XEP-0285: Encapsulating Digital Signatures in XMPP

Kurt Zeilenga
mailto:Kurt.Zeilenga@Isode.COM
xmpp:Kurt.Zeilenga@Isode.COM

2011-01-12
Version 0.3

Status	Type	Short Name
Deferred	Standards Track	N/A

This document provides a technical specification for Encapsulating Digital Signatures in the Extensible Messaging and Presence Protocol (XMPP).

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

This document is one of two proposals for digital signatures in XMPP. It is expected that only one of these proposals be progressed beyond Experimental on the Standards Track.

This document provides a technical specification for Digital Signatures in Extensible Messaging and Presence Protocol (XMPP [1]) based upon End-to-End Object Encryption (E2EEncrypt [2]) "work in progress".

The S/MIME approach defined in RFC 3923 [3] has never been implemented in XMPP clients to the best of our knowledge, but has some attractive features, especially the ability to store-and-forward a signed message at a user's server if the user is not online when the message is received (in the XMPP community this is called "offline storage" and the message is referred to as an "offline message"). The authors surmise that RFC 3923 has not been implemented mainly because it adds several new dependencies to XMPP clients, especially MIME (along with the CPIM and MSGFMT media types).

This document explores the possibility of an approach that is similar to but simpler than RFC 3923. Like the approach detailed in RFC 3923, the approach utilizes encapsulating digital signatures.

Like other encapsulating signature approaches (e.g., Current Jabber OpenPGP Usage (XEP-0027) [4]), this approach does not support *optimistic signing*.

2 Signing XMPP Stanzas

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

1. Constructs a cleartext version of the stanza, S.
2. Notes the current UTC date and time N when this stanza is constructed, formatted as described in Section 5.
3. Converts the stanza to a UTF-8, as defined by RFC 3629 [5], encoded string, optionally removing line breaks and other insignificant whitespace between elements and attributes, i.e., UTF8-encode(S) = S'. We call S' a "stanza-string" because for purposes of signing and verification it is treated not as XML but as an opaque string (this avoids the need for complex canonicalization of the XML input).

[1] Extensible Messaging and Presence Protocol (XMPP) <https://xmpp.org/>.
[2] End-to-End Object Encryption for the Extensible Messaging and Presence Protocol (XMPP), Miller, M. and P. Saint-Andre, work in progress <http://datatracker.ietf.org/doc/draft-miller-3923bis>.
[3] RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) <http://tools.ietf.org/html/rfc3923>.
[4] XEP-0027: Current Jabber OpenPGP Usage <https://xmpp.org/extensions/xep-0027.html>.
[5] RFC 3629: UTF-8, a transformation format of ISO 10646 <http://tools.ietf.org/html/rfc3629>.

4. Constructs a plaintext envelope (E) <plain/> qualified by the "urn:xmpp:signed:0" namespace as follows:
   - The attribute 'timestamp' set to the UTC date and time value N
   - The XML character data set to the base64-encoded form of S' (where the encoding adheres to the definition in Section 4 of BASE64 [6] and where the padding bits are set to zero). This encoding is necessary to preserve a canonicalized form of S'.
5. Converts the envelope (E) to a UTF-8 encoded string, optionally removing line breaks and other insignificant whitespace between elements and attributes, i.e., E' = UTF8-encode(E).
6. Produce a signature of UTF8-encoded envelope (E') using the intended signature algorithm. T = signature(E'). (This step is underspecified and will be expanded upon in later revision of this document.)
7. Base64-encodes T to produce the signature data T'.
8. Constructs an <signed/> element qualified by the "urn:xmpp:signed:0" namespace as follows:
   - The child element <signature> (implicitly qualified by the "urn:xmpp:signed:0" namespace) as follows:
     - The attribute 'algorithm' set to a string identifying the signature algorithm used.
     - The XML character data T'.
   - The child element <data> (implicitly qualified by the "urn:xmpp:signed:0" namespace) as follows:
     - The XML character data E'.
9. Sends the <signed> 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.

2.1 Example of Signing Messages

The sender begins with the cleartext version of the <message/> stanza "S":

```
<message     xmlns='jabber:client'
             from='juliet@capulet.net/balcony'
             id='183ef129'
             to='romeo@montague.net'
```

[6] RFC 4648: The Base16, Base32, and Base64 Data <http://tools.ietf.org/html/rfc4648>.

```
            type='chat'>
    <thread>8996aef0-061d-012d-347a-549a200771aa</thread>
    <body>Wherefore art thou, Romeo?</body>
</message>
```

The sender then performs the steps 1 through 4 from above to generate:

```
<plain  xmlns="urn:xmpp:signed:0"
          timestamp="2010-06-29T02:15:21.012Z">
     PG1lc3NhZ2UgeG1sbnM9ImphYmJlcjpjbGllbnQiIGZyb209Imp1bGlldEBjYXB
     1bGV0Lm5ldC9iYWxjb255IiB0bz0icm9tZW9AbW9udGVndWUubmV0IiB0eXBlPS
     JjaGF0Ij48dGhyZWFkPmM2MzczODI0LWEzMDctNDBkZC04ZmUwLWJhZDZlNzI5O
     WFkMDwvdGhyZWFkPjxib2R5PldoZXJlZm9yZSBhcnQgdGhvdSwgUm9tZW8/PC9i
     b2R5PjwvbWVzc2FnZT4=
   </plain>
```

And then performs steps 5 through 9 steps, causing the following to be sent:

```
<message  xmlns='jabber:client'
            from='juliet@capulet.net/balcony'
            id='6410ed123'
            to='romeo@montague.net'
            type='chat'>
    <signed   xmlns="urn:xmpp:signed:0">
      <signature    algorithm="RSA-SHA1">
              DxbxIziY1C1Ytcxkj0IFLsfmDLMv96JMlMAQZ7jh49IbsOIPsxI2LyLmqhKH
                 /994UXDJKQLHvLJz
        gAmw8V2b+zmyZeItJzSmB+
           HHiLFVXkD2Dd4JfetsafsfIcB7uNWg0gAeiKrTHfFgiyEC/2WxwOj3
        JUMRyQ9ykEPIzS0GZ/k=
      </signature>
      <data>
        PHBsYWluIHhtbG5zPSJ1cm46eG1wcDpzaWduZWQ6MCIgdGltZXN0YW1wPSIyMDEwLTA2LTI5VDAy

        OjE1OjIxLjAxMloiPgogIFBHMWxjM05oWjJVZ2VHMXNibk05SW1waFltSmxjanBqYkdsbGJuUWlJ

        R1p5YjIwOUltcDFiR2xsZEVCallYQgogIDFiR1YwTG01bGRDOWlZV3hqYjI1NUlpQjBiejBpY205

        dFpXOUFiVzl1ZEdWbmRXVXVibVYwSWlCMGVYQmxQUwogIEpqYUdGMElqNDhkR2h5WldGa1BtTTJN

        emN6T0RJMExXRXpNRGN0TkRCa1pDMDRabVV3TFdKaFpEWmxOekk1TwogIFdGa01Ed3ZkR2h5WldG

        a1BqeGliMlI1UGxkb1pYSmxabTl5WlNCaGNuUWdkR2h2ZFN3Z1VtOXRaVzgvUEM5aQogIGIyUjVQ

        and2YldWemMyRm5aVDQ9CjwvcGxhaW4+Cg==
      </data>
    </signed>
  </message>
```

2.2 Example of Securing IQs

To be added....

3 Interaction with Stanza Semantics

The following limitations and caveats apply:

- Undirected <presence/> stanzas SHOULD NOT be signed.
- Stanzas directed to multiplexing services (e.g. multi-user chat) SHOULD NOT be signed, unless the sender has established the service supports the handling of signed stanzas.

4 Handling of Inbound Stanzas

Several scenarios are possible when an entity receives an encrypted stanza:

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

Case #2: The receiving application understands the protocol and is able to verify the signature.

Case #3: The receiving application understands the protocol and is able to verify the signature, but the timestamps fail the checks specified under Checking of Timestamps.

Case #4: The receiving application understands the protocol and is unable to verify the signature.

In Case #1, the receiving application MUST do one and only one of the following: (1) ignore the <signed/> extension, (2) ignore the entire stanza, or (3), except where precluded by the protocol (RFC 6120 [7]), return a <service-unavailable/> error to the sender.
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, except where precluded by the protocol, return a <not-acceptable/> error to the sender, optionally supplemented by an application-specific error condition element of <bad-timestamp/> as shown below:

```
<message from='romeo@example.net/orchard'
         id='6410ed123'
         to='juliet@capulet.net/balcony'
         type='error'>
```

[7] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

```
  <signed xmlns='urn:xmpp:signed:0'>
    <!-{}- original content -{}->
  </signed>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-timestamp xmlns='urn:xmpp:signed:0'/>
  </error>
</message>
```

In Case #4, the receiving application SHOULD, except as precluded by the protocol, return a <bad-request/> error to the sender, optionally supplemented by an application-specific error condition element of <bad-signature/> as shown below:

```
<message from='romeo@example.net/orchard'
         id='6410ed123'
         to='juliet@capulet.net/balcony'
         type='error'>
  <signed xmlns='urn:xmpp:signed:0'>
    <!-{}- original content -{}->
  </signed>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-signature xmlns='urn:ietf:params:xml:xmpp-signed:0'/>
  </error>
</message>
```

Additionally in Case #4, the receiving application 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).

5 Inclusion and Checking of Timestamps

Timestamps are included to help prevent replay attacks. All timestamps MUST conform to DATETIME [8] and be presented as UTC with no offset, always including the seconds and fractions of a second to three digits (resulting in a datetime 24 characters in length). 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, except as described below for offline messages.

[8] RFC 3339: Date and Time on the Internet Timestamps <http://tools.ietf.org/html/rfc3339>.

- If the foregoing check fails, the timestamp SHOULD be presented to the receiving entity (human or application) marked with descriptive text indicating "old timestamp" or "future timestamp" and the receiving entity MAY return a stanza error to the sender (except as precluded in the protocol).

The foregoing timestamp checks assume that the recipient is online when the message is received. However, if the recipient is offline then the server will probably store the message for delivery when the recipient is next online (offline storage does not apply to <iq/> or <presence/> stanzas, only <message/> stanzas). As described in Best Practices for Handling Offline Messages (XEP-0160) [9], when sending an offline message to the recipient, the server SHOULD include delayed delivery data as specified in Delayed Delivery (XEP-0203) [10] so that the recipient knows that this is an offline message and also knows the original time of receipt at the server. In this case, the recipient SHOULD verify that the timestamp received in the encrypted message is within five minutes of the time stamped by the recipient's server in the <delay/> element.

6 Mandatory-to-Implement Cryptographic Algorithms

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

- TBD (RSA/SHA1? RSASSA-RKCS1-v1_5? RSASSA-PSS?)

7 Certificates

To participate in end-to-end signing using the methods defined in this document, a client needs to possess an X.509 certificate. It is expected that many clients will generate their own (self-signed) certificates rather than obtain a certificate issued by a certification authority (CA). In any case the certificate MUST include an XMPP address that is represented using the ASN.1 Object Identifier "id-on-xmppAddr" as specified in Section 5.1.1 of RFC 3920bis.

8 Security Considerations

TBD.

[9] XEP-0160: Best Practices for Handling Offline Messages <https://xmpp.org/extensions/xep-0160.html>.
[10] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.

9 XMPP Registrar Considerations

9.1 XML Namespace Name for Signed Data in XMPP

A URN sub-namespace of signed content for the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:xmpp:signed

Specification: ProtoXEP

Description: This is an XML namespace name of signed content for the Extensible Messaging and Presence Protocol as defined by ProtoXEP.

Registrant Contact: XSF

10 Acknowledgements

This document borrows ideas and text from End-to-End Object Encryption ”work in progress” by Matthew Miller and Peter Saint-Andre.