XMPP M. Miller
Internet-Draft Cisco Systems, Inc.
Intended status: Standards Track C. Wallace
Expires: January 5, 2015 Red Hound Software, Inc.
July 4, 2014

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

draft-miller-xmpp-e2e-07

Abstract

This document defines two methods for securing objects (often referred to as stanzas) for the Extensible Messaging and Presence Protocol (XMPP), which allows for efficient asynchronous communication between two entities, each with might have multiple devices operating simultaneously. One is a method to encrypt stanzas to provide confidentiality protection; another is a method to sign stanzas to provide authentication and integrity protection. This document also defines a related protocol for entities to request the ephemeral session keys in use.

Table of Contents

1. Introduction

End-to-end protection and authentication of traffic sent over the Extensible Messaging and Presence Protocol [RFC6120] is a desirable goal. Requirements and a threat analysis for XMPP encryption are provided in [E2E-REQ]. Many possible approaches to meet those (or similar) requirements have been proposed over the years, including methods based on PGP, S/MIME, SIGMA, and TLS.

Most proposals have not been able to support multiple end-points for a given recipient. As more devices support XMPP, it becomes more desirable to allow an entity to communicate with another in a more secure manner, regardless of the number of agents the entity is employing. This document specifies an approach for encrypting and signing communications between two entities which each might have multiple end-points.

A primary challenge with supporting multiple end-points is key distribution. This is complicated by the fact that some end points for a given recipient may share keys, some may use different keys, some may have no keys and some may not support encryption or signature verification at all. To address these differences, this specification defines a symmetric key table that is managed via three mechanisms that enable a key to be pushed to an end point, to be pulled from an originator or negotiated. The key table contains named master keys along with meta data describing usage of the key. Encrypted XMPP messages use a named master key to encrypt a content encryption key. Prior to decrypting a message, recipients of an encrypted message will either find the named key present in their key table (as the result of an earlier operation) or obtain the key from the sender.

Comments are solicited and should be addressed to XMPP mailing list. Information about the XMPP mailing list can be found here: https://www.ietf.org/mailman/listinfo/xmpp.

2. Terminology

This document inherits XMPP-related terminology from [RFC6120], JSON Web Algorithms (JWA)-related terminology from [JOSE-JWA], JSON Web Encryption (JWE)-related terminology from [JOSE-JWE], and JSON Web Key (JWK)-related terminology from [JOSE-JWK]. Security-related terms are to be understood in the sense defined in [RFC4949].

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 [RFC2119].

3. Changes to existing clients

3.1. End-point procedures

Existing XMPP clients will need to implement some new procedures in order to support end-to-end encryption and authentication. Changes for sending clients include:

- Generating session master keys (SMKs)
- Storing SMKs for use during active sessions
- Storing SMKs to provide to peers and to support reading of saved messages (may require use of storage key)
- Accepting requests for SMKs
- Releasing SMKs to authorized requestors (where requests may be received from multiple different resources associated with a single peer with each resource using a different means to authenticate)
- Generating content encryption keys (CEK)
- Using SMK and CEK values to encrypt XMPP stanzas
- Generating a signing key (optional)
- Using a signing key to sign XMPP stanzas
- Generating and using a long term storage key (optional)

Changes for receiving clients include:

- Sending requests for SMKs to peers

- Accepting public key to use when encrypting an SMK from peers

- Storing SMKs for use when decrypting XMPP stanzas during active session

- Using an SMK to decrypt a CEK used to decrypt XMPP stanzas

- Storing SMKs retrieved from peers to support reading of saved messages (may require use of storage key)

- Providing indication to users when encryption is in use

- Retrieving keys required to verify signatures on signed XMPP stanzas

- Verifying signatures and displaying indication of success/failure to user

- Storing keys required to verify signature to support reading of saved messages (may require use of storage key)

- Generating and using a long term storage key (optional)

3.2. End-point state

End points utilizing end-to-end encryption and signatures are required to maintain some new state information, and may find some additional information helpful to maintain. New state information includes:

- Session master key table (required)

- Public/private key store (required)

- Trust anchor store (optional)

- Intermediate certification authority (CA) store (optional)

- Long-term storage key (optional)

Session master keys (SMKs) are used to encrypt XMPP stanzas. An end-point may have many active SMKs at any given point in time, but only one SMK active per bare JID (TODO: or should this be per full JID?). Each SMK has a name generated by the entity who generated the key.

The name MUST be unique from the generator's perspective (i.e., full JID + SMK name MUST uniquely identify a specific SMK). When a new SMK is received, any previous SMK stored for the full JID of the entity providing the SMK may be destroyed. Alternatively, previous SMKs may be preserved to support future decryption of stored messages. This specification places no requirements on handling of stored messages. Clients may re-encrypt messages under a long-term storage key, store messages as-is encrypted using an SMK or store plaintext messages.

Each end-point must have at least one public/private key pair used for SMK distribution.

A trust anchor store or intermediate CA store may be useful to support automated release of encrypted SMKs or to verify signed XMPP stanzas.

A long-term storage key may be used to either encrypt data stored in the key table or to re-encrypt encrypted messages prior to storing the message for future review.

4. Key distribution

Several different types of keys are used to support end-to-end encryption and signatures. These keys may be distinct from any keys used to authenticate to XMPP servers and include the following:

- o Session master key (SMK)
- o Content encryption keys (CEKs) for XMPP stanzas
- o Public/private key pair for SMK distribution
- o Content encryption keys for SMK distribution
- o Public/private key pair for signature generation
- o Trust anchor and intermediate certification authority (CA) public keys
- o Long-term storage key

SMKs are symmetric keys generated by an end-point prior to utilizing end-to-end encryption (see Section 6.2.1). SMKs are used to encrypt the CEK used to encrypt an XMPP stanza. SMKs are stored in the SMK table and may be distributed using one of the following mechanisms:

o Manually pre-placed at some point prior to using end-to-end encryption

o Released to an end-point upon request after receiving an encrypted XMPP stanza

o Provided to an end-point using an IQ stanza sent prior to sending encrypted XMPP stanzas

CEKs for XMPP stanzas are symmetric keys generated by an end-point to encrypt an XMPP stanza (see item 5 in Section 6.2.2). CEKs are encrypted using the SMK and included with encrypted XMPP data.

Public/private key pairs for SMK distribution are asymmetric keys that may be generated by an end point, imported into an end point or used via a hardware cryptographic module. The public key is distributed to XMPP peers for use when distributing SMKs (see step 1 in Section 8.1). The public key is formatted as a JWK, which may include an X.509 certificate. An end-point MUST establish trust in a public key prior to releasing an SMK value. Trust establishment mechanisms include checking a key thumbprint provided via a trusted channel or by validating an X.509 certificate to a trust anchor. The public keys may be distributed using one of the following mechanisms:

o Manually pre-placed prior to using for SMK release (details for manual pre-placement are not defined by this specification)

o Presented when requesting an SMK from a peer after receiving an encrypted XMPP stanza from the peer (the peer may store the public key for use in providing future encrypted SMK values prior to using the SMK to encrypt XMPP stanzas see Section 8.1)

o Provided upon request in response to an IQ get request in preparation for receiving encrypted XMPP stanzas (TODO: define IQ for pushing SMK)

CEKs for SMK distribution are symmetric keys generated by an end-point to encrypt an SMK (see item 3 in Section 8.2). CEKs are encrypted using the public key used for SMK distribution and included with encrypted SMK data.

Public/private key pairs for SMK distribution are asymmetric keys that may be generated by an end point, imported into an end point or used via a hardware cryptographic module (see bullet 4 of section 5.1 in [JOSE-JWE]). The public key is distributed to XMPP peers for use when verifying signatures. Trust establishment may be performed by checking a key thumbprint provided via a trusted channel or by validating an X.509 certificate to a trust anchor.

Trust anchor and intermediate CA public keys may be used to validate X.509 certificates in support of SMK release or verification of signatures on signed XMPP stanzas.

A long-term storage key may be used to encrypt information stored in the key table or to re-encrypt encrypted messages prior to storing the message for future review. The long-term storage key may be a public/private key pair or a symmetric key.

5. Key table

The conceptual database for long-lived cryptographic keys described in [Key-Table] may be suitable for use in storing the SMKs described above for use in supporting end-to-end XMPP encryption. The columns that the table consists of are listed as follows:

TODO: figure out whether to read time values from JWKs. If so, augment section 8.2.

AdminKeyName: The AdminKeyName field contains a human-readable string meant to identify the key for the user. Implementations can use this field to uniquely identify rows in the key table. The same string can be used on the local system and peer systems, but this is not required.

LocalKeyName: The LocalKeyName field contains a string identifying the key. It can be used to retrieve the key in the local database when received in a message. For SMKs, this is the value of the 'id' attribute value of the <e2e/> element (see Section 6.3).

PeerKeyName: PeerKeyName is not used as the name is the same at each end point.

Peers: This field lists the full JID of each peer systems that has this key in their database. The peer name is read from the 'from' attribute of the wrapping stanza (see Section 6.3).

Interfaces: This field is not used and must be set to "all".

Protocol: The Protocol field identifies XMPP the protocol where this key may be used to provide cryptographic protection. (TODO: registry entry for the protocol?)

ProtocolSpecificInfo: This field is not used and must be be empty.

KDF: The KDF field is not used and must be set to "none". (TODO: define a use for this field?)

AlgID: The AlgID field indicates which cryptographic algorithm to be used with the security protocol for the specified peer or peers. Such an algorithm can be an encryption algorithm and mode (e.g., AES-128-CBC), an authentication algorithm (e.g., HMAC-SHA1-96 or AES-128-CMAC), or any other symmetric cryptographic algorithm needed by a security protocol. (TODO: identify source for algorithm strings)

Key: The Key field contains a long-lived symmetric cryptographic key in the format of a lower-case hexadecimal string. The size of the Key depends on the KDF and the AlgID. For instance, a KDF=none and AlgID=AES128 requires a 128-bit key, which is represented by 32 hexadecimal digits.

Direction: The Direction field indicates whether this key may be used for inbound traffic, outbound traffic, both, or whether the key has been disabled and may not currently be used at all. The supported values are "in", "out", "both", and "disabled", respectively.

SendLifetimeStart: The SendLifetimeStart field specifies the earliest date and time in Coordinated Universal Time (UTC) at which this key should be considered for use when sending traffic. The format is YYYYMMDDHHSSZ, where four digits specify the year, two digits specify the month, two digits specify the day, two digits specify the hour, two digits specify the minute, and two digits specify the second. The "Z" is included as a clear indication that the time is in UTC.

SendLifeTimeEnd: The SendLifeTimeEnd field specifies the latest date and time at which this key should be considered for use when sending traffic. The format is the same as the SendLifetimeStart field.

AcceptLifeTimeStart: The AcceptLifeTimeStart field specifies the earliest date and time in Coordinated Universal Time (UTC) at which this key should be considered for use when processing received traffic. The format is YYYYMMDDHHSSZ, where four digits specify the year, two digits specify the month, two digits specify the day, two digits specify the hour, two digits specify the minute, and two digits specify the second. The "Z" is included as a clear indication that the time is in UTC.

AcceptLifeTimeEnd: The AcceptLifeTimeEnd field specifies the latest date and time at which this key should be considered for use when processing the received traffic. The format of this field is identical to the format of AcceptLifeTimeStart.

6. Encryption

6.1. Determining Support

If an agent supports receiving end-to-end object encryption, it MUST advertise that fact in its responses to [XEP-0030] information ("disco#info") requests by returning a feature of "urn:ietf:params:xml:ns:xmpp-e2e:6:encryption".

```
<iq xmlns='jabber:client'
    id='disco1'
    to='romeo@montegue.lit/garden'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6:encryption'/>
    ...
  </query>
</iq>
```

To facilitate discovery, an agent SHOULD also include [XEP-0115] information in any directed or broadcast presence updates.

6.2. Encrypting XMPP Stanzas

The process that a sending agent follows for securing stanzas is the same regardless of the form of stanza (i.e., <iq/>, <message/>, or <presence/>).

6.2.1. Prerequisites

First, the sending agent prepares and retains the following:

- The JID of the sender (i.e. its own JID). This SHOULD be the bare JID (localpart@domainpart).

- The JID of the recipient. This SHOULD be the bare JID (localpart@domainpart).

- A Session Master Key (SMK). The SMK MUST have a length at least equal to that required by the key wrapping algorithm in use and MUST be generated randomly. See [RFC4086] for considerations on generating random values.

o A SMK identifier (SID). The SID MUST be unique for a given (sender, recipient, SMK) tuple, and MUST NOT be derived from SMK itself.

6.2.2. Process

For a given plaintext stanza (S), the sending agent performs the following:

1. Ensures the plaintext stanza is fully qualified, including the proper namespace declarations (e.g., contains the attribute 'xmlns' set to the value "jabber:client" for 'jabber:client' stanzas defined in [RFC6120]).

2. Notes the current UTC date and time (N) when this stanza is constructed, formatted as described under Section 10.

3. Constructs a forwarding envelope (M) using a <forwarded/> element qualified by the "urn:xmpp:forward:0" namespace (as defined in [XEP-0297]) as follows:

   * The child element <delay/> qualified by the "urn:xmpp:delay" namespace (as defined in [XEP-0203]) with the attribute 'stamp' set to the UTC date and time value N

   * The plaintext stanza S

4. Converts the forwarding envelope (M) to a UTF-8 encoded string (M'), optionally removing line breaks and other insignificant whitespace between elements and attributes, i.e. M' = UTF8-encode(M). We call M' a "stanza-string" because for purposes of encryption and decryption it is treated not as XML but as an opaque string (this avoids the need for complex canonicalization of the XML input).

5. Generates a Content Master Key (CMK). The CMK MUST have a length at least equal to that required by the content encryption algorithm in use and MUST be generated randomly. See [RFC4086] for considerations on generating random values.

6. Generates any additional unprotected block cipher factors (IV); e.g., initialization vector/nonce. A sending agent MUST ensure that no two sets of factors are used with the same CMK, and SHOULD NOT reuse such factors for other stanzas.

7. Performs the message encryption steps from [JOSE-JWE] to generate the JWE Header (H), JWE Encrypted Key (E), JWE Ciphertext (C), and JWE Integrity Value (I); using the following inputs:

   * The 'alg' property is set to an appropriate key wrapping algorithm (e.g., "A256KW" or "A128KW"); recipients use the key request process in Section 8 to obtain the SMK.

   * The 'enc' property is set to the intended content encryption algorithm.

   * SMK as the key for CMK Encryption.

   * CMK as the JWE Content Master Key.

   * IV as the JWE Initialization Vector.

   * M' as the plaintext content to encrypt.

8. Constructs an <e2e/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:

   * The attribute 'type' set to the value "enc".

   * The attribute 'id' set to the identifier value SID.

   * The child element <encheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].

* The child element <cmk/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character as E, encoded base64url as per [RFC4648].

* The child element <iv/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character as IV, encoded base64url as per [RFC4648].

* The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as C, encoded base64url as per [RFC4648].

* The child element <mac/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].

9. Sends the <e2e/> element as the payload of a stanza that SHOULD match the stanza from step 1 in kind (e.g., <message/>), type (e.g., "chat"), and addressing (e.g., to="romeo@montague.net" from="juliet@capulet.net/balcony"). If the original stanza (S) has a value for the 'id' attribute, this stanza MUST NOT use the same value for its 'id' attribute.

6.3. Decrypting XMPP Stanzas

6.3.1. Protocol Not Understood

If the receiving agent does not understand the protocol, it 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 [RFC6120].

NOTE: If the inbound stanza is an <iq/>, the receiving agent MUST return an error to the sending agent, to comply with the exchanging of IQ stanzas in [RFC6121].

6.3.2. Process

Upon receipt of an encrypted stanza, the receiving agent performs the following:

1. Determines if a valid SMK is available, associated with the SID specified by the 'id' attribute value of the <e2e/> element and

the sending agent JID specified by the 'from' attribute of the wrapping stanza. If the receiving agent does not already have the SMK, it requests it according to Section 8.

2. Performs the message decryption steps from [JOSE-JWE] to generate the plaintext forwarding envelope string M', using the following inputs:

   * The JWE Header (H) from the <encheader/> element's character data content.

   * The JWE Encrypted Key (E) from the <cmk/> element's character data content.

   * The JWE Initialization Vector/Nonce (I) from the <iv/> element's character data content.

   * The JWE Ciphertext (C) from the <data/> element's character data content.

   * The JWE Integrity Value (I) from the <mac/> element's character data content.

3. Converts the forwarding envelope UTF-8 encoded string M' into XML element (M).

4. Obtains the UTC date and time (N) from the <delay/> child element, and verifies it is within the accepted range, as specified in Section 10.

5. Obtains the plaintext stanza (S), which is a child element node of M; the stanza MUST be fully qualified with proper namespace declarations for XMPP stanzas, to help distinguish it from other content within M.

   .

6.3.3. Insufficient Information

At step 1, if the receiving agent is unable to obtain the CMK, or the receiving agent could not otherwise determine the additional information, it MAY return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <insufficient-information/>:

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <encheader>[XML character data]</encheader>
    <cmk>[XML character data]</cmk>
    <iv>[XML character data]</iv>
    <data>[XML character data]</data>
    <mac>[XML character data]</mac>
  </e2e>
  <error type='modify'>
    <bad-request
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <insufficient-information
        xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and 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).

6.3.4. Failed Decryption

At step 2, if the receiving agent is unable to successfully decrypt the stanza, the receiving agent SHOULD return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <decryption-failed/> (previously defined in [RFC3923]):

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <encheader>[XML character data]</encheader>
    <cmk>[XML character data]</cmk>
    <iv>[XML character data]</iv>
    <data>[XML character data]</data>
    <mac>[XML character data]</mac>
  </e2e>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <decryption-failed xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and 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).

6.3.5. Timestamp Not Acceptable

At step 4, if the stanza is successfully decrypted but the timestamp fails the checks outlined in Section 10, the receiving agent MAY return a <not-acceptable/> error to the sender (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <bad-timestamp/> (previously defined in [RFC3923]):

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <encheader>[XML character data]</encheader>
    <cmk>[XML character data]</cmk>
    <iv>[XML character data]</iv>
    <data>[XML character data]</data>
    <mac>[XML character data]</mac>
  </e2e>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-timestamp xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

6.3.6. Successful Decryption

If the receiving agent successfully decrypted the payload, it MUST NOT return a stanza error.

If the payload is an <iq/> of type "get" or "set", and the response to this <iq/> is of type "error", the receiving agent MUST send the encrypted response wrapped in an <iq/> of type "result", to prevent exposing information about the payload.

6.4. Example - Securing a Message

NOTE: unless otherwise indicated, all line breaks are included for readability.

The sending agent begins with the plaintext version of the <message/> stanza 'S':

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         to='romeo@montegue.lit'
         type='chat'>
  <thread>35740be5-b5a4-4c4e-962a-a03b14ed92f4</thread>
  <body>
    But to be frank, and give it thee again.
    And yet I wish but for the thing I have.
    My bounty is as boundless as the sea,
    My love as deep; the more I give to thee,
    The more I have, for both are infinite.
  </body>
</message>
```

and the following prerequisites:

o Sender JID as "juliet@capulet.lit/balcony"

o Recipient JID as "romeo@montegue.lit"

o Session Master Key (SMK) as (base64 encoded) "xWtdjhYsH4Va_9SfYSefsJfZu03m5RrbXo_UavxxeU8"

o SMK identifier (SID) as "835c92a8-94cd-4e96-b3f3-b2e75a438f92"

The sending agent performs steps 1, 2, and 3 from Section 6.2.2 to generate the envelope:

```
<forwarded xmlns='urn:xmpp:forward:0'>
  <delay xmlns='urn:xmpp:delay'
         stamp='1492-05-12T20:07:37.012Z'/>
  <message xmlns='jabber:client'
           from='juliet@capulet.lit/balcony'
           to='romeo@montegue.lit'
           type='chat'>
    <thread>35740be5-b5a4-4c4e-962a-a03b14ed92f4</thread>
    <body>
      But to be frank, and give it thee again.
      And yet I wish but for the thing I have.
      My bounty is as boundless as the sea,
      My love as deep; the more I give to thee,
      The more I have, for both are infinite.
    </body>
  </message>
</forwarded>
```

Then the sending agent performs steps 4 through 7 (with Content Master Key as "LViSXX0Jx-I3v1zY1-KcGeivmWKuq0QE_71ywQGU6OhlM2NoQo1zHi77zI3ieIUh7Wb1S3kXmNily0_FZoIG7A", base64url encoded) to generate the [JOSE-JWE] outputs:

JWE Header

```
{
  "alg":"A256KW",
  "enc":"A256CBC+HS512",
  "kid":"835c92a8-94cd-4e96-b3f3-b2e75a438f92"
}
```

JWE Encrypted Key

2tsmGH-WQdBxxJEs3d6LB2ovK6e1_9C1ogizJ9c6OvLmC6IeilHZ2Mimq2AElgI
ploz0VQv5LOH9ST93WvvhVzMHSfx0Cwl0

JWE Initialization Vector

ncOH4MsHT9HlJxnirx4qwg

JWE Ciphertext

FkFc4xGTVkjn7ojtS0SUY8IWfqsQKEIAlvLaBKieqVX1PAlq1ZjPp4TZC2I2eh7
01Lef3iRuNZd1nlgP2aREyHYCpE3FAelUoVG90B1FrJMnDUKAka7eb6GImamWPf
9onV-m5-GcUpejO9f1oPi-rwHzp475UPdAeKq5Z4zds8yXhQP-XyJbCPTtM-UQC
2-_q-3EKBHC4jM3qWDxVJ0JbIif3fCVRowzJh4AOB84YrfvkgUjMItqQPg2H6QB
NqGUspLI634lM8R-mhGciDZX2Jh_nKoXLAf5GCnvL9PlI7OdFqocPBIIPpjNrgX
_Z4PFjeq7ILx98GhVkryLYU9HVOFPCYci-lF9nfw1geliLfkoj5QZyi4J2SOtYa
O_zPmQvCXaUREqPf5UDAlgvc50a4ByYnNbkWSbhZ5Z388s8ELzPSE9XypdgP-1c
SyRke7V8iGe4eHNsm01TgWILYOFK4mYAM52OTitJxmQtmRp6izY5ZFdH9f_WdoB
1RXmGEZydvL-estcjx5ghsV3gktedIl0HA4R_M_N5TFIwv7hiisyRLi2aQtyFbE
7pZ6Oz-cYsLc4qFfXbb13U9a2-Byul8hm_E2b3m4GMhmsCiROm-uht9Ek4h9BIx
FhDKPr-htOXc93-uQNZlAQfkITAKlJfQ

JWE Integrity Value

Aj8lKdPMDE4U82UAhDJBaRrl3USmuzS2hfFOe_OBEv8

Then the sending agent performs steps 8 and 9, and sends the following:

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       type='enc'
       id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <encheader>
      eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2Q0JDK0hTNTEyIiwia2lkI
      joiODM1YzkyYTgtOTRjZC00ZTk2LWIzZjMtYjJlNzVhNDM4ZjkyIn0
    </encheader>
    <cmk>
      2tsmGH-WQdBxxJEs3d6LB2ovK6e1_9C1ogizJ9c6OvLmC6IeilHZ2Mimq
      2AElgIploz0VQv5LOH9ST93WvvhVzMHSfx0Cwl0
    </cmk>
    <iv>
      ncOH4MsHT9HlJxnirx4qwg
    </iv>
    <data>
      FkFc4xGTVkjn7ojtS0SUY8IWfqsQKEIAlvLaBKieqVX1PAlq1ZjPp4TZC
      2I2eh701Lef3iRuNZd1nlgP2aREyHYCpE3FAelUoVG90B1FrJMnDUKAka
      7eb6GImamWPf9onV-m5-GcUpejO9f1oPi-rwHzp475UPdAeKq5Z4zds8y
      XhQP-XyJbCPTtM-UQC2-_q-3EKBHC4jM3qWDxVJ0JbIif3fCVRowzJh4A
      OB84YrfvkgUjMItqQPg2H6QBNqGUspLI634lM8R-mhGciDZX2Jh_nKoXL
      Af5GCnvL9PlI7OdFqocPBIIPpjNrgX_Z4PFjeq7ILx98GhVkryLYU9HVO
      FPCYci-lF9nfw1geliLfkoj5QZyi4J2SOtYaO_zPmQvCXaUREqPf5UDAl
      gvc50a4ByYnNbkWSbhZ5Z388s8ELzPSE9XypdgP-1cSyRke7V8iGe4eHN
      sm01TgWILYOFK4mYAM52OTitJxmQtmRp6izY5ZFdH9f_WdoB1RXmGEZyd
      vL-estcjx5ghsV3gktedIl0HA4R_M_N5TFIwv7hiisyRLi2aQtyFbE7pZ
      6Oz-cYsLc4qFfXbb13U9a2-Byul8hm_E2b3m4GMhmsCiROm-uht9Ek4h9
      BIxFhDKPr-htOXc93-uQNZlAQfkITAKlJfQ
    </data>
    <mac>
      Aj8lKdPMDE4U82UAhDJBaRrl3USmuzS2hfFOe_OBEv8
    </mac>
  </e2e>
</message>
```


7. Signatures

7.1. Determining Support

If an agent supports receiving end-to-end object signatures, it MUST advertise that fact in its responses to [XEP-0030] information ("disco#info") requests by returning a feature of "urn:ietf:params:xml:ns:xmpp-e2e:6:signatures".

```
<iq xmlns='jabber:client'
    id='disco1'
    to='romeo@montegue.lit/garden'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6:signatures'/>
    ...
  </query>
</iq>
```

To facilitate discovery, an agent SHOULD also include [XEP-0115] information in any directed or broadcast presence updates.

7.2. Signing XMPP Stanzas

The basic process that a sending agent follows for authenticating stanzas is the same regardless of the kind of stanza (i.e., <iq/>, <message/>, or <presence/>).

7.2.1. Process

For a given plaintext stanza (S), the sending agent performs the following:

1. Ensures the plaintext stanza is fully qualified, including the proper namespace declarations (e.g., contains the attribute 'xmlns' set to the value "jabber:client" for 'jabber:client' stanzas defined in [RFC6120]).

2. Notes the current UTC date and time (N) when this stanza is constructed, formatted as described under Section 10.

3. Constructs a forwarding envelope (M) using a <forwarded/> element qualified by the "urn:xmpp:forward:0" namespace (as defined in [XEP-0297]) as follows:

   * The child element <delay/> qualified by the "urn:xmpp:delay" namespace (as defined in [XEP-0203]) with the attribute 'stamp' set to the UTC date and time value N

   * The plaintext stanza S

4. Converts the forwarding envelope (M) to a UTF-8 encoded string (M'), optionallly removing line breaks and other insignificant whitespace between elements and attributes, i.e. M' = UTF8-encode(M). We call M' a "stanza-string" because for purposes of encryption and decryption it is treated not as XML but as an opaque string (this avoids the need for complex canonicalization of the XML input).

5. Chooses a private asymmetric key (PK) for which the sending agent has published the corresponding public key to the intended recipients.

6. Performs the message signatures steps from [JOSE-JWS] to generate the JWS Header (H) and JWS Signature (I); using the following inputs:

   * The 'alg' property is set to an appropriate signature algorithm for PK (e.g., "R256").

   * M' as the JWS Payload.

7. Constructs an <e2e/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:

   * The attribute 'type' set to the value "sig"

   * The child element <sigheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].

   * The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as M', encoded base64url as per [RFC4648].

   * The child element <sig/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].

8. Sends the <e2e/> element as the payload of a stanza that SHOULD match the stanza from step 1 in kind (e.g., <message/>), type (e.g., "chat"), and addressing (e.g., to="romeo@montegue.lit" from="juliet@capulet.lit/balcony"). If the original stanza (S) has a value for the 'id' attribute, this stanza SHOULD NOT use the same value for its "id" attribute.

7.3. Verifying Signed XMPP Stanzas

7.3.1. Protocol Not Understood

If the receiving agent does not understand the protocol, it 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 [RFC6120].

NOTE: If the inbound stanza is an <iq/>, the receiving agent MUST return an error to the sending agent, to comply with the exchanging of IQ stanzas in [RFC6121].

7.3.2. Process

Upon receipt of a signed stanza, the receiving agent performs the following:

1. Ensures it has appropriate materials to verify the signature, which generally means ensuring that it possesses one or more public keys for the sending agent (if one is not provided as part of the JWS Header).

2. Performs the message validation steps from [JOSE-JWS], with the following inputs:

   * The JWS Header H from the <sigheader/> element's character data content.

   * The JWS payload M' from the <data/> element's character data content.

   * The JWS Signature from the <sig/> element's character data content.

3. Converts the forwarding envelope UTF-encoded string M' into XML element M.

4. Obtains the UTC date and time N from the <delay/> child element, and verifies it is within the accepted range, as specified in Section 10.

5. Obtains the plaintext stanza S, which is a child element node of M; the stanza MUST be fully qualified with the proper namespace declrations from XMPP stanzas, to help distinguish it from other content within M.

7.3.3. Insufficient Information

At step 1, if the receiving agent does not have the key used to sign the stanza, or the receiving agent could not otherwise determine it, it MAY return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <insufficient-information/>:

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       type='sig'>
    <sigheader>[XML character data]</sigheader>
    <data>[XML character data]</data>
    <sig>[XML character data]</sig>
  </e2e>
  <error type='modify'>
    <bad-request
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <insufficient-information
        xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and 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 verified).

7.3.4. Failed Verification

At step 2, if the receiving agent is unable to successfully verify the stanza, the receiving agent SHOULD return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <verification-failed/>:

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       type='sig'>
    <sigheader>[XML character data]</sigheader>
    <data>[XML character data]</data>
    <sig>[XML character data]</sig>
  </e2e>
  <error type='modify'>
    <bad-request
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <verification-failed
        xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and 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 verified).

7.3.5. Timestamp Not Acceptable

At step 4, if the stanza is successfully verified but the timestamp fails the checks outlined in Section 10, the receiving agent MAY return a <not-acceptable/> error to the sender (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <bad-timestamp/> (previously defined in [RFC3923]):

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='fJZd9WFIIwNjFctT'
         to='romeo@montegue.lit/garden'
         type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       type='sig'>
    <sigheader>[XML character data]</sigheader>
    <data>[XML character data]</data>
    <sig>[XML character data]</sig>
  </e2e>
  <error type='modify'>
    <not-acceptable
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-timestamp
        xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'/>
  </error>
</message>
```

7.3.6. Successful Verification

If the receiving agent successfully verified the payload, it SHOULD NOT return a stanza error. However, if the signed stanza is an <iq/> of type "get" or "set", the response MAY be sent unsigned if the receiving agent does not have an appropriate public-private key-pair.

Otherwise, the receiving agent SHOULD send the <iq/> response signed as per Section 7.2.1, with the 'type' attribute set to the value "result", even if the response to the signed <iq/> stanza is of type "error". The error applies to the signed stanza, not the wrapping stanza.

7.4. Example - Signing a Message

NOTE: unless otherwise indicated, all line breaks are included for readability.

The sending agent beings with the plaintext version of <message/> stanza 'S':

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         to='romeo@montegue.lit'
         type='chat'>
  <thread>35740be5-b5a4-4c4e-962a-a03b14ed92f4</thread>
  <body>
    But to be frank, and give it thee again.
    And yet I wish but for the thing I have.
    My bounty is as boundless as the sea,
    My love as deep; the more I give to thee,
    The more I have, for both are infinite.
  </body>
</message>
```

Then the sending agent performs steps 1, 2, and 3 from Section 7.2.1 generate the envelope M:

```
<forwarded xmlns='urn:xmpp:forward:0'>
  <delay xmlns='urn:xmpp:delay'
         stamp='1492-05-12T20:07:37.012Z'/>
  <message xmlns='jabber:client'
           from='juliet@capulet.lit/balcony'
           to='romeo@montegue.lit'
           type='chat'>
    <thread>35740be5-b5a4-4c4e-962a-a03b14ed92f4</thread>
    <body>
      But to be frank, and give it thee again.
      And yet I wish but for the thing I have.
      My bounty is as boundless as the sea,
      My love as deep; the more I give to thee,
      The more I have, for both are infinite.
    </body>
  </message>
</forwarded>
```

Then the sending agent performs steps 4, 5, and 6 to generate the [JOSE-JWS] outputs:

JWS Header (before base64url encoding)

```
{
  "alg":"RS512",
  "kid":"juliet@capulet.lit"
}
```

JWS Payload

PGZvcndhcmRlZCB4bWxucz0idXJuOnhtcHA6Zm9yd2FyZDowIj48ZGVsYXkgeG1
sbnM9InVybjp4bXBwOmRlbGF5IiBzdGFtcD0iMTQ5Mi0wNS0xMlQyMDowNzozNy
4wMTJaIi8-PG1lc3NhZ2UgeG1sbnM9ImphYmJlcjpjbGllbnQiIGZyb209Imp1b
GlldEBjYXB1bGV0LmxpdC9iYWxjb255IiB0bz0icm9tZW9AbW9udGVndWUubGl0
IiB0eXBlPSJjaGF0Ij48dGhyZWFkPjM1NzQwYmU1LWI1YTQtNGM0ZS05NjJhLWE
wM2IxNGVkOTJmNDwvdGhyZWFkPjxib2R5PkJ1dCB0byBiZSBmcmFuaywgYW5kIG
dpdmUgaXQgdGhlZSBhZ2Fpbi4gQW5kIHlldCBJIHdpc2ggYnV0IGZvciB0aGUgd
GhpbmcgSSBoYXZlLiBNeSBib3VudHkgaXMgYXMgYm91bmRsZXNzIGFzIHRoZSBz
ZWEsIE15IGxvdmUgYXMgZGVlcDsgdGhlIG1vcmUgSSBnaXZlIHRvIHRoZWUsIFR
oZSBtb3JlIEkgaGF2ZSwgZm9yIGJvdGggYXJlIGluZmluaXRlLjwvYm9keT48L2
1lc3NhZ2U-PC9mb3J3YXJkZWQ-

JWS Signature

YPfGouD50j0C_C-RneawG0jxXWDXgBkN3FJz6eaBFIPCh3hopiwtwKir7Yamvgt
OrqhXx2pcu-70caGi6mKKLWvpdwdJ3nEnhdjPOd3CmLdaK_PBAMtIt8d3155hdl
qNxSMsJN7PxmNLNwJhbksAsI-2TcCQsuxdIPXh6hcqBm44BpVio6AoRPqwF06XZ
MMBMOMnEFcV6Ht20wCK1BEGgOmN3KYPbwKeTctG8HKPAh25_K66aEXT66lI19uW
j1fGFJ79QQHUhc5y9pSKmpK7HKruPMRyrvpzBSfUhcb62nLXhM-LzY5taaDECzi
fCi-IxySBtJJtPCqYAYW_IbrRFg

Then the sending agent performs steps 7 and 8 and sends the following:

```
<message xmlns='jabber:client'
         from='juliet@capulet.lit/balcony'
         id='6aAWpciGV98qaegk'
         to='romeo@montegue.lit'
         type='cat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
       type='sig'>
    <sigheader>
      eyJhbGciOiJSUzUxMiIsImtpZCI6Imp1bGlldEBjYXB1bGV0LmxpdCJ9
    </sigheader>
    <data>
      PGZvcndhcmRlZCB4bWxucz0idXJuOnhtcHA6Zm9yd2FyZDowIj48ZGVsY
      XkgeG1sbnM9InVybjp4bXBwOmRlbGF5IiBzdGFtcD0iMTQ5Mi0wNS0xMl
      QyMDowNzozNy4wMTJaIi8-PG1lc3NhZ2UgeG1sbnM9ImphYmJlcjpjbGl
      lbnQiIGZyb209Imp1bGlldEBjYXB1bGV0LmxpdC9iYWxjb255IiB0bz0i
      cm9tZW9AbW9udGVndWUubGl0IiB0eXBlPSJjaGF0Ij48dGhyZWFkPjM1N
      zQwYmU1LWI1YTQtNGM0ZS05NjJhLWEwM2IxNGVkOTJmNDwvdGhyZWFkPj
      xib2R5PkJ1dCB0byBiZSBmcmFuaywgYW5kIGdpdmUgaXQgdGhlZSBhZ2F
      pbi4gQW5kIHlldCBJIHdpc2ggYnV0IGZvciB0aGUgdGhpbmcgSSBoYXZl
      LiBNeSBib3VudHkgaXMgYXMgYm91bmRsZXNzIGFzIHRoZSBzZWEsIE15I
      GxvdmUgYXMgZGVlcDsgdGhlIG1vcmUgSSBnaXZlIHRvIHRoZWUsIFRoZS
      Btb3JlIEkgaGF2ZSwgZm9yIGJvdGggYXJlIGluZmluaXRlLjwvYm9keT4
      8L21lc3NhZ2U-PC9mb3J3YXJkZWQ-
    </data>
    <sig>
      YPfGouD50j0C_C-RneawG0jxXWDXgBkN3FJz6eaBFIPCh3hopiwtwKir7
      YamvgtOrqhXx2pcu-70caGi6mKKLWvpdwdJ3nEnhdjPOd3CmLdaK_PBAM
      tIt8d3155hdlqNxSMsJN7PxmNLNwJhbksAsI-2TcCQsuxdIPXh6hcqBm4
      4BpVio6AoRPqwF06XZMMBMOMnEFcV6Ht20wCK1BEGgOmN3KYPbwKeTctG
      8HKPAh25_K66aEXT66lI19uWj1fGFJ79QQHUhc5y9pSKmpK7HKruPMRyr
      vpzBSfUhcb62nLXhM-LzY5taaDECzifCi-IxySBtJJtPCqYAYW_IbrRFg
    </sig>
  </e2e>
</message>
```


8. Requesting Session Keys

Because of the dynamic nature of XMPP stanza routing, the protocol does not exchange session keys as part of the encrypted stanza. Instead, a separate protocol is used by receiving agents to request a particular session key from the sending agent.

8.1. Request Process

Before a SMK can be requested, the receiving agent MUST have at least one public key for which it also has the private key. The public key(s) are provided to the sending agent as part of this process.

To request a SMK, the receiving agent performs the following:

1. Constructs a [JOSE-JWK] JWK Set (KS), containing information about each public key the requesting agent wishes to use. Each key SHOULD include a value for the property 'kid' which uniquely identifies it within the context of all provided keys. Each key MUST include a value for the property 'kid' if any two keys use the same algorithm.

2. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:

   * The attribute 'id' set to the SMK identifier value SID.

   * The child element <pkey/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as KS, encoded base64url as per [RFC4648].

3. Sends the <keyreq/> element as the payload of an <iq/> stanza with the attribute 'type' set to "get", the attribute 'to' set to the full JID of the original encrypted stanza's sender, and the attribute 'id' set to an opaque string value the receiving agent uses to track the <iq/> response.

8.2. Accept Process

If the sending agent approves the request, it performs the following steps:

1. Generate a JSON Web Key (JWK) representing the symmetric SMK (according to [JOSE-JWK]):

   * The "kty" parameter MUST be "oct".

   * The "kid" parameter MUST be the SID.

   * The "k" parameter MUST be the SMK, encoded as base64url.

   * The "alg" parameter, if present, MUST be set to the algorithm in use for encrypting messages from Section 6.2.

   * The "use" parameter, if present, MUST be set to "enc".

2. Chooses a key (PK) from the keys provided via KS, and notes its identifier value 'kid'.

3. Protects the SMK using the process outlined in [JOSE-KEYPROTECT] to generate the JWE Header (H), JWE Encrypted Key (E), JWE Initialization Vector (IV), JWE Ciphertext (C), and JWE Integrity Value (I); using the following inputs:

   * The 'alg' property is set to an algorithm appropriate for the chosen PK (e.g., "RSA-OAEP" for a "RSA" key).

   * The 'enc' property is set to the intended content encryption algorithm.

   * A randomly generated CMK. See [RFC4086] for considerations on generating random values.

   * A randomly generated initialization vector. See [RFC4086] for considerations on generating random values.

   * SMK, formatted as a JWK as above.

4. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:

   * The attribute 'id' set to the SMK Identifier (SID).

* The child element <encheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].

* The child element <cmk/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as E, encoded base64url as per [RFC4648].

* The child element <iv/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as IV, encoded base64url as per [RFC4648].

* The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as C, encoded base64url as per [RFC4648].

* The child element <mac/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].

5. Sends the <keyreq/> element as the payload of an <iq/> stanza with the attribute 'type' set to "result", the attribute 'to' set to the full JID from the request <iq/>'s 'from' attribute, and the attribute 'id' set to the value of the request <iq/>'s 'id' attribute.

8.3. Error Conditions

If the sending agent does not approve the request, it sends an <iq/> stanza of type "error" and containing the reason for denying the request:

o <forbidden/>: the key request is made by an entity that is not authorized to decrypt stanzas from the sending agent and/or for the indicated SID.

o <item-not-found/>: the requested SID is no longer valid.

- <not-acceptable/>: the key request did not contain any keys the sending agent understands.

8.4. Example of Successful Key Request

NOTE: unless otherwise indicated, all line breaks are included for readability.

To begin a key request, the receiving agent performs step 1 from Section 8.1 to generate the [JOSE-JWK]:


```
{
  "keys": [{
    "kty":"RSA",
    "kid":"romeo@montegue.lit/garden",
    "n":"vtqejkMF01h8oKEaHfHEYO0C2jM7eISbbSvNs0SNItYWO6GbjpJf
    N4ldXw2vpVRdysnwU3zk6o2_SD0YCH1WgeuI0QK1knMTDdNSXx52e1c4BTw
    hlA8iHuutTWmpBqesn1GNZmqB3jYsJOkVBYwCJtkB9APaBvk0itlRtizjCf
    1HHnau7nGStyshgu8-srxi_d8rC5TTLSB_zT1i6fP8fwDloemXOtC0U65by
    5P-1ZHxaf_bD8fpjps6gwSgdkZKMJAI0bOWZWuMpp2ntqa0wLB7Ndxb2Ijr
    eog_s5ssAoSiXDVdoswSbp36ZP-1lnCk2j-vZ4qbhaFg5bZtgt-gwQ",
    "e":"AQAB"
  }]
}
```


Then the receiving agent performs step 2 to generate the <keyreq/>:


```
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
        id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <pkey>
    eyJrZXlzIjpbeyJrdHkiOiJSU0EiLCJraWQiOiJyb21lb0Btb250ZWd1ZS5
    saXQvZ2FyZGVuIiwibiI6InZ0cWVqa01GMDFoOG9LRWFIZkhFWU8wQzJqTT
    dlSVNiYlN2TnMwU05JdFlXTzZHYmpwSmZONGxkWHcydnBWUmR5c253VTN6a
    zZvMl9TRDBZQ0gxV2dldUkwUUsxa25NVERkTlNYeDUyZTFjNEJUd2hsQThp
    SHV1dFRXbXBCcWVzbjFHTlptcUIzallzSk9rVkJZd0NKdGtCOUFQYUJ2azB
    pdGxSdGl6akNmMUhIbmF1N25HU3R5c2hndTgtc3J4aV9kOHJDNVRUTFNCX3
    pUMWk2ZlA4ZndEbG9lbVhPdEMwVTY1Ynk1UC0xWkh4YWZfYkQ4ZnBqcHM2Z
    3dTZ2RrWktNSkFJMGJPV1pXdU1wcDJudHFhMHdMQjdOZHhiMklqcmVvZ19z
    NXNzQW9TaVhEVmRvc3dTYnAzNlpQLTFsbkNrMmotdlo0cWJoYUZnNWJadGd
    0LWd3USIsImUiOiJBUUFCIn1dfQ
  </pkey>
</keyreq>
```


Then the receiving agent performs step 3 and sends the following:

```
<iq xmlns='jabber:client'
    from='romeo@montegue.lit/garden'
    id='xdJbWMA+'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
          id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <pkey>
      eyJrZXlzIjpbeyJrdHkiOiJSU0EiLCJraWQiOiJyb21lb0Btb250ZWd1Z
      S5saXQvZ2FyZGVuIiwibiI6InZ0cWVqa01GMDFoOG9LRWFIZkhFWU8wQz
      JqTTdlSVNiYlN2TnMwU05JdFlXTzZHYmpwSmZONGxkWHcydnBWUmR5c25
      3VTN6azZvMl9TRDBZQ0gxV2dldUkwUUsxa25NVERRkTlNYeDUyZTFjNEJU
      d2hsQThpSHV1dFRXbXBCcWVzbjFHTlptcUIzallzSk9rVkJZd0NKdGtCO
      UFQYUJ2azBpdGxSdGl6akNmMUhIbmF1N25HU3R5c2hndTgtc3J4aV9kOH
      JDNVRUTFNCX3pUMWk2ZlA4ZndEbG9lbVhPdEMwVTY1Ynk1UC0xWkh4YWZ
      fYkQ4ZnBqcHM2Z3dTZ2RrWktNSkFJMGJPV1pXdU1wcDJudHFhMHdMQjdO
      ZHhiMklqcmVvZ19zNXNzQW9TaVhEVmRvc3dTYnAzNlpQLTFsbkNrMmotd
      lo0cWJoYUZnNWJadGd0LWd3USIsImUiOiJBUUFCIn1dfQ
    </pkey>
  </keyreq>
</iq>
```


If the sending agent accepts this key request, it performs step 1 from Section 8.2 to generate JWK representation of the SMK:


```
{
  "kty":"oct",
  "kid":"835c92a8-94cd-4e96-b3f3-b2e75a438f92",
  "k":"xWtdjhYsH4Va_9SfYSefsJfZu03m5RrbXo_UavxxeU8"
}
```


Then the sending agent performs steps 2 and 3 to generate the protected SMK:

JWE Header (before base64url encoding)


```
{
  "alg":"RSA-OAEP",
  "kid":"romeo@montegue.lit/garden",
  "enc":"A256CBC+HS512",
  "cty":"application/jwk+json"
}
```

JWE Encrypted Key

hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4AAIk
rZJOtrPUqPZwYHjay3ggfgjVljJ_KGhgqI5cScIzaAQs0Pxep6FnrsnUrw09Sjv
2VRXOay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7GT35gZC9NgweX
3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F7NnOv9oLx1HtmfE3
_skkYtQoKMvMewLkIO88h325qCpWFdrLwPp63betCmewDJPaBdrp91rLchkXVo-
d2ueKkb59TxWjMx7esBdaxCAcDQ

JWE Initialization Vector

Ggiego8UiSsj7GgY94qOng

JWE Ciphertext

4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGAiMUs3RTviFO09choPhxJNlOj8KX8QIL
u4zZ-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpX173k6ZJV-sXGd4Mj9u7N0IqWQL
K5DMytv7XopsZsR9QFCDNGew

JWE Integrity Value

3GuaasWV0XGTBbRtNP6OQ14_cHL-ZJC1naDtU6EIecw

Then the sending agent performs step 4 to generate the <keyreq/>
response:

```
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
        id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <encheader>
    eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpdC9
    nYXJkZW4iLCJlbmMiOiJBMjU2Q0JDK0hTNTEyIiwiY3R5IjoiYXBwbGljYX
    Rpb24vandrK2pzb24ifQ
  </encheader>
  <cmk>
    hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4
    AAIkrZJOtrPUqPZwYHjay3ggfgjVljJ_KGhgqI5cScIzaAQs0Pxep6Fnrsn
    Urw09Sjv2VRXOay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7G
    T35gZC9NgweX3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F
    7NnOv9oLx1HtmfE3_skkYtQoKMvMewLkIO88h325qCpWFdrLwPp63betCme
    wDJPaBdrp91rLchkXVo-d2ueKkb59TxWjMx7esBdaxCAcDQ
  </cmk>
  <iv>
    Ggiego8UiSsj7GgY94qOng
  </iv>
  <data>
    4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGAiMUs3RTviFO09choPhxJNlOj8KX
    8QILu4zZ-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpX173k6ZJV-sXGd4Mj9u
    7N0IqWQLK5DMytv7XopsZsR9QFCDNGew
  </data>
  <mac>
    3GuaasWV0XGTBbRtNP6OQ14_cHL-ZJC1naDtU6EIecw
  </mac>
</keyreq>
```


Then the sending agent performs step 5 and sends the following:

```
<iq xmlns='jabber:client'
    from='juliet@capulet.lit/balcony'
    id='xdJbWMA+'
    to='romeo@montegue.lit/garden'
    type='result'>
  <keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
          id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <encheader>
      eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpdC9
      nYXJkZW4iLCJlbmMiOiJBMjU2Q0JDK0hTNTEyIiwiY3R5IjoiYXBwbGljYX
      Rpb24vandrK2pzb24ifQ
    </encheader>
    <cmk>
      hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4
      AAIkrZJOtrPUqPZwYHjay3ggfgjVljJ_KGhgqI5cScIzaAQs0Pxep6Fnrsn
      Urw09Sjv2VRXOay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7G
      T35gZC9NgweX3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F
      7NnOv9oLx1HtmfE3_skkYtQoKMvMewLkIO88h325qCpWFdrLwPp63betCme
      wDJPaBdrp91rLchkXVo-d2ueKkb59TxWjMx7esBdaxCAcDQ
    </cmk>
    <iv>
      Ggiego8UiSsj7GgY94qOng
    </iv>
    <data>
      4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGAiMUs3RTviFO09choPhxJNlOj8KX
      8QILu4zZ-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpX173k6ZJV-sXGd4Mj9u
      7N0IqWQLK5DMytv7XopsZsR9QFCDNGew
    </data>
    <mac>
      3GuaasWV0XGTBbRtNP6OQ14_cHL-ZJC1naDtU6EIecw
    </mac>
  </keyreq>
</iq>
```

9. Mulitple Operations

The individual processes for encrypting and signing can be nested; the output of each process a complete stanza that could then be performed with the other. An implementation MUST be able to process one level of nesting (e.g., an encrypted stanza nested within a signed stanza), and SHOULD handle multiple levels within reasonable limits for the receiving agent.

10. Inclusion and Checking of Timestamps

Timestamps are included to help prevent replay attacks. All timestamps MUST conform to [XEP-0082] and be presented as UTC with no offset, and SHOULD include the seconds and fractions of a second to

three digits. 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 agent:

- It MUST verify that the timestamp received is within an acceptable range of the current time. It is RECOMMENDED that implementations use an acceptable range of five minutes, although implementations MAY use a smaller acceptable range.

- 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.

Note the foregoing assumes the stanza is received while the receiving agent is online; see Section 12 for offline storage considerations.

11. Interaction with Stanza Semantics

The following limitations and caveats apply:

- Undirected <presence/> stanzas SHOULD NOT be encrypted. Such stanzas are delivered to anyone the sender has authorized, and can generate a large volume of key requests.

- Undirected <presence/> stanzas MAY be signed. However, note that signatures significantly increase the size of a stanza kind that is often multiplexed across to many XMPP entities; this could have large impacts on bandwidth and latency.

- Stanzas directed to multiplexing services (e.g., multi-user chat) SHOULD NOT be encrypted, unless the sender has established an acceptable trust relationship with the multiplexing service.

12. Interaction with Offline Storage

The server makes its best effort to deliver stanzas. When the receiving agent is offline at the time of delivery, the server might store the message until the recipient is next online (offline storage does not apply to <iq/> or <presence/> stanzas, only <message/> stanzas). The following need to be considered:

- If the sending agent is not also online when the message is delivered to the receiving agent from offline storage, then the decryption process fails for insufficient information as described in Section 6.3.3.

- When performing the timestamp checks in Section 10, if the server includes delayed delivery data as specified in [XEP-0203] for when the server received the message, then the receiving agent SHOULD use the delayed delivery timestmap rather than the current time.

13. Mandatory-to-Implement Cryptographic Algorithms

All algorithms that MUST be implemented for [JOSE-JWE] and [JOSE-JWS] also MUST be implemented for this specification. However, this specification further mandates the use of the following:

- MUST implement the "RSA1_5" JWE algorithm.

- MUST implement the "RS256" JWS algorithm.

14. Security Considerations

14.1. Storage of Encrypted Stanzas

The recipient's server might store any <message/> stanzas received until the recipient is next available; this duration could be anywhere from a few minutes to several months.

14.2. Re-use of Session Master Keys

A sender SHOULD NOT use the same SMK for stanzas intended for different recipients, as determined by the localpart and domainpart of the recipient's JID.

A sender MAY re-use a SMK for several stanzas to the same recipient. In this case, the SID remains the same, but the sending agent MUST

generate a new CMK and IV for each encrypted stanza. The sender SHOULD periodically generate a new SMK (and its associated SID); however, this specification does not mandate any specific algorithms or processes.

In the case of <message/> stanzas, a sending agent might generate a new SMK each time it generates a new ThreadID, as outlined in [XEP-0201].

15. IANA Considerations

15.1. XML Namespaces Name for e2e Data in XMPP

A number of URN sub-namespaces of encrypted and/or signed content for the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-e2e:6

Specification: RFC XXXX

Description: This is an XML namespace name of encrypted and/or signed content for the Extensible Messaging and Presence Protocol as defined [[this document]].

Registrant Contact: IESG, <iesg@ietf.org>

URI: urn:ietf:params:xml:ns:xmpp-e2e:6:encryption

Specification: RFC XXXX

Description: This is an XML namespace name signalling support for encrypted content for the Extensible Messaging and Presence Protocol as defined [[this document]].

Registrant Contact: IESG, <iesg@ietf.org>

URI: urn:ietf:params:xml:ns:xmpp-e2e:6:signatures

Specification: RFC XXXX

Description: This is an XML namespace name signalling support for signed content for the Extensible Messaging and Presence Protocol as defined [[this document]].

Registrant Contact: IESG, <iesg@ietf.org>

16. References

16.1. Normative References

[E2E-REQ] Saint-Andre, P., "Requirements for End-to-End Encryption in the Extensible Messaging and Presence Protocol (XMPP)", draft-saintandre-xmpp-e2e-requirements-01 (work in progress), March 2010.

[JOSE-JWA] Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-11 (work in progress), May 2013.

[JOSE-JWE] Jones, M., Rescola, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-11 (work in progress), May 2013.

[JOSE-JWK] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-11 (work in progress), December 2012.

[JOSE-JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-11 (work in progress), May 2013.

[JOSE-KEYPROTECT] Miller, M., "Using JSON Web Encryption (JWE) for Protecting JSON Web Key (JWK) Objects", draft-miller-jose-jwe-protected-jwk-00 (work in progress), February 2013.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 6121, March 2011.

[XEP-0030]
Eatmon, R., Hildebrand, J., Millard, P., and P. Saint-Andre, "Service Discovery", XSF XEP 0030, June 2006.

[XEP-0082]
Saint-Andre, P., "XMPP Date and Time Profiles", XSF XEP 0082, May 2003.

[XEP-0115]
Hildebrand, J., Troncon, R., and P. Saint-Andre, "Entity Capabilities", XSF XEP 0115, February 2008.

[XEP-0203]
Saint-Andre, P., "Delayed Delivery", XSF XEP 0203, September 2009.

[XEP-0297]
Wild, M. and K. Smith, "Stanza Forwarding", XSF XEP 0297, July 2012.

16.2. Informative References

[RFC3923] Saint-Andre, P., "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)", RFC 3923, October 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", RFC 4086, June 2005.

[XEP-0201]
Saint-Andre, P., Paterson, I., and K. Smith, "Best Practices for Message Threads", XSF XEP 0203, November 2010.

[Key-Table]
Housley, R., Polk, T., Hartman, S., and D. Zhang, "Database of Long-Lived Symmetric Cryptographic Keys", December 2013.

Appendix A. Schema for urn:ietf:params:xml:ns:xmpp-e2e:6

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:6'
```

```
    xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
    elementFormDefault='qualified'>

  <xs:element name='e2e'>
    <xs:complexType>
      <xs:attribute name='id' type='xs:string' use='optional'/>
      <xs:attribute name='type'use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NMTOKEN'>
            <xs:enumeration value='enc'/>
            <xs:enumeration value='sig'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:sequence>
        <xs:element ref='header' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='cmk' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='iv' minOccurs=1' maxOccurs='1'/>
        <xs:element ref='data' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='mac' minOccurs='1' maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='keyreq'>
    <xs:complexType>
      <xs:attribute name='id' type='xs:string' use='required'/>
      <xs:sequence>
        <xs:element ref='pkey' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='header' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='cmk' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='iv' minOccurs=1' maxOccurs='1'/>
        <xs:element ref='data' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='mac' minOccurs='1' maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='cmk'>
    <xs:complexType>
      <xs:simpleType>
        <xs:extension base='xs:string'>
        </xs:extension>
      </xs:simpleType>
    </xs:complexType>
  </xs:element>

  <xs:element name='iv'>
```

```
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
            </xs:extension>
          </xs:simpleType>
        </xs:complexType>
      </xs:element>

      <xs:element name='data'>
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
            </xs:extension>
          </xs:simpleType>
        </xs:complexType>
      </xs:element>

      <xs:element name='encheader'>
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
            </xs:extension>
          </xs:simpleType>
        </xs:complexType>
      </xs:element>

      <xs:element name='mac'>
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
            </xs:extension>
          </xs:simpleType>
        </xs:complexType>
      </xs:element>

      <xs:element name='pkey'>
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
            </xs:extension>
          </xs:simpleType>
        </xs:complexType>
      </xs:element>

      <xs:element name='sigheader'>
        <xs:complexType>
          <xs:simpleType>
            <xs:extension base='xs:string'>
```

```
           </xs:extension>
         </xs:simpleType>
       </xs:complexType>
     </xs:element>

     <xs:element name='bad-timestamp' type='empty'/>
     <xs:element name='decryption-failed' type='empty'/>
     <xs:element name='insufficient-information' type='empty'/>
     <xs:element name='verification-failed' type='empty'/>

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

   </xs:schema>
```

Appendix B. Acknowledgements

Thanks to Richard Barnes, Andrew Biggs, and Ben Schumacher for their feedback.

Authors' Addresses

Matthew Miller
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

Phone: +1-303-308-3204
Email: mamille2@cisco.com

Carl Wallace
Red Hound Software, Inc.

Email: carl@redhoundsoftware.com