signature algorithms used to produce the SignerInfo apply.

If the hashing and signing operations are performed by different entities, the entity creating the signature must ensure that the hash comes from a "trustworthy" source. This can be partially mitigated by requiring that multiple hashes using different algorithms are provided.

This attribute cannot be relied upon in the event that all of the algorithms used in the signer attribute are 'cracked'. It is not possible for a verifier to determine that a collision could not be found that satisfies all of the algorithms.

Local policy and applications greatly affect signature processing. The application of local policy and the requirements specific to an application can both affect signature processing. This means that a signature valid in one context or location can fail validation in a different context or location.

7. References

7.1. Normative References

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

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

[PROFILE] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[SMIME-CERT] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, January 2010.

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

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

[ESSCertID] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, August 2007.

7.2. Informative References

[ATTACK] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

Appendix A. ASN.1 Module

```
MultipleSignatures-2008

  { iso(1) member-body(2) us(840) rsadsi(113549)
    pkcs(1) pkcs9(9) smime(16) modules(0)
    id-mod-multipleSig-2008(34) }

   DEFINITIONS IMPLICIT TAGS ::=

   BEGIN

-- EXPORTS All

-- The types and values defined in this module are exported for use
-- in the other ASN.1 modules.  Other applications may use them for
-- their own purposes.

IMPORTS

-- Imports from RFC 5652 [CMS], 12.1

     DigestAlgorithmIdentifier, SignatureAlgorithmIdentifier
     FROM CryptographicMessageSyntax2004
       { iso(1) member-body(2) us(840) rsadsi(113549)
         pkcs(1) pkcs9(9) smime(16) modules(0) cms-2004(24) }

-- Imports from RFC 5035 [ESSCertID], Appendix A

     ESSCertIDv2
     FROM ExtendedSecurityServices-2006
       { iso(1) member-body(2) us(840) rsadsi(113549)
         pkcs(1) pkcs9(9) smime(16) modules(0) id-mod-ess-2006(30) }

;

-- Section 3.0

id-aa-multipleSignatures OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs9(9) id-aa(2) 51 }

MultipleSignatures ::= SEQUENCE {
  bodyHashAlg     DigestAlgorithmIdentifier,
  signAlg         SignatureAlgorithmIdentifier,
  signAttrsHash   SignAttrsHash,
  cert            ESSCertIDv2 OPTIONAL }
```

```
SignAttrsHash ::= SEQUENCE {
  algID            DigestAlgorithmIdentifier,
  hash             OCTET STRING }

END -- of MultipleSignatures-2008
```

Appendix B. Background

This is an informational appendix. This appendix enumerates all locations in CMS where hashes are used and the possible attacks on those hash locations.

B.1. Attacks

As noted in [ATTACK], the following types of resistance are needed against known attacks:

1) Collision Resistance: Find x and y where x != y and H(x) = H(y)

2) Preimage Resistance: Given y, find x where H(x) = y

3) Second Preimage Resistance: Given y, find x where H(x) = H(y)

Note: It is known that a collision resistance attack is simpler than a second preimage resistance attack, and it is presumed that a second preimage resistance attack is simpler than a preimage attack.

B.2. Hashes in CMS

Within a SignerInfo there are two places where hashes are applied and hence can be attacked: the body and the signed attributes. The following outlines the entity that creates the hash, the entity that attacks the hash, and the type of resistance required:

1) Hash of the Body (i.e., the octets comprising the value of the encapContentInfo.eContent OCTET STRING omitting the tag and length octets, as per 5.4 of [CMS]).

   a) If Alice creates the body to be hashed, then:

      i) Alice can attack the hash. This attack requires a successful collision resistance attack.

      ii) Mallory can attack the hash. This attack requires a successful second preimage resistance attack.

   b) If Alice hashes a body provided by Bob, then:

      i) Alice can attack the hash. This attack requires a successful second preimage attack.

ii) Bob can attack the hash. This attack requires a successful Collision Resistance attack. If Alice has the ability to "change" the content of the body in some fashion, then this attack requires a successful second preimage attack. (One example would be to use a keyed hash function.)

iii) Mallory can attack the hash. This attack requires a successful second preimage attack.

c) If Alice signs using a hash value provided by Bob (in this case, Alice is presumed to never see the body in question), then:

i) Alice can attack the hash. This attack requires a successful preimage attack.

ii) Bob can attack the hash. This attack requires a successful collision resistance attack. Unlike case (b), there is nothing that Alice can do to upgrade the attack.

iii) Mallory can attack the hash. This requires a successful preimage attack if the content is unavailable to Mallory and a successful second preimage attack if the content is available to Mallory.

2) Hash of signed attributes (i.e., the complete Distinguished Encoding Rules (DER) encoding of the SignedAttrs value contained in the signedAttrs field, as per 5.4 of [CMS]).

There is a difference between hashing the body and hashing the SignedAttrs value in that one should not accept a sequence of attributes to be signed from a third party. In fact, one should not accept attributes to be included in the signed attributes list from a third party. The attributes are about the signature you are applying and not about the body. If there is meta-information that needs to be attached to the body by a third party, then they need to provide their own signature and you need to add a countersignature. (Note: The fact that the signature is to be used as a countersignature is a piece of information that should be accepted, but it does not directly provide an attribute that is inserted in the signed attribute list.)

a) Alice can attack the hash. This requires a successful collision resistance attack.

b) Mallory can attack the hash. This requires a successful second preimage resistance attack.

c) Bob can attack the hash and Bob controls the value of the message digest attribute used. This case is analogous to the current attacks [ATTACK]. Bob can attack the hash value generated by Alice based on a prediction of the signed attributes and the hash algorithm Alice will be using to create the signature. If Bob successfully predicts these items, the attack requires a successful collision resistance attack. (It is expected that if Alice uses a keyed hashing function as part of the signature, this attack will be more difficult as Bob would have a harder time prediction the key value.)

It should be noted that both of these attacks are considered to be more difficult than the attack on the body since more structure is designed into the data to be hashed than is frequently found in the body and the data is shorter in length than that of the body.

The successful prediction of the signing-time attribute is expected to be more difficult than with certificates as the time would not generally be rounded. Time stamp services can make this more unpredictable by using a random delay before issuing the signature.

Allowing a third party to provide a hash value could potentially make an attack simpler when keyed hash functions are used since there is more data than can be modified without changing the overall structure of the signed attribute structure.

Authors' Addresses

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

EMail: turners@ieca.com

Jim Schaad
Soaring Hawk Consulting

EMail: jimsch@exmsft.com