Network Working Group
Request for Comments: 3923
Category: Standards Track

P. Saint-Andre
Jabber Software Foundation

October 2004

End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)

Status of this Memo

Copyright Notice

Abstract

This memo defines methods of end-to-end signing and object encryption for the Extensible Messaging and Presence Protocol (XMPP).

Table of Contents

1. Introduction

This memo defines methods of end-to-end signing and object encryption for the Extensible Messaging and Presence Protocol (XMPP). (For information about XMPP, see [XMPP-CORE] and [XMPP-IM].) The method specified herein enables a sender to sign and/or encrypt an instant message sent to a specific recipient, sign and/or encrypt presence information that is directed to a specific user, and sign and/or encrypt any arbitrary XMPP stanza directed to a specific user. This memo thereby helps the XMPP specifications meet the requirements specified in [IMP-REQS].

1.1. Terminology

This document inherits terminology defined in [CMS], [IMP-MODEL], [SMIME], and [XMPP-CORE].

The capitalized 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 BCP 14, RFC 2119 [TERMS].

2. Requirements

For the purposes of this memo, we stipulate the following requirements:

1. The method defined MUST address signing and encryption requirements for minimal instant messaging and presence, as those are defined in [IMP-REQS]. In particular, the method MUST address the following requirements, which are copied here verbatim from [IMP-REQS]:

   * The protocol MUST provide means to ensure confidence that a received message (NOTIFICATION or INSTANT MESSAGE) has not been corrupted or tampered with. (Section 2.5.1)

   * The protocol MUST provide means to ensure confidence that a received message (NOTIFICATION or INSTANT MESSAGE) has not been recorded and played back by an adversary. (Section 2.5.2)

   * The protocol MUST provide means to ensure that a sent message (NOTIFICATION or INSTANT MESSAGE) is only readable by ENTITIES that the sender allows. (Section 2.5.3)

   * The protocol MUST allow any client to use the means to ensure non-corruption, non-playback, and privacy, but the protocol MUST NOT require that all clients use these means at all times. (Section 2.5.4)

   * When A establishes a SUBSCRIPTION to B's PRESENCE INFORMATION, the protocol MUST provide A means of verifying the accurate receipt of the content B chooses to disclose to A. (Section 5.1.4)

   * The protocol MUST provide A means of verifying that the presence information is accurate, as sent by B. (Section 5.3.1)

   * The protocol MUST provide A means of ensuring that no other PRINCIPAL C can see the content of M. (Section 5.4.6)

   * The protocol MUST provide A means of ensuring that no other PRINCIPAL C can tamper with M, and B means to verify that no tampering has occurred. (Section 5.4.7)

2. The method defined MUST enable interoperability with non-XMPP messaging systems that support the Common Presence and Instant Messaging (CPIM) specifications published by the Instant Messaging and Presence (IMPP) Working Group. Two corollaries of this requirement are:

   * Prior to signing and/or encrypting, the format of an instant message MUST conform to the CPIM Message Format defined in [MSGFMT].

   * Prior to signing and/or encrypting, the format of presence information MUST conform to the CPP Presence Information Data Format defined in [PIDF].

3. The method MUST follow the required procedures (including the specific algorithms) defined in [CPIM] and [CPP]. In particular, these documents specify:

   * Signing MUST use [SMIME] signatures with [CMS] SignedData.

   * Encryption MUST use [SMIME] encryption with [CMS] EnvelopeData.

4. In order to enable interoperable implementations, sending and receiving applications MUST implement the algorithms specified under Mandatory-to-Implement Cryptographic Algorithms (Section 6.10).

We further stipulate that the following functionality is out of scope for this memo:

- Discovery of support for this protocol. An entity could discover whether another entity supports this protocol by (1) attempting to send signed or encrypted stanzas and receiving an error stanza ("technical" discovery) or a textual message in reply ("social" discovery) if the protocol is not supported, or (2) using a dedicated service discovery protocol, such as [DISCO] or [CAPS]. However, the definition of a service discovery protocol is out of scope for this memo.

- Signing or encryption of XMPP groupchat messages, which are mentioned in [XMPP-IM] but not defined therein since they are not required by [IMP-REQS]; such messages are best specified in [MUC].

- Signing or encryption of broadcasted presence as described in [XMPP-IM] (the methods defined herein apply to directed presence only).

- Signing or encryption of communications that occur within the context of applications other than instant messaging and presence as those are described in [IMP-MODEL] and [IMP-REQS].

3. Securing Messages

3.1. Process for Securing Messages

In order to sign and/or encrypt a message, a sending agent MUST use the following procedure:

1. Generate a "Message/CPIM" object as defined in [MSGFMT].

2. Sign and/or encrypt both the headers and content of the "Message/CPIM" object as specified in Requirement 3 of Section 2 above.

3. Provide the resulting signed and/or encrypted object within an XML CDATA section (see Section 2.7 of [XML]) contained in an <e2e/> child of a <message/> stanza, where the <e2e/> element is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace as specified more fully in Section 9 below.

3.2. Example of a Signed Message

The following example illustrates the defined steps for signing a message.

First, the sending agent generates a "Message/CPIM" object in accordance with the rules and formats specified in [MSGFMT].

Example 1: Sender generates "Message/CPIM" object:

```
|   Content-type: Message/CPIM
|
|   From: Juliet Capulet <im:juliet@example.com>
|   To: Romeo Montague <im:romeo@example.net>
|   DateTime: 2003-12-09T11:45:36.66Z
|   Subject: Imploring
|
|   Content-type: text/plain; charset=utf-8
|   Content-ID: <1234567890@example.com>
|
|   Wherefore art thou, Romeo?
```

Once the sending agent has generated the "Message/CPIM" object, the sending agent may sign it. The result is a multipart [SMIME] object (see [MULTI]) that has a Content-Type of "multipart/signed" and includes two parts: one whose Content-Type is "Message/CPIM" and another whose Content-Type is "application/pkcs7-signature".

Example 2: Sender generates multipart/signed object:

```
|   Content-Type: multipart/signed; boundary=next;
|                 micalg=sha1;
|                 protocol=application/pkcs7-signature
|
|   --next
|   Content-type: Message/CPIM
|
|   From: Juliet Capulet <im:juliet@example.com>
|   To: Romeo Montague <im:romeo@example.net>
|   DateTime: 2003-12-09T23:45:36.66Z
|   Subject: Imploring
|
|   Content-type: text/plain; charset=utf-8
|   Content-ID: <1234567890@example.com>
|
|   Wherefore art thou, Romeo?
|   --next
|   Content-Type: application/pkcs7-signature
|   Content-Disposition: attachment;handling=required;\
|                                    filename=smime.p7s
|
|   [signed body part]
|
|   --next--
```

The sending agent now wraps the "multipart/signed" object in an XML CDATA section, which is contained in an <e2e/> element that is included as a child element of the XMPP message stanza and that is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace.

Example 3: Sender generates XMPP message stanza:

```
|   <message to='romeo@example.net/orchard' type='chat'>
|     <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
|   <![CDATA[
|   Content-Type: multipart/signed; boundary=next;
|                 micalg=sha1;
|                 protocol=application/pkcs7-signature
|
|   --next
|   Content-type: Message/CPIM
|
|   From: Juliet Capulet <im:juliet@example.com>
|   To: Romeo Montague <im:romeo@example.net>
|   DateTime: 2003-12-09T23:45:36.66Z
|   Subject: Imploring
|
|   Content-type: text/plain; charset=utf-8
|   Content-ID: <1234567890@example.com>
|
|   Wherefore art thou, Romeo?
|   --next
|   Content-Type: application/pkcs7-signature
|   Content-Disposition: attachment;handling=required;\
|                                   filename=smime.p7s
|
|   [signed body part]
|
|   --next--
|   ]]>
|     </e2e>
|   </message>
```

3.3. Example of an Encrypted Message

The following example illustrates the defined steps for encrypting a message.

First, the sending agent generates a "Message/CPIM" object in accordance with the rules and formats specified in [MSGFMT].

Example 4: Sender generates "Message/CPIM" object:

```
|   Content-type: Message/CPIM
|
|   From: Juliet Capulet <im:juliet@example.com>
|   To: Romeo Montague <im:romeo@example.net>
|   DateTime: 2003-12-09T11:45:36.66Z
|   Subject: Imploring
|
|   Content-type: text/plain; charset=utf-8
|   Content-ID: <1234567890@example.com>
|
|   Wherefore art thou, Romeo?
```

Once the sending agent has generated the "Message/CPIM" object, the sending agent may encrypt it.

Example 5: Sender generates encrypted object:

```
|   U2FsdGVkX19okeKTlLxa/1n1FE/upwn1D20GhPWqhDWlexKMUKYJInTWzERP+vcQ
|   /OxFs40uc9Fx81a5/62p/yPb/UWnuG6SR6o3Ed2zwcusDImyyz125HFERdDUMBC9
|   Pt6Z4cTGKBmJzZBGyuc3Y+TMBTxqFFUAxeWaoxnZrrl+LP72vwbriYc3KCMxDbQL
|   Igc1Vzs5/5JecegMieNY24SlNyX9HMFRNFpbI64vLxYEk55A+3IYbZsluCFT31+a
|   +GeAvJkvH64LRV4mPbUhENTQ2wbAwnOTvbLIaQEQrii78xNEh+MK8Bx7TBTvi4yH
|   Ddzf9Sim6mtWsXaCAvWSyp0X91d7xRJ4JIgKfPzkxNsWJFCLthQS1p734eDxXVd3
|   i08lEHzyll6htuEr59ZDAw==
```

The sending agent now wraps the encrypted object in an XML CDATA section, which is contained in an <e2e/> element that is included as a child element of the XMPP message stanza and that is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace.

Example 6: Sender generates XMPP message stanza:

```
|   <message to='romeo@example.net/orchard' type='chat'>
|     <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
|   <![CDATA[
|   U2FsdGVkX19okeKTlLxa/1n1FE/upwn1D20GhPWqhDWlexKMUKYJInTWzERP+vcQ
|   /OxFs40uc9Fx81a5/62p/yPb/UWnuG6SR6o3Ed2zwcusDImyyz125HFERdDUMBC9
|   Pt6Z4cTGKBmJzZBGyuc3Y+TMBTxqFFUAxeWaoxnZrrl+LP72vwbriYc3KCMxDbQL
|   Igc1Vzs5/5JecegMieNY24SlNyX9HMFRNFpbI64vLxYEk55A+3IYbZsluCFT31+a
|   +GeAvJkvH64LRV4mPbUhENTQ2wbAwnOTvbLIaQEQrii78xNEh+MK8Bx7TBTvi4yH
|   Ddzf9Sim6mtWsXaCAvWSyp0X91d7xRJ4JIgKfPzkxNsWJFCLthQS1p734eDxXVd3
|   i08lEHzyll6htuEr59ZDAw==
|   ]]>
|     </e2e>
|   </message>
```

4. Securing Presence

4.1. Process for Securing Presence Information

In order to sign and/or encrypt presence information, a sending agent MUST use the following procedure:

1. Generate an "application/pidf+xml" object as defined in [PIDF].
2. Sign and/or encrypt the "application/pidf+xml" object as specified in Requirement 3 of Section 2 above.
3. Provide the resulting signed and/or encrypted object within an XML CDATA section (see Section 2.7 of [XML]) contained in an <e2e/> child of a <presence/> stanza, where the <e2e/> element is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace. The <presence/> stanza MUST include a 'to' attribute, i.e., it must be an instance of directed presence as defined in [XMPP-IM].

4.2. Example of Signed Presence Information

The following example illustrates the defined steps for signing presence information.

First, the sending agent generates an "application/pidf+xml" object in accordance with the rules and formats specified in [PIDF].

Example 7: Sender generates "application/pidf+xml" object:

```
|   <?xml version="1.0" encoding="UTF-8"?>
|   <presence xmlns="urn:ietf:params:xml:ns:pidf"
|             xmlns:im="urn:ietf:params:xml:ns:pidf:im"
|             entity="pres:juliet@example.com">
|     <tuple id="hr0zny"
|       <status>
|         <basic>open</basic>
|         <im:im>away</im:im>
|       </status>
|       <note xml:lang="en">retired to the chamber</note>
|       <timestamp>2003-12-09T23:53:11.31</timestamp>
|     </tuple>
|   </presence>
```

Once the sending agent has generated the "application/pidf+xml" object, the sending agent may sign it. The result is a multipart [SMIME] object (see [MULTI]) that has a Content-Type of "multipart/signed" and includes two parts: one whose Content-Type is "application/pidf+xml" and another whose Content-Type is "application/pkcs7-signature".

Example 8: Sender generates multipart/signed object:

```
|   Content-Type: multipart/signed; boundary=next;
|                 micalg=sha1;
|                 protocol=application/pkcs7-signature
|
|   --next
|   Content-type: application/pidf+xml
|   Content-ID: <2345678901@example.com>
|
|   <xml version="1.0" encoding="UTF-8"?>
|   <presence xmlns="urn:ietf:params:xml:ns:pidf"
|             xmlns:im="urn:ietf:params:xml:ns:pidf:im"
|             entity="pres:juliet@example.com">
|     <tuple id="hr0zny">
|       <status&gt;
|         <basic>open</basic>
|         <im:im>away</im:im>
|       </status>
|       <note xml:lang="en">retired to the chamber</note>
|       <timestamp>2003-12-09T23:53:11.31Z</timestamp>
|     </tuple>
|   </presence>
|   --next
|   Content-Type: application/pkcs7-signature
|   Content-Disposition: attachment;handling=required;\
|                                   filename=smime.p7s
|
|   [signed body part]
|
|   --next--
```

The sending agent now wraps the "multipart/signed" object in an XML CDATA section, which is contained in an <e2e/> element that is included as a child element of the XMPP message stanza and that is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace.

Example 9: Sender generates XMPP presence stanza:

```
|   <presence to='romeo@example.net/orchard'>
|     <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
|   <![CDATA[
|   Content-Type: multipart/signed; boundary=next;
|                 micalg=sha1;
|                 protocol=application/pkcs7-signature
|
|   --next
|   Content-type: application/pidf+xml
|   Content-ID: <2345678901@example.com>
|
|   <xml version="1.0" encoding="UTF-8"?>
|   <presence xmlns="urn:ietf:params:xml:ns:pidf"
|             xmlns:im="urn:ietf:params:xml:ns:pidf:im"
|             entity="pres:juliet@example.com">
|     <tuple id="hr0zny">
|       <status>
|         <basic>open</basic>
|         <im:im>away</im:im>
|       </status>
|       <note xml:lang="en">retired to the chamber</note>
|       <timestamp>2003-12-09T23:53:11.31Z</timestamp>
|     </tuple>
|   </presence>
|   --next
|   Content-Type: application/pkcs7-signature
|   Content-Disposition: attachment;handling=required;\
|                                   filename=smime.p7s
|
|   [signed body part]
|
|   --next--
|   ]]>
|     </e2e>
|   </presence>
```

4.3. Example of Encrypted Presence Information

The following example illustrates the defined steps for encrypting presence information.

First, the sending agent generates an "application/pidf+xml" object in accordance with the rules and formats specified in [PIDF].

Example 10: Sender generates "application/pidf+xml" object:

```
|   <?xml version="1.0" encoding="UTF-8"?>
|   <presence xmlns="urn:ietf:params:xml:ns:pidf"
|             xmlns:im="urn:ietf:params:xml:ns:pidf:im"
|             entity="pres:juliet@example.com">
|     <tuple id="hr0zny"
|       <status>
|         <basic>open</basic>
|         <im:im>away</im:im>
|       </status>
|       <note xml:lang="en">retired to the chamber</note>
|       <timestamp>2003-12-09T23:53:11.31</timestamp>
|     </tuple>
|   </presence>
```

Once the sending agent has generated the "application/pidf+xml" object, the sending agent may encrypt it.

Example 11: Sender generates encrypted object:

```
|   U2FsdGVkX18VJPbx5GMdFPTPZrHLC9QGiVP+ziczu6zWZLFQxae6O5PP6iqpr2No
|   zOvBVMWvYeRAT0zd18hr6qsqKiGl/GZpAAbTvPtaBxeIykxsd1+CX+U+iw0nEGCr
|   bjiQrk0qUKJ79bNxwRnqdidjhyTpKSbOJC0XZ8CTe7AE9KDM3Q+uk+O3jrqX4byL
|   GBlKThbzKidxz32ObojPEEwfFiM/yUeqYUP1OcJpUmeQ8lcXhD6tcx+m2MAyYYLP
|   boKQxpLknxRnbM8T/voedlnFLbbDu69mOlxDPbr1mHZd3hDsyFudb1fb4rI3Kw0K
|   Nq+3udr2IkysviJDgQo+xGIQUG/5sED/mAaPRlj4f/JtTzvT4EaQTawv69ntXfKV
|   MCr9KdIMMdjdJzOJkYLoAhNVrcZn5tw8WsJGwuKuhYb/SShy7InzOapPaPAl7/Mm
|   PHj7zj3NZ6EEIweDOuAwWlIG/dT506tci27+EW7JnXwMPnFMkF+6a7tr/0Y+iiej
|   woJxUIBqCOgX+U7srHpK2NYtNTZ7UQp2V0yEx1JV8+Y=
```

The sending agent now wraps the encrypted object in an XML CDATA section, which is contained in an <e2e/> element that is included as a child element of the XMPP message stanza and that is qualified by the ’urn:ietf:params:xml:ns:xmpp-e2e’ namespace.

Example 12: Sender generates XMPP presence stanza:

```
|   <presence to='romeo@example.net/orchard'>
|     <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
|   <![CDATA[
|   U2FsdGVkX18VJPbx5GMdFPTPZrHLC9QGiVP+ziczu6zWZLFQxae6O5PP6iqpr2No
|   zOvBVMWvYeRAT0zd18hr6qsqKiGl/GZpAAbTvPtaBxeIykxsd1+CX+U+iw0nEGCr
|   bjiQrk0qUKJ79bNxwRnqdidjhyTpKSbOJC0XZ8CTe7AE9KDM3Q+uk+O3jrqX4byL
|   GBlKThbzKidxz32ObojPEEwfFiM/yUeqYUP1OcJpUmeQ8lcXhD6tcx+m2MAyYYLP
|   boKQxpLknxRnbM8T/voedlnFLbbDu69mOlxDPbr1mHZd3hDsyFudb1fb4rI3Kw0K
|   Nq+3udr2IkysviJDgQo+xGIQUG/5sED/mAaPRlj4f/JtTzvT4EaQTawv69ntXfKV
|   MCr9KdIMMdjdJzOJkYLoAhNVrcZn5tw8WsJGwuKuhYb/SShy7InzOapPaPAl7/Mm
|   PHj7zj3NZ6EEIweDOuAwWlIG/dT506tci27+EW7JnXwMPnFMkF+6a7tr/0Y+iiej
|   woJxUIBqCOgX+U7srHpK2NYtNTZ7UQp2V0yEx1JV8+Y=
|   ]]>
|     </e2e>
|   </presence>
```

5. Securing Arbitrary XMPP Data

The foregoing sections of this memo describe how to secure "least common denominator" messaging and presence data of the kind that can be directly translated into the MSGFMT or PIDF formats. However, XMPP possesses a third base-level stanza type (<iq/>) in addition to <message/> and <presence/>, as well as the ability to include extended XML data within arbitrary child elements of the three core stanza types. Therefore, it would be desirable to secure such data if possible.

Because [MSGFMT] specifies the ability to encapsulate any MIME type, the approach taken in this memo is to include arbitrary XMPP data in an XML media type named "application/xmpp+xml" as specified more fully in Section 10 below.

The following examples illustrate the structure of the "application/xmpp+xml" MIME type. (Note: The 'http://jabber.org/protocol/evil' namespace used in these examples is associated with an April Fool's protocol written to be the instant messaging equivalent of RFC 3514; it is included only as an instance of extended information included in an XML stanza and should not be taken seriously as a functional XMPP extension.)

Example 13: Message stanza with extended data contained in "application/xmpp+xml" MIME type:

```
|   <?xml version='1.0' encoding='UTF-8'?>
|   <xmpp xmlns='jabber:client'>
|     <message
|         from='iago@example.com/pda'
|         to='emilia@example.com/cell'>
|       <body>
|         I told him what I thought, and told no more
|         Than what he found himself was apt and true.
|       </body>
|       <evil xmlns='http://jabber.org/protocol/evil'/>
|     </message>
|   </xmpp>
```

Example 14: Presence stanza with extended data contained in "application/xmpp+xml" MIME type:

```
|   <?xml version='1.0' encoding='UTF-8'?>
|   <xmpp xmlns='jabber:client'>
|     <presence from='iago@example.com/pda'>
|       <show>dnd</show>
|       <status>Fomenting dissension</status>
|       <evil xmlns='http://jabber.org/protocol/evil'/>
|     </presence>
|   </xmpp>
```

Example 15: IQ stanza with extended data contained in "application/xmpp+xml" MIME type:

```
|   <?xml version='1.0' encoding='UTF-8'?>
|   <xmpp xmlns='jabber:client'>
|     <iq type='result'
|         from='iago@example.com/pda'
|         to='emilia@example.com/cell'
|         id='evil1'>
|       <query xmlns='jabber:iq:version'>
|         <name>Stabber</name>
|         <version>666</version>
|         <os>FiendOS</os>
|       </query>
|       <evil xmlns='http://jabber.org/protocol/evil'/>
|     </iq>
|   </xmpp>
```

Just as with the "Message/CPIM" and "application/pidf+xml" objects, the "application/xmpp+xml" object would be signed and/or encrypted, then encapsulated within an XML CDATA section (see Section 2.7 of [XML]) contained in an <e2e/> child of a <presence/> stanza, where the <e2e/> element is qualified by the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace.

6. Rules for S/MIME Generation and Handling

6.1. Certificate Enrollment

[SMIME] does not specify how to obtain a certificate from a certificate authority, but instead mandates that every sending agent must already have a certificate. The PKIX Working Group has, at the time of this writing, produced two separate standards for certificate enrollment: [CMP] and [CMC]. Which method to use for certificate enrollment is outside the scope of this memo.

6.2. Certificate Retrieval

A receiving agent MUST provide some certificate retrieval mechanism in order to gain access to certificates for recipients of digital envelopes. This memo does not address how S/MIME agents handle certificates, only what they do after a certificate has been validated or rejected. S/MIME certification issues are covered in [CERT].

However, at a minimum, for initial S/MIME deployment, a user agent SHOULD automatically generate a message to an intended recipient requesting that recipient's certificate in a signed return message. Receiving and sending agents SHOULD also provide a mechanism to allow a user to "store and protect" certificates for correspondents in such a way so as to guarantee their later retrieval.

6.3. Certificate Names

End-entity certificates used by XMPP entities in the context of this memo SHOULD contain a valid instant messaging and presence address. The address SHOULD be specified as both an 'im:' URI (for instant messaging, as defined in [CPIM]) and a 'pres:' URI (for presence, as defined in [CPP]); each of these URIs SHOULD be specified in a separate GeneralName entry of type uniformResourceIdentifier inside the subjectAltName (i.e., two separate entries). Information in the subject distinguished name SHOULD be ignored.

Each URI MUST be of the form <im:address> or <pres:address>, where the "address" portion is an XMPP address (also referred to as a Jabber Identifier or JID) as defined in [XMPP-CORE], prepended with

the 'im:' or 'pres:' URI scheme. The address SHOULD be of the form <node@domain> (i.e., a "bare JID"), although any valid JID form MAY be used.

The value of the JID contained in the XMPP 'from' attribute MUST match a JID provided in the signer's certificate, with the exception that the resource identifier portion of the JID contained in the 'from' attribute SHOULD be ignored for matching purposes.

Receiving agents MUST check that the sending JID matches a JID provided in the signer's certificate, with the exception that the resource identifier portion of the JID contained in the 'from' attribute SHOULD be ignored for matching purposes. A receiving agent SHOULD provide some explicit alternate processing of the stanza if this comparison fails, which may be to display a message informing the recipient of the addresses in the certificate or other certificate details.

The subject alternative name extension is used in S/MIME as the preferred means to convey the instant messaging and presence address that corresponds to the entity for this certificate. Any XMPP address present in the certificate MUST be encoded using the ASN.1 Object Identifier "id-on-xmppAddr" as specified in Section 5.1.1 of [XMPP-CORE].

6.4. Transfer Encoding

Because it is expected that XMPP applications will not interface with older 7-bit systems, the transfer encoding (as defined in Section 3.1.2 of [SMIME]) MUST be "binary".

6.5. Order of Signing and Encrypting

If a stanza is both signed and encrypted, it SHOULD be signed first, then encrypted.

6.6. Inclusion of Certificates

If the sender and recipient are involved in an active messaging session over a period of time, the sending agent SHOULD include the sender's certificate along with at least one encrypted message stanza every five minutes. Outside the context of an active messaging session, the sending agent SHOULD include the sender's certificate along with each encrypted message stanza. A sending agent MAY include the sender's certificate along with each encrypted presence stanza. However, a sending agent SHOULD NOT include a certificate more than once every five minutes.

6.7. Attachment and Checking of Signatures

Sending agents SHOULD attach a signature to each encrypted XML stanza. If a signature is attached, a Content-Disposition header field (as defined in [DISP]) SHOULD be included to specify how the signature is to be handled by the receiving application.

If the receiving agent determines that the signature attached to an encrypted XML stanza is invalid, it SHOULD NOT present the stanza to the intended recipient (human or application), SHOULD provide some explicit alternate processing of the stanza (which may be to display a message informing the recipient that the attached signature is invalid), and MAY return a stanza error to the sender as described under Recipient Error Handling (Section 7).

6.8. Decryption

If the receiving agent is unable to decrypt the encrypted XML stanza, it SHOULD NOT present the stanza to the intended recipient (human or application), SHOULD provide some explicit alternate processing of the stanza (which may be to display a message informing the recipient that it has received a stanza that cannot be decrypted), and MAY return a stanza error to the sender as described under Recipient Error Handling (Section 7).

6.9. Inclusion and Checking of Timestamps

Timestamps are included in "Message/CPIM" and "application/pidf+xml" objects to help prevent replay attacks. All timestamps MUST conform to [DATETIME] and be presented as UTC with no offset, including fractions of a second as appropriate. Absent a local adjustment to the sending agent's perceived time or the underlying clock time, the sending agent MUST ensure that the timestamps it sends to the receiver increase monotonically (if necessary by incrementing the seconds fraction in the timestamp if the clock returns the same time for multiple requests). The following rules apply to the receiving application:

- It MUST verify that the timestamp received is within five minutes of the current time.

- It SHOULD verify that the timestamp received is greater than any timestamp received in the last 10 minutes which passed the previous check.

- If any of the foregoing checks fails, the timestamp SHOULD be presented to the receiving entity (human or application) marked as "old timestamp", "future timestamp", or "decreasing timestamp", and the receiving entity MAY return a stanza error to the sender as described under Recipient Error Handling (Section 7).

6.10. Mandatory-to-Implement Cryptographic Algorithms

All implementations MUST support the following algorithms. Implementations MAY support other algorithms as well.

For CMS SignedData:

- The SHA-1 message digest as specified in [CMS-ALG] section 2.1.

- The RSA (PKCS #1 v1.5) with SHA-1 signature algorithm, as specified in [CMS-ALG] section 3.2.

For CMS EnvelopedData:

- The RSA (PKCS #1 v1.5) key transport, as specified in [CMS-ALG] section 4.2.1.

- The AES-128 encryption algorithm in CBC mode, as specified in [CMS-AES].

7. Recipient Error Handling

When an XMPP entity receives an XML stanza containing data that is signed and/or encrypted using the protocol described herein, several scenarios are possible:

Case #1: The receiving application does not understand the protocol.

Case #2: The receiving application understands the protocol and is able to decrypt the payload and verify the sender's signature.

Case #3: The receiving application understands the protocol and is able to decrypt the payload and verify the sender's signature, but the timestamps fail the checks specified above under Checking of Timestamps (Section 6.9).

Case #4: The receiving application understands the protocol and is able to decrypt the payload but is unable to verify the sender's signature.

Case #5: The receiving application understands the protocol but is unable to decrypt the payload.

In Case #1, the receiving application MUST do one and only one of the following: (1) ignore the <e2e/> extension, (2) ignore the entire stanza, or (3) return a <service-unavailable/> error to the sender, as described in [XMPP-CORE].

In Case #2, the receiving application MUST NOT return a stanza error to the sender, since this is the success case.

In Case #3, the receiving application MAY return a <not-acceptable/> error to the sender (as described in [XMPP-CORE]), optionally supplemented by an application-specific error condition element <bad-timestamp/> as shown below:

Example 16: Recipient returns <not-acceptable/> error:

```
<message from='romeo@example.net/orchard' type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
  [CDATA section here]
  </e2e>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-timestamp xmlns='urn:ietf:params:xml:xmpp-e2e'/>
  </error>
</message>
```

In Case #4, the receiving application SHOULD return a <not-acceptable/> error to the sender (as described in [XMPP-CORE]), optionally supplemented by an application-specific error condition element <unverified-signature/> as shown below:

Example 17: Recipient returns <not-acceptable/> error:

```
<message from='romeo@example.net/orchard' type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
  [CDATA section here]
  </e2e>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <unverified-signature xmlns='urn:ietf:params:xml:xmpp-e2e'/>
  </error>
</message>
```

In Case #5, the receiving application SHOULD return a <bad-request/> error to the sender (as described in [XMPP-CORE]), optionally supplemented by an application-specific error condition element <decryption-failed/> as shown below:

Example 18: Recipient returns <bad-request/> error:

```
<message from='romeo@example.net/orchard' type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'>
  [CDATA section here]
  </e2e>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <decryption-failed xmlns='urn:ietf:params:xml:xmpp-e2e'/>
  </error>
</message>
```

8. Secure Communications Through a Gateway

A common method for achieving interoperability between two disparate services is through the use of a "gateway" that interprets the protocols of each service and translates them into the protocols of the other. The CPIM specifications (specifically [MSGFMT] and [PIDF] define the common profiles to be used for interoperability between instant messaging and presence services that comply with [IMP-REQS]. In the case of communications between an XMPP service and a non-XMPP service, we can visualize this relationship as follows:

The end-to-end encryption method defined herein enables the exchange of encrypted and/or signed instant messages and presence through an XMPP-CPIM gateway. In particular:

- When a gateway receives a secured XMPP message or presence stanza from the XMPP service that is addressed to a user on the non-XMPP service, it MUST remove the XMPP "wrapper" (everything down to and including the <e2e> and </e2e> tags) in order to reveal the multipart S/MIME object, then route the object to the non-XMPP service (first wrapping it in the protocol used by the non-XMPP service if necessary).

- When a gateway receives a secured non-XMPP instant message or presence document from the non-XMPP service that is addressed to a user on the XMPP service, it MUST remove the non-XMPP "wrapper" (if any) in order to reveal the multipart S/MIME object, wrap the object in an XMPP message or presence "wrapper" (including the <e2e> and </e2e> tags), and then route the XMPP stanza to the XMPP service.

The wrapped S/MIME object MUST be immutable and MUST NOT be modified by an XMPP-CPIM gateway.

9. urn:ietf:params:xml:xmpp-e2e Namespace

The <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e'/> element is a wrapper for an XML CDATA section (see Section 2.7 of [XML]) that contains a "Message/CPIM", "application/pidf+xml", or "application/xmpp+xml" object. Thus the 'urn:ietf:params:xml:xmpp-e2e' namespace has no inherent semantics, and the semantics of the encapsulated object are defined by one of the following specifications:

- [MSGFMT] for "Message/CPIM"
- [PIDF] for "application/pidf+xml"
- [XMPP-CORE] for "application/xmpp+xml"

Although the "application/xmpp+xml" media type is specified in this document, the <xmpp/> element is simply a wrapper for a <message/>, <presence/>, or <iq/> stanza, where the semantics of those stanza types are specified in [XMPP-CORE].

Given that the 'urn:ietf:params:xml:ns:xmpp-e2e' namespace has no inherent semantics and specifies a using protocol only, versioning is the responsibility of the protocols that define the encapsulated objects ([MSGFMT], [PIDF], and [XMPP-CORE]).

10. application/xmpp+xml Media Type

The "application/xmpp+xml" media type adheres to the guidelines specified in [XML-MEDIA]. The root element for this MIME type is <xmpp/>, and the root element MUST contain one and only one child element, corresponding to one of the XMPP stanza types (i.e., message, presence, or iq) if the default namespace is 'jabber:client' or 'jabber:server' as defined in [XMPP-CORE]. The character encoding for this XML media type MUST be UTF-8, in accordance with Section 11.5 of [XMPP-CORE].

11. Security Considerations

This entire memo discusses security. Detailed security considerations for instant messaging and presence protocols are given in [IMP-REQS] (Sections 5.1 through 5.4), and for XMPP in particular are given in [XMPP-CORE] (Sections 12.1 through 12.6). In addition, all of the security considerations specified in [XML-MEDIA] apply to the "application/xmpp+xml" media type.

The end-to-end security method defined here MAY result in exchanging secured instant messages and presence information through a gateway that implements the CPIM specifications. Such a gateway MUST be compliant with the minimum security requirements of the instant messaging and presence protocols with which it interfaces.

12. IANA Considerations

12.1. XML Namespace Name for e2e Data in XMPP

A URN sub-namespace of signed and encrypted content for the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [XML-REG].)

URI: urn:ietf:params:xml:ns:xmpp-e2e
Specification: RFC 3923
Description: This is an XML namespace name of signed and encrypted content for the Extensible Messaging and Presence Protocol as defined by RFC 3923.
Registrant Contact: IESG, <iesg@ietf.org>

12.2. Content-type Registration for "application/xmpp+xml"

To: ietf-types@iana.org

Subject: Registration of MIME media type application/xmpp+xml

MIME media type name: application
MIME subtype name: xmpp+xml
Required parameters: (none)
Optional parameters: (charset) Same as charset parameter of application/xml as specified in RFC 3023; per Section 11.5 of [XMPP-CORE], the charset must be UTF-8.
Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023; per Section 11.5 of [XMPP-CORE], the encoding must be UTF-8.

Security considerations: All of the security considerations specified in RFC 3023 and [XMPP-CORE] apply to this XML media type. Refer to Section 11 of RFC 3923.
Interoperability considerations: (none)
Specification: RFC 3923
Applications which use this media type: XMPP-compliant instant messaging and presence systems.
Additional information: (none)
Person and email address to contact for further information: IESG, <iesg@ietf.org>
Intended usage: COMMON
Author/Change controller: IETF, XMPP Working Group

13. References

13.1. Normative References

[CERT] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, July 2004.

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

[CMS-AES] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[CMS-ALG] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[CPIM] Peterson, J., "Common Profile for Instant Messaging (CPIM)", RFC 3860, August 2004.

[CPP] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[DATETIME] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[DISP] Troost, R., Dorner, S., and K. Moore, Ed., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[IMP-MODEL] Day, M., Rosenberg, J., and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[IMP-REQS] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging/Presence Protocol Requirements", RFC 2779, February 2000.

[MSGFMT] Klyne, G. and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", RFC 3862, August 2004.

[MULTI] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[PIDF] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

[SMIME] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

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

[XML-MEDIA] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[XMPP-CORE] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[XMPP-IM] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP) Instant Messaging and Presence", RFC 3921, October 2004.

13.2. Informative References

[CAPS] Hildebrand, J. and P. Saint-Andre, "Entity Capabilities", JSF JEP 0115, August 2004.

[CMC] Myers, M., Liu, X., Schaad, J. and J. Weinstein, "Certificate Management Messages over CMS", RFC 2797, April 2000.

[CMP] Adams, C. and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999.

[DISCO] Hildebrand, J., Millard, P., Eatmon, R. and P. Saint-Andre, "Service Discovery", JSF JEP 0030, July 2004.

[MUC] Saint-Andre, P., "Multi-User Chat", JSF JEP 0045, June 2004.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler, "Extensible Markup Language (XML) 1.0 (3rd ed)", W3C REC-xml, February 2004, <http://www.w3.org/TR/REC-xml>.

[XML-REG] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

Appendix A. Schema for urn:ietf:params:xml:ns:xmpp-e2e

The following XML schema is descriptive, not normative.

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:ietf:params:xml:ns:xmpp-e2e'
    xmlns='urn:ietf:params:xml:ns:xmpp-e2e'
    elementFormDefault='qualified'>

  <xs:element name='e2e' type='xs:string'/>

  <xs:element name='decryption-failed' type='empty'/>
  <xs:element name='signature-unverified' type='empty'/>
  <xs:element name='bad-timestamp' type='empty'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Author's Address

Peter Saint-Andre
Jabber Software Foundation

EMail: stpeter@jabber.org

Full Copyright Statement

Intellectual Property

Acknowledgement

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