Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 7, 2013

J. Herzog
R. Khazan
MIT Lincoln Laboratory

October 4, 2012

A set-key attribute for symmetric-key packages
draft-herzog-setkey-07

Abstract

A set-key is a symmetric key (or set of keys) associated with an immutable set of participants. This document defines a set-key attribute for use in the CMS-based symmetric-key package structure [RFC6031].

Disclaimer

This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 7, 2013.

Copyright Notice

Table of Contents

1. Introduction

This document defines a new set-key attribute for use in the symmetric-key package structure defined in [RFC6031].

1.1. Set-keys

A 'set-key' is a symmetric key associated with (and to be limited to) a specific set of participants (entities). The exact definition of 'participant', in the context of a particular key, will depend on the key's intended application. Roughly speaking, however, the set of participants corresponds to the collection of individual agents which must possess and use the key for the key's intended purpose. In practice, a 'participant' will often be an entity with a certified public key, but other definitions of 'participant' are possible.

A set-key can be used in many ways, including:

- To secure broadcast or multicasts of e.g., pay-per-view movies,
- To secure group-communication such as chat rooms, or
- To secure data-at-rest which belongs to a group (such as on an encrypted file-system or on a server).

The only requirement of a set-key is that it be associated with a specific and unchanging set of participants. That is, this document draws a distinction between sets and groups.

- Sets are immutable structures: a set S of participants is a mathematical entity and does not change.
- A group, on the other hand, is a mutable structure that might have a certain name or set of names; may be under the control of a given administration; and may be mapped to a sequence of sets over time according to the needs of some particular application.

This document considers only sets, and therefore will not consider issues such as group-administration, adding or removing members, revoking keys, and so on. Set-keys can be used to build group-keying protocols, but such issues are outside the scope of this document.

By the 'participant-set' of a set-key, we mean the set of participants associated with the set-key. We further note that the participant-set of a set-key can be partitioned into two sub-sets: the active participants and the passive participants. In some set-key applications, such as group-chat, all participants may be active: both sending and receiving. In other applications, such as multicast

of pay-per-view movies, most participants are passive: only receiving. Because it may be important in some applications to know which partipants are active and which are passive, we will allow these two sub-sets to be explicitly distinguished from each other.

1.2. Symmetric key packages

The Cryptographic Message Syntax (CMS) [RFC5652] is a standard notation and representation for cryptographic messages. CMS is based on ASN.1 [X.680], [X.681], [X.682], [X.683] and uses that notation to define a number of structures relevant to cryptography such as certificates, encrypted or signed messages, and so on.

[RFC6031] uses CMS to define a structure for symmetric key packages: collections of symmetric keys which share common properties and are intended for the same participants. The syntax for this structure follows:

```
SymmetricKeyPackage ::= SEQUENCE {
  version           KeyPkgVersion DEFAULT v1,
  sKeyPkgAttrs  [0] SEQUENCE SIZE (1..MAX) OF Attribute
                                  {{ SKeyPkgAttributes }} OPTIONAL,
  sKeys             SymmetricKeys,
  ... }


SymmetricKeys ::= SEQUENCE SIZE (1..MAX) OF OneSymmetricKey

OneSymmetricKey ::= SEQUENCE {
```

```
    sKeyAttrs  SEQUENCE SIZE (1..MAX) OF Attribute
                                    {{ SKeyAttributes }}  OPTIONAL,
    sKey       OCTET STRING OPTIONAL }
    ( WITH COMPONENTS { ..., sKeyAttrs PRESENT } |
      WITH COMPONENTS { ..., sKey PRESENT } )

KeyPkgVersion ::= INTEGER  { v1(1) } ( v1, ... )

SKeyPkgAttributes ATTRIBUTE ::= { ... }

SKeyAttributes ATTRIBUTE ::= { ... }
```

A key-package (SymmetricKeyPackage) contains some meta-information, including attributes (sKeyPkgAttrs) [RFC6031], and a sequence of key-structures. Each individual key-structure (OneSymmetricKey) contains some attributes (sKeyAttrs) and some actual keying material (sKey). Attributes in the sKeyPkgAttrs field apply to every key in the package, while attributes in a sKeyAttrs field apply only to the key

in the containing OneSymmetricKey structure. The same attribute cannot appear at both the package-level and the key-level. That is, a given attribute can:

- appear in the sKeyPkgAttrs field but not in the sKeyAttrs field of any OneSymmetricKey structure,
- appear in the sKeyAttrs field of one or more OneSymmetricKey structures but not the sKeyPkgAttrs field, or
- not be included in the key-package at all.

Also, it is not required that every attribute be applicable at both levels. That is, [RFC6031] allows a given attribute to be valid for only the sKeyPkgAttrs field or for only the sKeyAttrs field.

The grammar for attributes is quite complex and given in [RFC5912]. (We also reproduce the relevant portion in Appendix B.) We intend to use only a small subset of this grammar, and therefore do not consider the full definition of attributes in this document.

1.3. Intended Usage

The attribute defined in this document is meant to be embedded in a SymmetricKeyPackage structure, perhaps with other attributes as well. The resulting value will then hold a set-key in a common and standard format. This format is appropriate both for transporting set-keys and for holding set-keys computed from some higher-level protocol. That is, one could create and distribute set-keys in this format through some distribution protocol, or one could use this format to hold set-keys computed by some protocol as Logical Key Hierarchy [RFC2627].

1.4. Requirements Terminology

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

2. The set-key attribute

The set-key attribute is used to associate a participant-set with a symmetric key-package, where the participant-set MAY be divided into the active and passive participants. It has the following syntax:


```
aa-setkey-information ATTRIBUTE ::= {
     TYPE SetKeyInformation
     IDENTIFIED BY id-aa-setKeyInformation }

id-aa-setKeyInformation     OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
         smime(16) aa(2) 53 }

SetKeyInformation::= SEQUENCE {
     active     SetKeyParticipantSet,
     passive    SetKeyParticipantSet OPTIONAL }
```

This attribute MUST NOT appear in both the sKeyPkgAttrs field of a SymmetricKeyPackage structure and the sKeyAttrs field of a

OneSymmetricKey structure. Furthermore, the sKeyPkgAttrs field of a SymmetricKeyPackage MUST contain either zero or one instance of a set-key attribute.

The 'active' and 'passive' fields jointly identify the participant-set of this set-key. The 'active' field identifies the active members of this set, meaning those which may use the key to protect data or messages. The 'passive' field identifies the passive members of this set, meaning those participants who are expected to have this key but never use it to apply cryptographic protection to messages or data. (They may use the key to process such protection, however, such as to decrypt the data or verify its integrity.)

Both the 'active' and 'passive' field will be values of type SetKeyParticipantSet, which has the following syntax:

```
SetKeyParticipantSet ::= CHOICE {
     union        [0] SEQUENCE SIZE (2..MAX) OF SetKeyParticipantSet,
     intersection [1] SEQUENCE SIZE (2..MAX) OF SetKeyParticipantSet,
     setdiff      [2] SEQUENCE {
                       orig SetKeyParticipantSet,
                       without SetKeyParticipantSet },
     community    [3] Community,
     groupID      [4] OCTET STRING,
     explicit     [5] SEQUENCE SIZE (1..MAX) OF SetMember,
     ...}
```

These values can be any of:

o A 'union' of other SetKeyParticipantSet values. The sequence of SetKeyParticipantSet values MUST contain at least two members.

The semantics of this sequence is as follows: a participant is in the union if and only if it is in any of the SetKeyParticipantSet values of the sequence. If any set in the sequence is non-empty, then the union will be non-empty. If all sets in the union sequence are empty, then the union is empty as well.

o The 'intersection' of other SetKeyParticipantSet values. The sequence of SetKeyParticipantSet values MUST contain at least two

members.The semantics of this sequence is as follows: a participant is in intersection if and only if it is in all of the SetKeyParticipantSet values of the sequence. If any set in the sequence is empty, the intersection is empty, too. If all sets in the sequence are non-empty, the intersection might still be empty (which it would be if there is no element common to all sets in the sequence).

- The set-difference ('setdiff') of two other SetKeyParticipantSet values. The semantics of this is as follows: a participant is in the set-difference if and only if it is in the set identified in the 'orig' field and not in the set identified by the 'without' field. If the set identified in the 'orig' field is empty, then the set-difference will be empty. If the set identified in the 'orig' field is a subset of the set identified in the 'without' field, then the set-difference will be empty. If neither of the two previous cases apply, the set-difference will be non-empty.

- A community, which is used by the Trust Anchor Management Protocol (TAMP) [RFC5934] to identify a set of 'cryptographic modules.' Community values used in set-key attributes MUST represent unchanging sets of participants.

- A 'groupID', which is interpreted in an application-independent way. One possible use for this option is to name pre-established groups such as organizational departments or roles. The details of establishing or using such a name-space are outside the scope of this document. However, groupID values used in set-key attributes MUST represent unchanging sets of participants.

- An 'explicit' list of SetMember values. Order has no meaning in this sequence. This sequence MUST NOT be empty.

This structure may be expanded at a later date with additional types.

A SetMember value, meant to identify a specific unique participant, has the following syntax:


```
SetMember ::= CHOICE {
```

```
      issuerAndSerialNumber  [0] IssuerAndSerialNumber,
      publicKey              [1] SubjectPublicKeyInfo,
      participantID          [2] OCTET STRING,
      ...}
```

A SetMember value can be any of:

- An IssuerAndSerialNumber value, defined in [RFC5911],

- A SubjectPublicKeyInfo, defined in [RFC5912], or

- A free form ('participantID') octet-string, which is interpreted in an application-dependent way. One possible use of this option is to use pre-established names for participants. However, such values MUST refer to a single participant, and MUST have a static (unchanging) meaning. The details of establishing or using such a name-space are outside the scope of this document.

As above, this structure may be expanded at a later date with additional types. Implementations SHOULD gracefully handle values and types which they do not recognize.

A single entity can be identified in multiple different ways. One example of this is that a certified key can be identified using the IssuerAndSerialNumber option, the SubjectPublicKeyInfo, or some application specific method (for example, a common name) in the free form field. This can cause ambiguity when evaluating a SetKeyParticipantSet. Suppose that a given key is sometimes identified using the IssuerAndSerialNumber form (value 'I') and sometimes identified using the SubjectPublicKeyInfo form (value 'S'). If these two distinct values are interpreted as identifying distinct keys, then the following structures may be misinterpreted:

- The 'explicit' sequence ...I, S... may be interpreted as having more elements than it actually does, due to the participant in question being counted twice (one for the I value, one for the S value).

- Similarly, the structure "union {...I...} {...S...}" may be interpreted as containing one more participant than it actually does.

- Likewise, the set "setdiff {...I...} {...S...}" may be interpreted as being non-empty when it is empty.

- On the other hand, the set "intersection {...I...} {...S...}" may be interpreted as being empty when it should not. (It will always contain at least one element: the participant indicated by both I and S.)

For these reasons, implementations SHOULD use a single method for identification of a single individual or have a well-established method of being able to compare the different locations that an individual could be identified in. We expand on this issue in Section 3.

3. Attribute generation

When creating a set-key attribute, the attribute generator (which will often also be the key source) begins with a set S of participants compromising the participant-set of the set-key. (This set MUST include the key-source.) The initial representation of this set or its members is beyond the scope of this document. The generator may also inherit, from context or application, an environment in which some sets S1, S2, S3... have been assigned names N1, N2, N3... Lastly, the attribute generator may know, from context or application, that some members of S will be passive users of the key. (That is, they will use the key to process messages created by others but not to create messages of their own using that key.)

First, the attribute generator must decide whether to use the optional passive field of the SetKeyInformation structure:

- If the generator knows that some members of S will be passive users, then it collects these members into a set P. The set P SHOULD only contain those participants in S which are known to be passive.

  The attribute generator then calculates the set A = S \ P (i.e., members of S not in P). The set A MUST NOT be empty. (This would indicate, absurdly, that the key will never be used to apply cryptographic protection.) The generator MUST ensure that the set A contains at least one member. If the set A is computed to be empty, the generator MUST NOT generate the set-key attribute.

  The attibute generator will then create a representation of P, generated as described below, and emplace it in the passive field of the set-key attribute. The generator then generates a representation of A as described below and stores this representation in the active field.

- Otherwise, the generator creates a representation of S as described below, stores it in the active field. Also, the generator MUST omit the passive field.

If the generator uses some other method to compute the sets to be represented in the active and passive fields, then:

- The generator SHOULD ensure that each element of S is contained either in the active field or the passive field, but not both.

- The generator MUST ensure that the set represented in the active field contains at least one member.

- If the generator uses the optional passive field, it MUST ensure that the value of that field represents a set containing at least one element. That is, the passive field MUST either be omitted or contain a value representing a non-empty set.

To emplace a set in either the active or passive fields of the set-key attribute, the attribute generator must construct a SetKeyParticipantSet value which represents that set. In general, the 'best' representation of a set S' (which may be S, A or P) will depend on the higher-level application being executed by the attribute generator. For example:

- Suppose that the set S' is small, or there is no mapping from names N1, N2, N3... to sets S1, S2, S3... available to the attribute generator. In this case the generator can create an explicit list of the set members. That is, the attribute generator represents each member of S' by a SetMember value as described above. It then collects these SetMember values into a sequence of SetMember values, and emplaces that sequence into a SetKeyParticipantSet value (using the 'explicit' option).

- As another example, suppose that purpose of the set-key is to re-key a group which is changing over time. If the current membership definition of this group has been given a name, then the attribute generator can simply emplace this name in a SetKeyParticipantValue (using the 'groupID' option).

- o To elaborate on the previous example: Suppose that the membership-set of a group had been given a name at some point in the past, but the group's membership has changed since then. Because the sets represented by SetKeyParticipantSet values represent unchanging sets, the name represents the prior membership-set, not the current one. However, the name in question can be used to represent the current membership set in a way which may be more compact than the explicit list described in the first example

above. Suppose that the group's prior membership-set was assigned the name N. Then the attribute generator can create two explicit lists of participants:

* In the first list, L1, the generator lists all participants which are currently in the group but were not in the group when it received the name N.

* In the second list, L2, the generator lists all participants which were in the group when it received the name N but are not in the group currently.

In both cases, the attribute generator resolves each participant in the list to a SetMember value as described below, gathers all SetMember values into a sequence, and emplaces the sequence into a SetKeyParticipantSet value (using the 'explicit' option). The attribute generator can then represent the current membership of the group by the value:

```
setdiff { orig: (union (groupID: N) L1),
          without: l2 }
```

That is, it first emplaces the name N in a SetKeyParticipantSet value using the 'groupID' option. It then creates a sequence compromised of this value and the SetKeyParticipantSet representing L1, and emplaces that in a SetKeyParticipantSet value (using the 'union' option). It then creates a SetKeyParticipantSet value using the 'setdiff' option, where the previously-mentioned SetKeyParticipantSet value is in the 'orig' field and the SetKeyParticipantSet value representing L2 is in the

'without' field. In this way, the attribute-generator can represent the current membership-set of the group by describing the changes since it was given the name N, which may be a more compact representation than explicitly listing the entire current membership.

Regardless of how the attribute generator chooses to represent the set:

- If a representation includes a Community value (the 'community' option) or an octet string (the 'groupID' option), then the attribute generator MUST know (through some method beyond the scope of this document) that this octet string unambiguously indicates the participant in question to every participant in the set key's participant-set.

- If a representation requires that a participant be explicitly described by a SetMember value, then it SHOULD be represented by exactly one such value. That is, the participant may be 'mentioned' multiple times in the representation. In this case, though, the same value should be used in each such instance. Specifically, the attribute-generator SHOULD select a SetMember representation for the participant in this order:

  1. An IssuerAndSerialNumber value, if the participant possesses a certificate. If more than one IssuerAndSerialNumber value is associated with that participant, the attribute generator MAY choose one using any criteria it likes.

  2. A SubjectPublicKeyInfo value. However, the attribute-generator MUST know (though some method beyond the scope of this document) that the associated private key is known to the participant and only that participant.

  3. An octet string (the 'participantID' option). However, the attribute generator MUST know (through some method beyond the scope of this document) that this octet string unambiguously indicates the participant in question to every participant in the set key's participant-set.

- Although the sets represented by the 'active' and 'passive' fields

cannot be empty, sub-components of these values MAY be empty. This is because resource-constrained entities may wish to do the minimum processing required to verify that the 'active' and 'passive' sets are non-empty. If the 'active' value is a 'union' value at the top level, for example, the sender may wish to terminate processing when it first finds a non-empty component of that union (as opposed to confirming that every component represents a non-empty set). Senders MAY invest the processing necessary to ensure that no component of the 'active' and 'passive' sets represent non-empty sets, but receivers of these values MUST properly handle the case that a sub-component represents an empty set.

4. Attribute processing

As with attribute generation, attribute processing will be heavily dependent on the intended application of the attribute and the set key. However, we describe here two common cases: reconstruction of the described set, and testing set-membership.

- Suppose the attribute processor wishes to reconstruct the set represented in either the 'active' or 'passive' field. The

processor's final, internal representation of this set is beyond the scope of this document. However, one possible process for constructing this set mirrors the semantics of the attribute:

* If the SetKeyParticipantSet is of the 'union' option, then the processor recursively resolves each SetKeyParticipantSet value in the sequence into participant sets. The final participant set is the union of each set in the sequence.

* If the SetKeyParticipantSet is of the 'intersection' option, then the processor recursively resolves each SetKeyParticipantSet value in the sequence into participant sets. The final participant set is the intersection of these sets.

* If the SetKeyParticipantSet is of the 'setdiff' option, then the processor recursively resolves the SetKeyParticipantSet values of the 'orig' and 'without' fields into sets. The final

participant set contains exactly those participants in the 'orig' set and not in the 'without' set.

* If the SetKeyParticipantSet is of the 'community' or 'groupID' option, then the attribute processor resolves the Community value or octet string, respectively, into a participant set. The methods by which the attribute-processor performs this operation are beyond the scope of this document. However, if the identifier in question cannot unambiguously be resolved to a set of participants, then the attempt to reconstruct the set represented by this attribute MUST fail with an error.

* If the SetKeyParticipantSet is of the 'explicit' option, then the attribute processor resolves each SetMember value in the sequence into a participant. The resulting participant set contains exactly those participants. To resolve a SetMember value into a participant, the attribute processor might use the following process:

  + If the SetMember value is of the 'issuerAndSerialNumber' option, then the participant is the entity indicated in the name field of the indicated certificate [RFC5280].

  + If the SetMember value is of the 'publicKey' option, then the participant is the entity who knows the associated private key. If the number of such entities is other than exactly one, then the attempt to reconstruct the set represented by this attribute MUST fail with an error.

  + If the SetMember value of the 'participantID' option, then the attribute processor resolves the octet string into a participant. The method by which the attribute-processor performs this operation is beyond the scope of this document. However, if the identifier in question cannot unambiguously be resolved to exactly one participant, then the attempt to reconstruct the set represented by this attribute MUST fail with an error.

This process SHOULD return an error if either the 'active' or 'passive' fields are resolved to empty sets.

- o Suppose, on the other hand, the attribute processor wishes test whether a particular participant is a member of the indicated set. To do so, the processor must receive as input both the SetKeyParticipantSet value and the participant in question (the 'target' participant). The representation of this participant is beyond the scope of this document, so long as the attribute processor can match it against SetMember values (see below).

  To test set-membership of the target participant, the attribute processor can again follow a process which mirrors the underlying semantics of the attribute:

  - * If the SetKeyParticipantSet is of the 'union' option, then the processor iterates over the sequence. For each SetKeyParticipantSet value in the sequence, the attribute processor recursively tests the membership of the target participant in the represented set. This recursive test MAY return true, false, or an error. If any recursive test returns true, then the current test (for this 'union' value) will also return true. Furthermore, the attribute processor MAY immediately terminate iteration over the sequence upon receiving a 'true' value from a recursive test. If the attribute processor finishes iterating over the sequence, however, and each recursive test returned 'false', then the current test returns 'false' as well. If any recursive test returned 'error' and no recursive test returned 'true', then the current test also returns 'error'.

  - * If the SetKeyParticipantSet is of the 'intersection' option, then the processor iterates over the sequence. For each SetKeyParticipantSet value in the sequence, the attribute processor recursively tests the membership of the target participant in the represented set. This recursive test MAY return true, false, or an error. If any recursive test returns false, then the current test (for this 'intersection' value) will also return false. Furthermore, the attribute processor

MAY immediately terminate iteration over the sequence upon receiving a 'false' value from a recursive test. If the attribute processor finishes iterating over the sequence, however, and each recursive test returned 'true', then the

current test returns 'true' as well. If any recursive test returned 'error' and no recursive test returned 'false', then the current test also returns 'error'.

* If the SetKeyParticipantSet is of the 'setdiff' option, then the processor recursively resolves the SetKeyParticipantSet values of the 'orig' and 'without' fields into sets. It then recursively tests membership of the target participant in these two sets. It returns 'true' if the test of the 'orig' set returns 'true' and the test for the 'without' set returns 'false'. If either test returned 'error', then the current test (for this 'setdiff' value) returns 'error' as well. Else, the current test returns 'false'.

* If the SetKeyParticipantSet is of the 'community' or 'groupID' option, then the attribute processor resolves the Community value or octet string, respectively, into a participant set. It then returns success if participant is the set, and failure if not. The methods by which the attribute-processor resolves the identifier into a set are beyond the scope of this document. However, if the identifier in question cannot unambiguously be resolved to a set of participants, then the attempt to reconstruct the set represented by this attribute MUST fail with an error.

* If the SetKeyParticipantSet is of the 'explicit' option, then processor iterates over the sequence. That is, the attribute processor attempts to match the target participant against each SetMember value in the sequence. This matching can return true, false, or an error. If any matching returns true, then the current test (for this 'explicit' value) will also return true. Furthermore, the attribute processor MAY immediately terminate iteration over the sequence upon receiving a 'true' value from a matching. If the attribute processor finishes iterating over the sequence, however, and each matching returned 'false', then the current test returns 'false' as well. If any matching returned 'error' and no recursive test returned 'true', then the current test also returns 'error'.

If the attribute processor further wishes to determine whether the target participant is active or passive, it recursively tests whether the target participant is in the set represented by the SetKeyParticipantSet value in the 'active' field of the attribute. If so, the target participant MUST be considered active. If not,

the attribute processor tests whether the target participant is in the set represented by the SetKeyParticipantSet value in the 'passive' field of the attribute. If so, the participant MUST be considered passive. If not, the target participant is not in the participant set of the key.

If the attribute processor uses some other method to process the set-key attribute, then:

- An attribute processor MUST gracefully handle the case where a given participant is the set represented by the SetKeyParticipantSet value of the 'active' field and the value of the 'passive' field. In this case, the participant in question MUST be considered to be active.

- The attribute processor MUST gracefully handle SetKeyParticipantSet values in which a given participant is represented by two different values. Specifically, if the attribute processor ever fails to resolve a SetMember value unambiguously to a single participant, it MUST produce an error that it either handles itself or returns to a higher-level caller. Furthermore, the attribute processor MUST NOT use purely syntactic equality-tests for participants. That is, attribute processors must internally represent participants in such a way that two different SetMember values will be recognized as representing the same participant (if, in fact, they do).

- If the SetKeyParticipantSet value of either the 'active' or 'passive' field is resolved as representing an empty set, the attribute processor MUST consider this to be an invalid value and return an error.

- The attribute processor SHOULD gracefully handle values and types which they do not recognize (for sets and members).

5. Security Considerations

As with the entire symmetric-key package, the set-key attribute is not protected. The symmetric key package content type can be combined with a security protocol to protect the contents of the attribute.

6. IANA Considerations

This document makes use of object identifiers. These object

identifiers are defined in an arc delegated to the IETF S/MIME

Working Group. This arc and its registration procedures will be transferred to IANA soon. No further action by IANA is necessary for this document. Future extensions may require action by IANA, but such actions will be described at the time of extension.

7. Acknowledgments

The authors would like to thank Jim Schaad, Russ Housley, Sean Turner, Carl Wallace, Menachem Dodge, and Alexey Melnikov for their helpful comments and suggestions.

8. References

8.1. Normative References

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Request For Comments 5280, May 2008.

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

[RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, June 2010.

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

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, August 2010.

[RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax

(CMS) Symmetric Key Package Content Type", RFC 6031, December 2010.

[X.680] ITU-T, "Information Technology - Abstract Syntax Notation One", Recommendation X.680, ISO/IEC 8824-1:2002, 2002.

[X.681] ITU-T, "Information Technology - Abstract Syntax Notation One: Information Object Specification",

Recommendation X.681, ISO/IEC 8824-2:2002, 2002.

[X.682] ITU-T, "Information Technology - Abstract Syntax Notation One: Constraint Specification", Recommendation X.682, ISO/IEC 8824-3:2002, 2002.

[X.683] ITU-T, "Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications", Recommendation X.683, ISO/IEC 8824-4:2002, 2002.

8.2. Informative References

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, June 1999.

Appendix A. ASN.1 Module

This appendix provides the normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680] through [X.683].

```
SetKeyAttributeV1
  { iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0) id-mod-setKeyAttributeV1(62) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS
```

```
   ATTRIBUTE
     FROM PKIX-CommonTypes-2009
       {iso(1) identified-organization(3) dod(6) internet(1) security(5)
       mechanisms(5) pkix(7) id-mod(0) id-mod-pkixCommon-02(57)}
   IssuerAndSerialNumber
     FROM CryptographicMessageSyntax-2009
       { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
         smime(16) modules(0) id-mod-cms-2004-02(41)}
   SubjectPublicKeyInfo
     FROM PKIX1Explicit-2009
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-pkix1-explicit-02(51) }
   Community
```



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

aa-setkey-information ATTRIBUTE ::= {
     TYPE SetKeyInformation
     IDENTIFIED BY id-aa-setKeyInformation }

id-aa-setKeyInformation     OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
         smime(16) aa(2) 53 }

SetKeyInformation::= SEQUENCE {
     active     SetKeyParticipantSet,
     passive    SetKeyParticipantSet OPTIONAL }

SetKeyParticipantSet ::= CHOICE {
     union        [0] SEQUENCE SIZE (2..MAX) OF SetKeyParticipantSet,
     intersection [1] SEQUENCE SIZE (2..MAX) OF SetKeyParticipantSet,
     setdiff      [2] SEQUENCE {
                       orig SetKeyParticipantSet,
                       without SetKeyParticipantSet },
     community    [3] Community,
     groupID      [4] OCTET STRING,
     explicit     [5] SEQUENCE SIZE (1..MAX) OF SetMember,
```

```
     ...}

SetMember ::= CHOICE {
     issuerAndSerialNumber  [0] IssuerAndSerialNumber,
     publicKey              [1] SubjectPublicKeyInfo,
     participantID          [2] OCTET STRING,
     ...}

END
```

Appendix B. ASN.1 structures for Attributes

ATTRIBUTE values are defined in the ASN.1 module PKIX-CommonTypes-2009 of [RFC5912]. For information purposes, we reproduce the relevant portion of this module here:


```
ATTRIBUTE ::= CLASS {
    &id             OBJECT IDENTIFIER UNIQUE,
    &Type           OPTIONAL,
    &equality-match MATCHING-RULE OPTIONAL,
    &minCount       INTEGER DEFAULT 1,
    &maxCount       INTEGER OPTIONAL
} WITH SYNTAX {
    [TYPE &Type]
    [EQUALITY MATCHING RULE &equality-match]
    [COUNTS [MIN &minCount] [MAX &maxCount]]
    IDENTIFIED BY &id
}


MATCHING-RULE ::= CLASS {
  &ParentMatchingRules   MATCHING-RULE OPTIONAL,
  &AssertionType         OPTIONAL,
  &uniqueMatchIndicator  ATTRIBUTE OPTIONAL,
  &id                    OBJECT IDENTIFIER UNIQUE
```

```
    }
    WITH SYNTAX {
      [PARENT &ParentMatchingRules]
      [SYNTAX &AssertionType]
      [UNIQUE-MATCH-INDICATOR &uniqueMatchIndicator]
      ID &id
    }
```

Authors' Addresses

Jonathan C. Herzog
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02144
USA

Email: jherzog@ll.mit.edu

Roger Khazan
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02144
USA

Email: rkh@ll.mit.edu